

## Swansea University E-Theses

---

### Drawing from calculators.

Thimbleby, Will

---

#### How to cite:

Thimbleby, Will (2010) *Drawing from calculators..* thesis, Swansea University.  
<http://cronfa.swan.ac.uk/Record/cronfa43088>

---

#### Use policy:

This item is brought to you by Swansea University. Any person downloading material is agreeing to abide by the terms of the repository licence: copies of full text items may be used or reproduced in any format or medium, without prior permission for personal research or study, educational or non-commercial purposes only. The copyright for any work remains with the original author unless otherwise specified. The full-text must not be sold in any format or medium without the formal permission of the copyright holder. Permission for multiple reproductions should be obtained from the original author.

Authors are personally responsible for adhering to copyright and publisher restrictions when uploading content to the repository.

Please link to the metadata record in the Swansea University repository, Cronfa (link given in the citation reference above.)

<http://www.swansea.ac.uk/library/researchsupport/ris-support/>

# Drawing from Calculators

Will Thimbleby MEng

Submitted to Swansea University in  
fulfilment of the requirements for the  
Degree of Doctor of Philosophy



**Swansea University**  
**Prifysgol Abertawe**

September 2010

ProQuest Number: 10821480

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10821480

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code  
Microform Edition © ProQuest LLC.

ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 – 1346

# Drawing from calculators

Will Thimbleby  
PhD Thesis, Swansea University, 2010

## Summary

Two novel interactive systems, a new calculator and a new drawing program, are developed. The novel user interfaces derive from the application and development of design principles during the software development. It is the principles, their relationship to the development process, and their potential future role in interactive system development, that form the main contributions of the thesis.

Each system was created using an iterative, principle-driven method, in which the principles and implementation built on each other. The principle-driven design process led to original user interfaces and to refined principles. The design, development and underlying principles of each system form two complementary parts of the thesis:

- The calculator is designed to work as though it is “paper with answers.” The user can write any mathematical expression by hand, and the calculator recognises the written expression, then morphs the user’s input to a neat typeset expression, corrects any syntax errors, and then provides an answer. The neat typeset expression can then be edited freely by direct manipulation or by adding further writing.
- The vector graphics drawing program design follows a similar principle-driven approach. It applies the principles developed with the calculator, but to a very different style of user interface.

Both systems provide substantial examples of user interface design and development. Their design and development resulted in four key user interface principles: *projection*, *continuity*, *what you see is what you edit*, and *declarative interaction*. These four *flow* principles are, it is argued, the main reasons the user interfaces are effective.

User studies, qualitative feedback, heuristic, and analytic evidence is provided for the user interfaces. Both systems have been well received by users and are commercially distributed.

The design principles may support future user interface design and development. They provide further research opportunities, particularly in exploring exactly where they are applicable, and how and when they can be applied to future designs.



# Acknowledgements

This thesis would not have been finished without the encouragement, support and goading I received from many people. I appreciate all their effort even if it appeared otherwise at the time.

Both the calculator and Lineform are now commercial products, and both publishers deserve my thanks for their enthusiasm and support in creating two quality products. Freeverse, who published Lineform, and enabled many people to have the chance of using it. Also thanks are due to Promethean, especially for their enthusiasm about the calculator when it seemed that it would languish never used, and their publishing of that program. Promethean have made it possible for the calculator to end up in many schools, to be used to teach and explore mathematics and to hopefully inspire adults and children alike.

Harold Thimbleby my father and supervisor, who kept me from doing too many other things. He not only provided some of the initial thoughts that inspired the direction of this thesis but also provided me with my initial experiences of computers and a belief in what I could achieve. (All relevant previous work is properly cited as appropriate throughout the body of the thesis.) Thanks to rest of the FIT lab staff for the good conversations, particularly Matt Jones, George Buchanan and Parisa Eslambolchilar.

Microsoft Research for getting me to write my thesis, while pretending I was working for them, and specifically Richard Harper who provided much good advice and direction. Without whom I might still be contemplating starting to write.

Many other HCI peers have also provided good advice that this thesis would have been poorer without, especially Michael Harrison, Matt Jones and Paul Cairns.

My thanks also to my family, friends and fellow PhD students at Swansea. Specifically Emily Bridge, Ben Spencer, Will Harwood (whose pictures I've used in places) and Emily Thimbleby.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Abstract . . . . .	1
1.2	Introduction . . . . .	2
1.2.1	Contributions, principles and evaluation . . . . .	2
1.2.2	Structure of the thesis . . . . .	3
1.3	Calculators . . . . .	3
1.3.1	Typical use . . . . .	4
1.3.2	Calculator design principles . . . . .	5
1.3.3	Evaluation . . . . .	6
1.3.4	Calculator summary . . . . .	6
1.4	Drawing . . . . .	7
1.4.1	Drawing design principles . . . . .	7
1.4.2	Drawing evaluation . . . . .	8
1.4.3	Drawing summary . . . . .	9
1.5	Opening new research questions . . . . .	9
1.6	Publications and outputs related to this thesis . . . . .	10
1.7	Conclusions . . . . .	10
1.8	Bibliography . . . . .	11
<b>I</b>	<b>Calculating</b>	<b>13</b>
<b>2</b>	<b>Context</b>	<b>15</b>
2.1	History . . . . .	15
2.2	Pen and paper . . . . .	17
2.3	Calculators . . . . .	18
2.4	History of computer user interfaces for mathematics . . . . .	18
2.5	Types of mathematical user interfaces . . . . .	22
2.5.1	Linear user interfaces . . . . .	23
2.5.2	Template-based user interfaces . . . . .	25
2.5.3	Visual methods . . . . .	26
2.6	Summary . . . . .	29
<b>3</b>	<b>Design &amp; development</b>	<b>31</b>
3.1	The development process . . . . .	32

3.1.1	Like paper . . . . .	32
3.1.2	Affordance . . . . .	34
3.1.3	Feedback . . . . .	34
3.1.4	Morphing . . . . .	35
3.1.5	Editing . . . . .	36
3.1.6	Drag and drop . . . . .	36
3.1.7	Erasing . . . . .	37
3.1.8	Computing . . . . .	37
3.1.9	Partial expressions . . . . .	38
3.1.10	Storage . . . . .	39
3.1.11	Undo . . . . .	39
3.1.12	Teaching applications . . . . .	40
3.2	Summary . . . . .	41
<b>4</b>	<b>Principles</b>	<b>43</b>
4.1	Principles . . . . .	43
4.2	Projection . . . . .	44
4.2.1	User interfaces . . . . .	46
4.2.2	Multiple views . . . . .	47
4.2.3	Projected user interfaces . . . . .	48
4.2.4	Visibility of system status . . . . .	49
4.2.5	Similar concepts . . . . .	49
4.2.6	Editing . . . . .	49
4.2.7	Key concepts . . . . .	50
4.2.8	Example: Internet search . . . . .	50
4.2.9	Example: Calculators . . . . .	52
4.3	Continuity . . . . .	52
4.3.1	Animation . . . . .	53
4.3.2	Key concepts . . . . .	54
4.3.3	Example: Calculators . . . . .	55
4.4	What you see is what you edit . . . . .	55
4.4.1	Different models . . . . .	56
4.4.2	Modes and hidden state . . . . .	57
4.4.3	Key concepts . . . . .	58
4.4.4	Example: Syntax-directed editors . . . . .	58
4.4.5	Example: Template editors . . . . .	58
4.4.6	Example: Calculators . . . . .	59
4.5	Declarative interaction . . . . .	60
4.5.1	User interfaces . . . . .	61
4.5.2	Similar concepts . . . . .	62
4.5.3	Difficulties . . . . .	63
4.5.4	Key concepts . . . . .	64
4.5.5	Example: Calculators . . . . .	64
4.5.6	Predictable . . . . .	65
4.5.7	Refining . . . . .	65
4.6	Synergy . . . . .	66

---

4.6.1	Flow . . . . .	66
4.6.2	Reducing the user's cognitive workload . . . . .	67
4.6.3	Error recovery . . . . .	67
4.6.4	Breaking principles . . . . .	68
4.7	Summary . . . . .	68
<b>5</b>	<b>User interface overview</b>	<b>71</b>
5.1	Overview . . . . .	71
5.2	User interface sections . . . . .	73
5.3	Simple mathematics entry . . . . .	74
5.4	Editing . . . . .	77
5.5	Drag and drop . . . . .	77
5.6	Deletion . . . . .	81
5.7	Partial expressions . . . . .	82
5.8	Hiding the answers . . . . .	85
5.9	The dock . . . . .	86
5.10	History . . . . .	88
5.11	User interface concepts . . . . .	89
5.11.1	Handwriting . . . . .	89
5.11.2	Morphing . . . . .	89
5.11.3	Feedback . . . . .	90
5.11.4	Freeform editing . . . . .	90
5.11.5	Calculation . . . . .	90
5.11.6	Exploration . . . . .	91
5.12	Summary . . . . .	91
<b>6</b>	<b>Implementation</b>	<b>93</b>
6.1	Tools . . . . .	93
6.2	Overview . . . . .	94
6.3	Symbol recognition . . . . .	96
6.3.1	Character models . . . . .	97
6.3.2	Segmentation . . . . .	98
6.3.3	Recognition . . . . .	101
6.4	Expression recognition . . . . .	101
6.4.1	Some difficulties . . . . .	102
6.4.2	Alternative solutions . . . . .	102
6.4.3	The solution . . . . .	103
6.4.4	Structure specification schemes . . . . .	103
6.4.5	The algorithm . . . . .	105
6.4.6	Exponentiation . . . . .	106
6.4.7	Missing components . . . . .	107
6.5	Calculation . . . . .	108
6.5.1	Implementation . . . . .	109
6.6	User Interface . . . . .	112
6.6.1	Interaction . . . . .	113
6.6.2	Pen-based interaction . . . . .	114

6.6.3	Expressions and ink editing . . . . .	114
6.6.4	Selection . . . . .	115
6.6.5	Drag and drop . . . . .	115
6.6.6	Undo . . . . .	116
6.6.7	The dock . . . . .	117
6.7	Summary . . . . .	118
<b>7</b>	<b>Evaluation</b>	<b>119</b>
7.1	Evaluation in HCI . . . . .	119
7.1.1	Creating or iterating . . . . .	120
7.2	Evaluation in this thesis . . . . .	121
7.3	Declarative calculators . . . . .	122
7.4	Pen-based mathematics . . . . .	122
7.5	Initial evaluation . . . . .	123
7.5.1	User studies . . . . .	124
7.5.2	Results . . . . .	125
7.5.3	Time on task . . . . .	126
7.5.4	Ease of use . . . . .	127
7.5.5	Accuracy . . . . .	128
7.5.6	Summary . . . . .	129
7.6	Royal Society evaluation . . . . .	130
7.6.1	Visitors . . . . .	131
7.6.2	Results . . . . .	132
7.6.3	Quotes . . . . .	133
7.6.4	Mumbai, India . . . . .	136
7.6.5	Summary . . . . .	136
7.7	A comparison with <i>xThink</i> . . . . .	138
7.7.1	Specific differences . . . . .	139
7.7.2	Worked example . . . . .	142
7.8	Cognitive Dimensions evaluation . . . . .	145
7.9	A note on the philosophy of science . . . . .	147
7.10	Summary . . . . .	148
<b>II</b>	<b>Drawing</b>	<b>149</b>
<b>8</b>	<b>Context</b>	<b>151</b>
8.1	History . . . . .	151
8.2	Drawing applications . . . . .	153
8.3	Drawing in vector applications . . . . .	153
8.3.1	What are vector graphics? . . . . .	153
8.3.2	Why use vector graphics? . . . . .	154
8.3.3	Technical reasons . . . . .	155
8.3.4	Where do they come from? . . . . .	155
8.3.5	Semantic requirements . . . . .	155
8.4	User interface requirements . . . . .	156
8.4.1	Graphics . . . . .	156

---

8.4.2	Bézier splines . . . . .	156
8.4.3	Tools . . . . .	157
8.4.4	Selection . . . . .	158
8.4.5	Editing . . . . .	158
8.4.6	Navigating . . . . .	159
8.4.7	Style . . . . .	159
8.4.8	Compositing . . . . .	159
8.4.9	Filters . . . . .	159
8.4.10	Documents . . . . .	160
8.5	Poor user interface design . . . . .	160
8.5.1	Direct manipulation . . . . .	160
8.5.2	Modes . . . . .	162
8.5.3	Rigid design . . . . .	164
8.5.4	Over complication . . . . .	165
8.5.5	Lack of immediacy . . . . .	165
8.6	Summary . . . . .	166
<b>9</b>	<b>Design &amp; development</b>	<b>167</b>
9.1	Motivation . . . . .	168
9.2	Initial design . . . . .	169
9.3	Continued development . . . . .	170
9.3.1	Apple Design Awards . . . . .	170
9.3.2	Initial user feedback . . . . .	171
9.3.3	Commercialisation . . . . .	172
9.3.4	User feedback . . . . .	173
9.4	Flow principles . . . . .	174
9.4.1	Projection . . . . .	174
9.4.2	Continuity . . . . .	176
9.4.3	What You See Is What You Edit . . . . .	176
9.4.4	Declarative interaction . . . . .	177
9.5	Lineform principles . . . . .	178
9.5.1	Physical modes . . . . .	178
9.5.2	Flexible design . . . . .	178
9.5.3	Appropriate controls . . . . .	179
9.6	Summary . . . . .	180
<b>10</b>	<b>Principles</b>	<b>181</b>
10.1	Physical modes . . . . .	181
10.1.1	Lineform . . . . .	183
10.1.2	Recall . . . . .	184
10.1.3	Disadvantages . . . . .	185
10.1.4	Key concepts . . . . .	185
10.2	Flexible design . . . . .	186
10.2.1	Key concepts . . . . .	187
10.3	Appropriate controls . . . . .	187
10.3.1	Key concepts . . . . .	189

---

10.4	Other principles . . . . .	189
10.4.1	Direct manipulation . . . . .	189
10.4.2	Simplicity . . . . .	190
10.4.3	Well defined roles . . . . .	190
10.5	Summary . . . . .	191
<b>11</b>	<b>User interface overview</b>	<b>193</b>
11.1	The interface . . . . .	193
11.1.1	The toolbar . . . . .	193
11.1.2	The status bar . . . . .	194
11.1.3	Inspectors . . . . .	194
11.1.4	The media browser . . . . .	194
11.1.5	Keyboard . . . . .	194
11.2	Manipulating the canvas . . . . .	195
11.2.1	Zoom . . . . .	195
11.2.2	Drag . . . . .	195
11.3	Creating graphics . . . . .	196
11.3.1	The brush tool . . . . .	196
11.3.2	The pen tool . . . . .	196
11.3.3	The rectangle and oval tool . . . . .	196
11.3.4	The text tool . . . . .	197
11.4	Manipulating graphics . . . . .	197
11.4.1	Selecting graphics . . . . .	197
11.4.2	Moving, scaling and rotating . . . . .	198
11.4.3	Transform inspector . . . . .	199
11.4.4	Transforming with the keyboard . . . . .	200
11.4.5	Align and distribute . . . . .	200
11.4.6	Flip . . . . .	200
11.5	Layers and Z-order . . . . .	200
11.5.1	Isolation mode . . . . .	201
11.6	Groups and combining . . . . .	202
11.6.1	Clipping . . . . .	203
11.6.2	Combining . . . . .	203
11.7	Style . . . . .	203
11.7.1	Fill . . . . .	204
11.7.2	Solid . . . . .	205
11.7.3	Image . . . . .	205
11.7.4	Gradient . . . . .	205
11.7.5	Text . . . . .	206
11.7.6	The text inspector . . . . .	207
11.8	Stroke . . . . .	207
11.8.1	Arrows . . . . .	208
11.8.2	Artistic strokes . . . . .	209
11.8.3	Pressure . . . . .	210
11.8.4	Custom artistic strokes . . . . .	210
11.8.5	Text . . . . .	211

---

11.9 Effects . . . . .	212
11.10 Filters . . . . .	212
11.10.1 Filter resolution . . . . .	214
11.11 Editing graphics in depth . . . . .	214
11.11.1 Rectangles and ovals . . . . .	215
11.11.2 Bézier paths . . . . .	215
11.11.3 Boolean operations . . . . .	217
11.11.4 Outline . . . . .	218
11.12 The canvas in depth and exporting . . . . .	218
11.12.1 Rulers, guides and grid . . . . .	218
11.12.2 Page layout . . . . .	219
11.12.3 CMYK preview . . . . .	220
11.12.4 Outline view . . . . .	220
11.12.5 Export . . . . .	221
11.12.6 SVG . . . . .	221
11.12.7 AppleScript . . . . .	222
11.13 Summary . . . . .	222
<b>12 Implementation</b>	<b>223</b>
12.1 Document model structure . . . . .	223
12.1.1 Drawing model . . . . .	224
12.1.2 The document . . . . .	224
12.1.3 Graphics . . . . .	225
12.1.4 Groups . . . . .	225
12.1.5 Fill . . . . .	225
12.1.6 Stroke . . . . .	226
12.2 User interface . . . . .	227
12.3 Representations . . . . .	228
12.3.1 Semantics . . . . .	229
12.3.2 Comparisons . . . . .	230
12.4 Other features . . . . .	231
12.4.1 Linkback . . . . .	231
12.4.2 Scripting . . . . .	231
12.4.3 Core Image . . . . .	233
12.5 Summary . . . . .	233
<b>13 Evaluation</b>	<b>235</b>
13.1 Why expert reviews? . . . . .	235
13.2 User reaction . . . . .	236
13.3 Expert reviews . . . . .	237
13.4 Apple Design Award . . . . .	239
13.5 Commercial success . . . . .	240
13.6 Cognitive evaluation . . . . .	240
13.7 Summary . . . . .	242
<b>14 Conclusions</b>	<b>245</b>
14.1 Contributions . . . . .	245



---

14.1.1	Utility . . . . .	246
14.1.2	Validity . . . . .	246
14.2	Principles . . . . .	247
14.2.1	Projection . . . . .	247
14.2.2	Continuity . . . . .	248
14.2.3	WYSIWYE — What you see is what you edit . . . . .	248
14.2.4	Declarative interaction . . . . .	249
14.2.5	Physical modes . . . . .	250
14.2.6	Flexible design . . . . .	250
14.2.7	Appropriate controls . . . . .	250
14.3	Further work . . . . .	251
14.3.1	Computer science . . . . .	251
14.3.2	Human computer interaction . . . . .	253
14.3.3	Interactive editing . . . . .	254
14.3.4	Teaching . . . . .	254
14.3.5	Learning . . . . .	255
14.4	Summary . . . . .	255
<b>A</b>	<b>Anonymous questionnaire</b>	<b>267</b>
<b>B</b>	<b>Initial results</b>	<b>271</b>
<b>C</b>	<b>Royal Society — Briefing notes</b>	<b>277</b>
<b>D</b>	<b>Royal Society — Evaluation form</b>	<b>283</b>
<b>E</b>	<b>Royal Society — Results</b>	<b>285</b>
<b>F</b>	<b>Calculator manual</b>	<b>299</b>
<b>G</b>	<b>Dock equations</b>	<b>309</b>
<b>H</b>	<b>Recdit draft paper &amp; timeline of chapters</b>	<b>311</b>
<b>I</b>	<b>Published papers</b>	<b>327</b>
I.1	A Novel Pen-based Calculator and Its Evaluation . . . . .	327
I.2	A Novel Gesture-based Calculator and its Design Principles . . . . .	332
I.3	Mathematical Mathematical User Interfaces . . . . .	337
I.4	Internalist and Externalist HCI . . . . .	355

# Chapter 1

## Introduction

This chapter is designed as a complete and self-contained version of the larger thesis. The chapter provides a brief summary of the work in the entire thesis.

### 1.1 Abstract

The user interfaces for a novel calculator and a drawing program are described. The design and the associated design principles of these systems form the two complementary parts of this thesis.

These two systems provide, what will be proposed, are novel user interfaces for the conventional tasks of calculating and drawing. It is the user interfaces that make the applications distinct and could account for their appeal to users and their success in the marketplace of discretionary use.

The novel interfaces have been created using the insights from the application and development of certain design principles. During the development of the applications these principles helped shape the designs, and they were themselves shaped and refined as they were used. It is these diverse principles, their relationship to the development process, and more generally their potential future role in application development, that is the substance of the thesis.

This thesis focuses on clarifying and specifying the principles as well as describing how they emerged, developed and were used. The purpose will be to articulate what these principles are, how they were used and how they can be used in future design.

## 1.2 Introduction

Both drawing and calculation are very old: humans have been drawing and calculating for over tens of thousands of years. Unsurprisingly, tools (in addition to fingers) which support these activities have existed for almost as long.

With the invention of computers, their use as a tool to aid drawing and calculation was an obvious step, and thus computer applications to support drawing and calculation have been used since the earliest computers. The initial primary use of the modern digital computer was mathematical computation. And as early as 1963, Ivan Sutherland's Sketchpad was one of the first interactive drawing applications for a computer.

The calculator and Lineform, a vector graphics drawing program, described in this thesis were designed in a principled way with original user interface features. These systems have been well received by users: the calculator was selected for exhibition at the UK's top science exhibition, the Royal Society's Summer Science Exhibition; the drawing program was awarded an Apple Design Award, an award that recognises the best and most innovative Macintosh software. Although part of a PhD research programme, both these systems are robust, commercial-quality pieces of software: the calculator has been used for teaching in schools; the graphics program has been used in professional design. Both are now being distributed and supported commercially. They have tens of thousands of active users.

### 1.2.1 Contributions, principles and evaluation

The main contribution of the thesis is the specific novel design and implementation of both systems, and also their principles and argued design rationale that may be applied to future user interface design. In summary this thesis makes distinctive contributions at several levels:

- The programs, description of the design process, and novel user interfaces are themselves contributions, and are available to commercial standards.
- The principles can be used in other design processes.
- Features of the programs can be used in other systems.
- This thesis opens new research questions and makes suggestions for further work (Section 1.5).
- Additionally, there are refereed publications and other forms of competitively reviewed outputs (Section 1.6) arising from the thesis work.

A key question for any contribution is its validity. Conventionally, contributions and claims are evaluated, at least if their value is not self-evident.

Unfortunately, it is not possible to directly evaluate a principle, only its expression in the systems whose design it informed, and even then there may be other factors to control for. For example, only one programmer — the thesis author — was involved in the software development described in this thesis. It is also possible that *any* reasonable principle would improve design, perhaps because of the structure it provides. In an ideal world, one might use double-blinded reverse-result experiments (i.e., participants do not know the purpose of the principles they are given, but the experimental design has also included principles intended to make user interfaces *worse*), however, this is beyond what this thesis attempts.

The principles this thesis explores have not directly been subject to any empirical testing outside of their impact on the design of the particular applications. Of course, empirical user-based evaluation is only one form of evidence that can be recruited to argue successful research, though it is one which is very widely used; other methods include expert inspection, cognitive walkthrough, and so forth. However, as well as presenting substantial exploratory evaluation, this thesis will *argue* that the principles are an important aspect of the design of the novel user interfaces described, and further, their use in future user interface design has potential to lead to other novel and easy to use interfaces. Future studies might provide further evaluation of how the principles express themselves in other user interface styles (e.g., text-intensive, mobile, walk-up-and-use, CSCW), how the principles are used by other programmers, and so on.

### 1.2.2 Structure of the thesis

This thesis is split into two parts, which mirror each other. Each part in turn describes the context, design and development, principles, user interface, implementation and evaluation of the two different programs. The calculator is described first, and Lineform, which builds on some of the calculator's principles follows. This thesis then concludes with a summary of the contributions and the new opportunities for further work.

## 1.3 Calculators

Imagine writing a calculation down on paper and the paper magically working out the answers. The paper recognises your handwriting and you write naturally, using the ordinary mathematical notation you are already familiar with. The new calculator works like this and provides a user interface for pen-based interaction; or for interactive whiteboard use, for instance in lectures or classrooms.

The calculator, first described in [Thimbleby, 2004], provides a natural, dynamic method of entering conventional arithmetic expressions using hand-

Figure 1.1: Using the calculator

writing. It provides continual feedback showing the calculation and results. The user interface adjusts and copes with partial expressions, morphing the expressions to the correct position and result. Gestures are also used to edit and manipulate calculations. The actual interaction is very fluid, and is hard to convey well in a static, non-interactive paper format such as a PhD thesis. Video demonstrations of users interacting with the calculator and of the user interface are available on-line<sup>1</sup>.

### 1.3.1 Typical use

Figures 1.1.1—1.1.6 show a sequence of screen snapshots of the calculator in use. In the first screenshot, the user has written  $3 \times 4$  and the calculator is at the moment of the screenshot “catching up” with the user’s handwriting, and has just rendered the 3 in a typographically neat font.

In Figure 1.1.2, next, the calculator has morphed all the user’s input, and immediately combined it with the output (here, ‘=12’) and displayed it all as a typeset equation. The output generated from the calculator is shown in red, distinguishing it from the black of the user’s input. The user continues to edit the equation and by Figure 1.1.3, they have deleted the 4 and written ‘=18’. Effectively this poses the question “three times what is eighteen?” making the calculator compute “ $3 \times x = 18$ ”. Additionally in Figure 1.1.3, the user can be seen to continue to edit this solved equation as if it were their own input; the user has written underneath of the equation to divide the left hand side by 5.

By Figure 1.1.4 the calculator has morphed these changes and combined the typeset output and the user’s input into another neatly typeset equation, now showing a generated 30.

In Figures 1.1.5 and 1.1.6, the user “drag selects” the “ $3 \times$ ” from the previous screen and drags it below the division line. (This is an “ink edit”, the “ $3 \times$ ” is not a syntactically nor semantically meaningful unit.) Finally, Figure

<sup>1</sup><http://www.cs.swan.ac.uk/calculators/>

1.1.6 shows the result of this edit, and it is mathematically correct — thus providing a solution to “ $x/(3 \times 5) = 18$ ”.

### 1.3.2 Calculator design principles

The design of the calculator was based on the idea of a natural pen and paper user interface. As the calculator was created, principles developed that informed the design. As the design was advanced and built on, these principles were integrated deeper and refined through use and implementation, this process is described in Chapter 3. These principles, that both guided and evolved through the calculator’s design are described more fully in [Thimbleby and Thimbleby, 2005] and in Chapter 4. The main principles are summarised here.

- *Projection* — Changes to the system’s state are immediately visible everywhere. This expands on the term *projected editors* used by [Simonyi et al., 2006] and is similar to *tight coupling* as described by Ahlberg and Shneiderman [1994]. An important aspect of projected editing is that the input and output of the user interface can never be inconsistent. This means that the display of output data (e.g. the answer) has to be correct instantly without further user action.
- *Continuity* — Continuous feedback and morphing provide the continuity between state changes. The user is always provided with clear feedback [Shneiderman, 1992] about what is happening. For example, the user’s hand-written input is morphed into a typeset sum, this provides a clearer knowledge of the mathematics being calculated and how the output relates to the input.
- *What You See is What You Edit* (WYSIWYE) — Only what is visible in the user interface determines how system can be edited. A user is not forced to think syntactically about the structure of the mathematics, they edit the actual mathematics they see without constraint.
- *Declarative interaction* — There is no distinction between input and output [Runciman and Thimbleby, 1986]. The inputs and outputs of a user interface should not be rigid concepts. For example, it is possible to change the output (“ $3 \times x = 18$ ”) to find what input generates it.

The novel user interface the calculator provides is argued to derive from these principles. They are expanded and explored further in Chapter 4, and potentially open up much possible new, fruitful work in user interface design.

### 1.3.3 Evaluation

The results of a pilot user study of nine participants, using national mathematic exam questions for 16 year olds, have shown that users enjoy using the calculator and it can be faster at computing more complex sums like  $\frac{2^2}{21-4}$ . However the most interesting result from this study is that regardless of intermediate mistakes *no* user arrived at any incorrect answers, compared with several wrong answers from pocket calculators the users were familiar with.

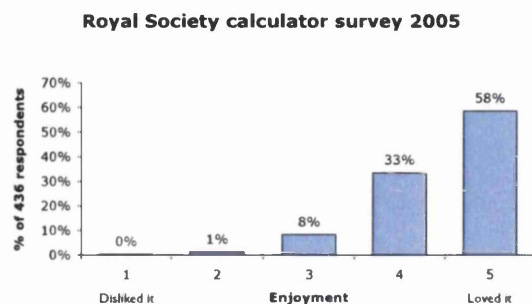


Figure 1.2: Royal Society evaluation: Enjoyment

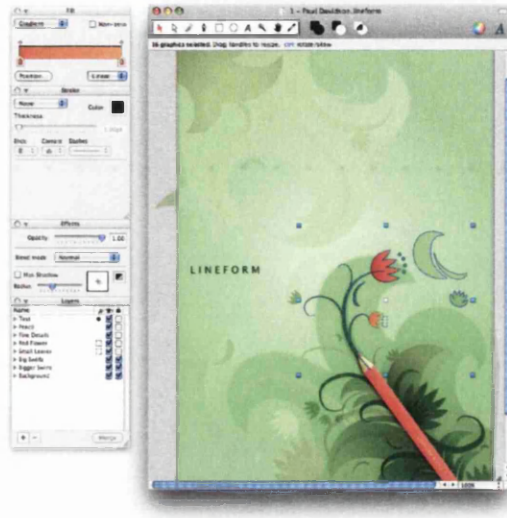
Several thousand people were also able to see and use the calculator at the Royal Society's 2005 Summer Science Exhibition, a public exhibition of the UK's top science. The 436 evaluation forms completed by users provided encouraging results. Figure 1.2 shows how users rated their enjoyment of using the calculator on a SMART interactive whiteboard.

The user feedback came from a wide range of occupations, education, backgrounds, and ages. The average age of the users who returned feedback forms was approximately 30, with an even split between male and female. 34% of respondents said they had problems with their current calculator or mathematical method.

### 1.3.4 Calculator summary

The calculator provides a novel user interface for a calculator and as an interface for enjoying and exploring mathematics. A longitudinal user study which would provide insight into the lasting success of the calculator has not yet been undertaken. However, the current studies provide good support for the calculator's design, especially aspects like exploring, teaching and accuracy. The reaction and enjoyment users get out of the interface is real. Many users have laughed, smiled and grinned whilst doing mathematics, even if they did not realise it, which is great for mathematics, which most people claim to hate! Underlying the calculator's design are the principles that shaped it, these provide useful ideas for future design.

## 1.4 Drawing



**Figure 1.3: Lineform’s user interface, editing a drawing.** (Drawing © Paul Davidson)

Lineform, shown in Figure 1.3, is a novel vector drawing application, similar in functionality to, and aimed at providing the capabilities of applications such as CorelDraw and Adobe Illustrator. Lineform was initially created to provide a drawing and illustration program for the author’s personal use and expanded into a commercial product that thousands of people use.

Lineform provides a different example of novel design; it was designed and built after the calculator and its design was informed by, and builds on the principles that came from the calculator’s development. During development, other principles specific to Lineform were also articulated and used to guide the design, Chapter 9 describes this process in more detail.

### 1.4.1 Drawing design principles

The primary influence from the calculator was the *projection* principle, but other principles also provide other design suggestions. How the calculator principles informed Lineform’s design is summarised below.

- *Projection* — All views of the drawing, whether the drawing itself or data in inspectors, are always consistent and always reflect the underlying drawing.
- *Continuity* — During any user interaction editing the drawing, the whole user interface updates immediately. No state changes are initiated without the user’s control, thus reducing any continuity problems.



- *What You See is What You Edit* — Is supported through various features that attempt to make the structure of a drawing more visible, like an outline mode. WYSIWYE also suggests future features similar to those that bitmap editors provide, such as vector based bucket-fills and erasers.
- *Declarative interaction* — All views, when possible, are editable. For example, the width in the *Transform* inspector both shows the selected graphics' width as output and allows editing which enables the exact width to be input.

The design and implementation of Lineform also led to some additional guiding principles.

- *Physical modes* — User interface modes should be controlled by what the user is physically doing [Sellen et al., 1992]. The reasoning is that continuous physical force makes a mode less likely to be forgotten (compare using the Shift-key to the Caps-lock key). Other approaches providing different forms of feedback [Monk, 1986] are also possible.
- *Flexible design* — Allow users to delay decisions until they are ready, and to easily change their mind. Lineform is designed so that any graphic can be easily repurposed in a different role, instead of being rigidly defined by how it was created.
- *Appropriate controls* — Discrete values should have discrete controls and continuous values should have continuous controls. The correct use of user interface controls allows both the easy exploration and exact setting of a value.

These ideas are described further in Chapter 10.

## 1.4.2 Drawing evaluation

Conventional user interface evaluation involves empirical work with users, (for instance as was undertaken with the calculator) or in some cases as expert evaluation, e.g., heuristic evaluation (though expert evaluation is used less often in research). These sorts of evaluation have not been undertaken with Lineform. Independent reviews by professional artists and users provide the majority of the user evaluation. Lineform also won an international design competition, focusing on innovation, user experience and technology, indicating that in some sense it can already be considered to provide a good user interface.

Lineform won the Mac OS X Student Apple Design Award and is published by Freeverse<sup>2</sup>; it has now been acquired by Apple. Lineform has been very well received by thousands of users and reviewers. The following typical quotes are provided to illustrate this.

---

<sup>2</sup><http://www.freeverse.com/lineform/>

Lineform has two other selling points. First, its speed: the program launches in a couple of seconds and shames Illustrator throughout in its responsiveness. Second, its ease of use. The simple interface alone makes it easier to find things.

— MacUser review (Oct 2006, vol 22 issue 22)

There is nothing that even comes close to this program for ease of use, adaptability and creative potential.

— Peter Marino (Amazon user review, May 2007)

### 1.4.3 Drawing summary

The development of Lineform drew heavily on the experience and development of principles from the calculator, previously described. It provides novel user interface concepts that have appealed to users and uses the ideas from the calculator in a very different type of user interface. The use of the principles in Lineform is an additional confirmation of the claims made of their value and of their potential for designing other applications with novel interfaces.

## 1.5 Opening new research questions

These two applications provide novel user interfaces and themselves provide many possible future research possibilities, such as extending the current designs. They also raise research questions about the principles, where these principles are valid or applicable, as well as how they can be applied to future user interface design. Obvious open-ended design questions include the following:

- How can the fluid correctness of the calculator be extended to more complicated maths, from simple algebra to completely different domains?
- Are the principles generally useful and where are they effective?
- How can the contrasting benefits of bitmap drawing be combined with vector drawing while retaining the principles such as WYSIWY?
- How can the calculator be extended to facilitate use by teachers in novel and interactive teaching methods?
- How can the calculator encourage exploration and learning?

## 1.6 Publications and outputs related to this thesis

Four publications which I have co-authored and which are related to this thesis are included as published in Appendix I. These publications cover the design, development, and principles of the calculator, and also thoughts on the different methods of thinking about interaction design.

The last paper in Appendix I introduces and contrasts *internalist* and *externalist* design. This thesis adopts a primarily internalist perspective, that is, one that broadly emphasises (although by no means exclusively) the design and principles of a systems rather than external perspectives such as user evaluation. The paper discusses the validity of this perspective and its relation to the wider interests of successful HCI design.

In addition to refereed papers, the research in this thesis has generated other types of output and recognition, including the following.

- The calculator, described in Part I has been exhibited at the Royal Society (at the Royal Society Workshop, “Rags to Riches,” 2004, and at the Royal Society Summer Science Exhibition 2005), at the Welsh National Eistedfodd 2006, at the National Waterfront Museum (2006 and 2007) and at Techfest 2008 (Bombay, India). It was also exhibited at the UK Parliamentary Young Engineers Competition in 2006, where it won the Vodafone Prize.
- The drawing program, Lineform, described in Part II, has been published as a successful retail and on-line commercial product by Freeverse. Lineform won the Apple Student Design Award, 2006, and has amassed considerable praise from users and reviewers, as described in Chapter 13. In 2008 Apple purchased the rights to Lineform.

## 1.7 Conclusions

Two novel user interfaces have been designed and their design and underlying principles articulated. A large number of people have used both applications and have provided positive feedback, these and other results support the claims of the quality of the design of the applications, and in association the utility of the underlying design principles that were used. Thinking about the design of a user interface in a principled way has been a successful strategy for these two systems. It’s hoped that the principles described in this thesis will be useful for future user interface design.

## 1.8 Bibliography

- [Ahlberg and Shneiderman, 1994] Christopher Ahlberg and Ben Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *CHI 94: Proceedings of the SIGCHI conference on Human factors in computing systems CHI*, pages 313–317, New York, NY, USA, 1994. ACM Press.
- [Monk, 1986] Andrew Monk. Mode errors: a user-centered analysis and some preventative measures using keying-contingent sound. *International Journal of Man-Machine Studies*, 24(4):313–327, 1986.
- [Runciman and Thimbleby, 1986] Colin Runciman and Harold Thimbleby. Equal opportunity interactive systems. *International Journal of Man-Machine Studies*, 25(4):439–451, 1986.
- [Sellen et al., 1992] Abigail J. Sellen, Gordon Kurtenbach, and William Buxton. Prevention of mode errors through sensory feedback. *Human-Computer Interaction*, 7(2):141–164, 1992.
- [Shneiderman, 1992] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1992.
- [Simonyi et al., 2006] Charles Simonyi, Magnus Christerson, and Shane Clifford. Intentional software. In *OOPSLA 06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 451–464, New York, NY, USA, 2006. ACM Press.
- [Thimbleby, 2004] Will Thimbleby. A novel pen-based calculator and its evaluation. In *NordiCHI 04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 445–448, New York, NY, USA, 2004. ACM Press.
- [Thimbleby and Thimbleby, 2005] Will Thimbleby and Harold Thimbleby. A novel gesture-based calculator and its design principles. In N. Bryan-Kinns L. MacKinnon, O. Bertelsen, editor, *Proceedings 19th. BCS HCI Conference*, volume 2, pages 27–32, 2005.



# **Part I**

## **Calculating**

## Chapter 2

## Context



### 2.1 History

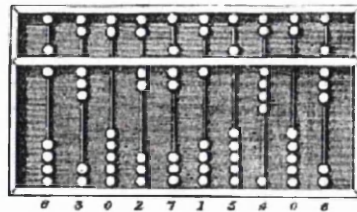


Figure 2.1: A Chinese abacus originating from around 200 BC

Calculation aids have been in existence for thousands of years. Abacuses, shown in Figure 2.1, are thought to have been invented around 200 BC,

and long before that bones and tally marks were used. The oldest known mathematical object, the Lebombo bone, a bone marked with tally notches is dated circa 35,000 BC. Amazingly, similar aids are still in use today. Tally marks are a common way of recording counting in most cultures and abacus arithmetic was still being taught in schools as late as the 1990s in Taiwan and still finds use in Asia. The virtually limitless precision (perhaps by utilising more than one abacus) more than compensates for the lack of more advanced features like trigonometric functions.

We have always developed instruments to aid our mental arithmetic and to help us with mathematics. A variety of other instruments have been devised over the centuries. In 1614 John Napier invented his “bones” and then his logarithms, providing tools to multiply and divide easily.

Slide rules were devised shortly afterwards, circa 1630 by Edmund Gunter, utilising logarithms to perform multiplication and division by addition and subtraction. Slide rules utilise the mathematical rules  $\log(xy) = \log(x) + \log(y)$  and  $\log(\frac{x}{y}) = \log(x) - \log(y)$ . These remained popular until the widespread use of electronic calculators.

In 1642, Blaise Pascal invented a mechanical adding machine, and in 1942 Gottfried Leibnitz constructed the first mechanical calculator capable of multiplication and division. Leibnitz’s methods formed the mainstay of calculating devices until the late nineteenth century.

Modern electronic calculators were introduced in the 1960s and became popular in the 1970s. The world’s first “handheld” battery operated calculator was the Sharp QT-8B, Figure 2.2, which provided an eight digit display and four functions (addition, subtraction, multiplication and division).



**Figure 2.2: Sharp QT-8B “micro Compet” — First handheld electronic calculator (1970). Source: Vintage Calculators**

Over thirty years later as part of the latest state-of-the-art operating system, the normal calculator looks much the same! Microsoft’s latest calculator included with Vista is shown in its basic mode in Figure 2.3, the user interface it provides is not far removed from the Sharp QT-8B.



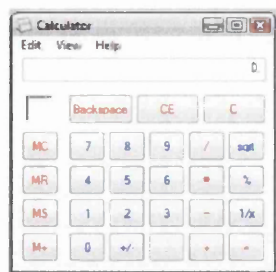


Figure 2.3: Microsoft Vista's Calculator (2008)

Of course, this is slightly unfair, modern computers and software can do an incredible amount of mathematical calculation and manipulation. Software packages like Mathematica, Figure 2.4, now provide many different powerful mathematical tools and aids.

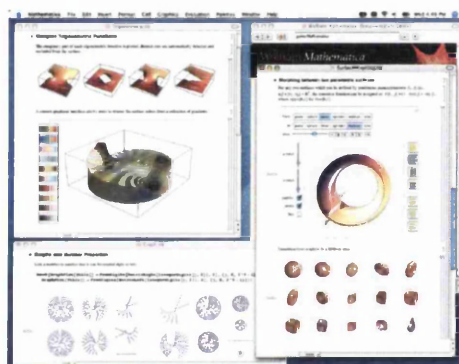


Figure 2.4: Mathematica (2007)

## 2.2 Pen and paper

A thread that is similar to that of the evolution of mathematical tools running through history is the thread of mathematical notation. However while mathematical notation has changed and its evolution through different ideas is fascinating [Cajori, 1993], the tools that mathematical notation is designed for have not changed. Pen and paper, or their equivalents, have been the tools of choice for mathematics for thousands of years and still are today.

Pen and paper as mathematical calculating aids, like every calculation aid, provide a physical representation of the abstract mathematical thoughts of the mind. Written mathematics is a notation that has been developed for thousands of years, and something that we are taught from early childhood. The discussion of the optimality of standard mathematical notation, in the age where computers can provide user interfaces only dreamt of before, is

$$K(Z, O_1, O_2) = \frac{Z^{2 \cdot k+1} \cdot \sum_{R=1}^k Z^R \cdot A_R \cdot \left( e^{\frac{2 \cdot R \cdot 1 \cdot O_1 \cdot e}{k}} + e^{\frac{2 \cdot R \cdot 1 \cdot O_2 \cdot e}{k}} \right)}{2 \cdot \left[ Z \cdot W(Z) - \text{ALPHA}(O_1, O_2) \right]}$$

$$W(Z) = e - e \cdot Z^k + 1$$

**Figure 2.5: Plotted expressions from the Symbolic Mathematical Laboratory [Martin, 1967b]**

an interesting one, but beyond the scope of this thesis. We are taught and use a two dimensional notation, designed for pen and paper.

Today, the ubiquitous tools of pen and paper are usually augmented with additional calculating aids, that actually perform calculations. In contrast to calculators, paper is not interactive. Answers to even simple sums have to be worked out in the user's head or using another calculation aid. This is the most common way in which basic mathematics are performed: pen and paper aided by a handheld calculator. These users of calculators are also thus hopefully competent with two dimensional mathematical notation.

## 2.3 Calculators

Many of the calculator functions that run on bitmapped workstations and personal computers are designed to simulate real handheld calculators. Such a design allows rapid transfer of skill to an otherwise unfamiliar situation. However, such a design is naïve, in that simulated calculators inherit all of the problems of real calculators and fail to exploit the opportunities for improvement that a graphics workstation provides.

— Johnson [1985] on the state of calculators in 1985

Jeff Johnson's statement above about calculators is for the most part still true.

## 2.4 History of computer user interfaces for mathematics

Computers have always been used for performing mathematical computations. Indeed, Ada Lovelace is widely thought of as the first programmer [Fuegi and Francis, 2003], having written a program in 1843 to calculate Bernoulli numbers for the never-completed Analytical Engine. The first

$$\begin{array}{r}
 3 \\
 2 + 3 \\
 \hline
 1 \\
 1 + - \\
 2
 \end{array}$$

Figure 2.6: A Macsyma rendering of  $\frac{2+3^3}{1+\frac{1}{2}}$

“modern” computer system for mathematics ran on batch processing systems, that took input as punched cards and provided output some time later. The use of computer user interfaces for general manipulation of mathematics developed later. In the 1960s two dimensional output was possible, and Magic Paper I [Clapp and Kain, 1963] was the first system to display two dimensional output: it used a typewriter for inputting mathematical expressions and a display scope or plotter for showing the typeset outputs. Minsky’s [1963] Mathscope proposal for manipulating mathematics was one of the first handling of mathematics from a user interface point of view. This proposal was built on by Martin [1967b], creating the Symbolic Mathematical Laboratory, which was capable of displaying normal mathematical notation using different fonts and special symbols, Figure 2.5 shows an example of the plotter output from the Symbolic Mathematical Laboratory. The same display of the mathematics was shown on a scope and basic interaction was possible using a light pen to select variables and operators.

Martin’s work was far ahead of many of the systems that followed. The majority of these displayed mathematical expressions using multiline text, like that shown in Figure 2.6. Engelman [1965] created MATHLAB in 1964, an early computer algebra systems for manipulating symbolic mathematics, which was a precursor to the commercial Macsyma [Martin and Fateman, 1971] and the actively developed open-source Maxima [Joyner, 2006] computer algebraic systems.

The Reduce pretty printer [Leler and Soiffer, 1985] followed in 1985, adding a powerful user interface to Reduce [Hearn, 1968] that allowed editing multiple expressions, mouse selection and subexpression collapsing. Young [1987] created GI/S in 1987: GI/S was the first system to allow the selection of mathematics unrelated to the underlying mathematical structure. The user could select any rectangular sequence of linear expressions, for example selecting  $a \times b +$  from the expression  $a \times b + c - d$ .

Milo [Avitzur, 1988] was developed for the Macintosh, originally as an aid for undergraduate physics homework. Milo combines text, expressions and plots within the same document. Milo included only a basic algebraic solver, but provided an easier to use interface than most other applications. Parts of Milo were embedded in FrameMaker and the original Macintosh graph-

$$y = ax + ab + 1 \quad y = ax + ab + 1$$

$$y = ax = ab + 1 \quad y = a \left( x + b + \frac{1}{a} \right)$$

$$-ax + y = ab + 1 \quad y \frac{1}{a} = x + b + \frac{1}{a}$$

Figure 2.7: Dragging terms in Graphing Calculator

ing calculator, that was included at one point with every Macintosh sold. Avitzur [1998] provides a good discussion of the user interface of the Graphing Calculator and of its predecessor. Milo allowed the selection of symbols by dragging a box over the subexpressions but no selection of operators. It maintained the correct mathematical syntax at all times, so that the mathematics would never be syntactically incorrect. Milo was also the first system to allow the direct manipulation of expressions; subexpressions could be dragged left or right and the expression around them was adjusted so that the moving of the expression did not alter the equality. This makes use of various mathematical laws such as the distributive law of multiplication over addition, and uses subtraction or division to move a subexpression across an equals sign. Figure 2.7 shows two examples of dragging subexpressions further to the left in the Graphing Calculator, as the subexpression is dragged a simple rewriting of the whole expression ensures that it maintains the same meaning.

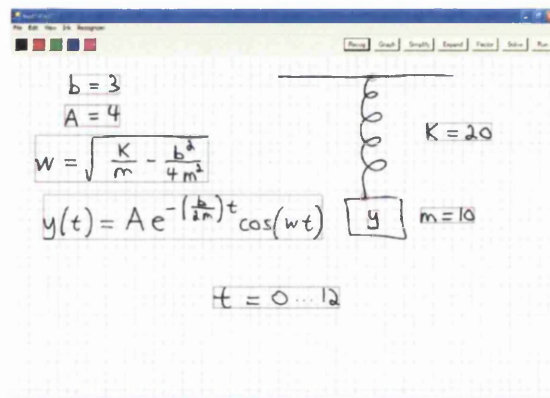
Although the pen is naturally suited to mathematical expression input, the complexity of recognising handwritten mathematics has meant that development of this as a user interface has been slower. Early work in handwriting recognition of mathematics was done by Anderson [1968] recognising type-set mathematics and Martin [1971] provides a good analysis of some of the difficulties.

Littin [1993] makes use of a modified 2D LR parser to handle mathematical expressions. This grammar requires symbols to be written in a particular sequence, thus restricting input and making editing nearly impossible. Grbavec and Blostein [1995] approached the same problem using a graph rewriting language. The graph consists of the symbols and their connections relating to spatial relationships, such as “below” and “left-of.” This is matched against templates and reduced into a full parse tree. Blostein and Schüerr [1999] and Lavirotte and Pottier [1997] built on this concept.

The Freehand Formula Entry System (FFES) is a complete system for formula entry and conversion to L<sup>A</sup>T<sub>E</sub>X [Smithies et al., 1999, Smithies, 1999]

that uses the same graph rewriting concept to recognise mathematics. This later used Diagram Recognition Application for Computer Understanding of Large Algebraic Expressions (DRACULAE) [Zanibbi et al., 2001, 2002] which implements a tree-transformation based approach for recognising the syntax and semantics of mathematical expressions.

Eto and Suzuki [2001] use minimal spanning trees to reconstruct the mathematical formula. OpenXM, the Open message eXchange protocol for Mathematics, is a communication protocol for various computer algebra systems, which has been used to provide online recognition of handwritten mathematical expressions for various mathematical software [Fujimoto and Suzuki, 2002].



**Figure 2.8:** A sketch in MathPad [LaViola and Zeleznik, 2004] exploring damped harmonic oscillation

MathPad [LaViola and Zeleznik, 2004] provides a unique approach to mathematical recognition: it allows the interaction of mathematics with sketches. MathPad provides the ability to link equations and drawings, such that the drawings animate. Figure 2.8 shows a sketch of a spring and mass system, the mathematics of the system are linked to the sketch so that it animates correctly. An initial evaluation of MathPad [LaViola, 2006] found users really enjoyed the interactivity and were forgiving of recognition accuracy but often failed to use implicit associations correctly.

Interactive pen-based systems like xThink<sup>1</sup> and Microsoft Math are more recent visual developments that use pen-based entry of mathematical expressions. Microsoft Math and its free component the Microsoft Equation Writer, shown in Figure 2.9, provide good all-round mathematical equation entry.

<sup>1</sup>[www.xthink.com](http://www.xthink.com)

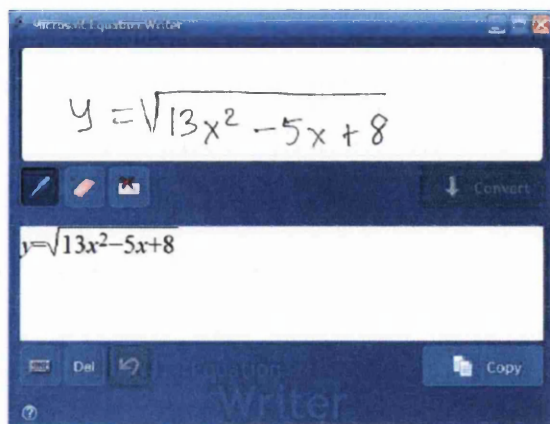


Figure 2.9: Screenshot of Microsoft Math

There are several problems with current computer algebra systems (CASs) that are interface-related. These problems include the use of an unnatural linear notation to enter and edit expressions, the inherent difficulty of selecting and modifying subexpressions with commands, and the display of large expressions that run off the screen.

— Kajler and Soiffer [1998]

Today punched cards are only of historical interest and there are many more flexible and powerful graphical user interfaces providing mathematical expression entry, manipulation and computation.

## 2.5 Types of mathematical user interfaces

User interfaces for mathematical entry and manipulation can be split into three main categories: linear, template-based and visual user interfaces. All of these user interfaces can be used to provide an interactive calculator; however, some are more suitable than others. Kajler and Soiffer [1998] provide a useful overview of the whole area of algebra entry concentrating on template-based entry systems, although their paper also covers pen and voice user interfaces.

- Linear user interfaces are typified by a need for mathematical expression entry as a linear sequence of commands.
- Template-based user interfaces build up a mathematical expression from building blocks.
- Visual user interfaces make use of computer vision techniques to read input as standard mathematical expressions.

### 2.5.1 Linear user interfaces

All cheap and simple pocket calculators require mathematical expressions to be entered as a simple linear sequence of digits and symbols or operators. Many advanced mathematical user interfaces for complex mathematical packages also rely on this sort of user interface, although programs like Mathematica [Wolfram, 1991] and Maple<sup>2</sup> have recently added template-based expression entry and aspects of visual methods.

The primary disadvantage of linear user interfaces is that mathematical notation is not linear, and users tend to think of, and treat mathematical expressions as they would write them on paper, as two-dimensional expressions. These expressions have to be converted from their two dimensional form, that the user has conceptually in their mind, to the linear form that the computer can understand. This is often done by adding lots of brackets and unusual symbols, such as  $\wedge$  which is used for exponentiation. This process of linearisation, taking the two-dimensional notation and converting it into a linear sequence of button presses, has to be performed by the user and is an additional cognitive burden. For example the unbracketed equation, Equation 2.1, is written in linear form as  $(2+3\wedge3)/(1+1/2)$ .

$$\frac{2 + 3^3}{1 + \frac{1}{2}} \quad (2.1)$$

#### Handheld calculators

In contrast to paper, a typical handheld calculator uses buttons and a small display. Some handheld calculators provide a formula and answer format on two lines, but even then the formula is written in a one-dimensional textual notation. In addition, most handheld calculators, because of the limitations of their small screen size, by necessity hide relevant information (such as the last number entered), and this makes them harder to use.

The simplest and most common calculators, often found in school classrooms, enforce further constraints. These calculators often have no concept of precedence or parentheses. These restrictions mean that for simple handheld calculators a simple equation such as Equation 2.1 is nearly impossible to calculate without the aid of paper. Equation 2.1 would have to be entered as  $\boxed{\text{MC}} \boxed{1} \boxed{\div} \boxed{2} \boxed{+} \boxed{1} \boxed{\text{M}+} \boxed{\text{C}} \boxed{3} \boxed{\times} \boxed{3} \boxed{\times} \boxed{3} \boxed{+} \boxed{2} \boxed{=} \boxed{\div} \boxed{\text{MR}}$ , which is a convoluted and awkward translation that the user is burdened with performing themselves, and only works with memory.

Furthermore, the majority of more complicated functions, or notations such as log or  $\int$ , often have unusual and strange input command sequences that have to be learnt.

<sup>2</sup><http://www.maplesoft.com/>

Unfortunately the design of common calculating aids, such as Microsoft's and Apple's calculator applications, has often been to emulate these real hand-held calculators and their difficult-to-use and restricted user interfaces.

### Reverse Polish Notation

A variation of the linear method of input is Reverse Polish Notation, RPN, or postfix notation. RPN is a notation where the operator is entered after the operands, RPN thus removes the need for parentheses. For example the expression  $4 \times (2 + 3)$  would be entered as `4 [Enter] 2 [Enter] 3 [+] [×]`. Equation 2.1 becomes `2 [Enter] 3 [Enter] 3 [↑] [+] 1 [Enter] 1 [Enter] 2 [÷] [+] [÷]`, which is shorter but places a larger cognitive burden on the user to convert the mathematics to the format the calculator understands.

### Proprietary packages

Mathematica [Wolfram, 1991] and Maple each provide their own proprietary format for entering expressions.

$$\int_0^{\infty} \frac{4x^3}{\log x} dx \quad (2.2)$$

An example of such a command sequences for calculating the integration shown in Equation 2.2 is shown below in the formatted for Maxima (an open source mathematical package) Mathematica, and Maple (both commercial packages), in this order.

- `integrate (4x**3/log(x), x, 0, inf);`
- `Integrate[4x^3/Log[x],{x,0,Infinity}]`
- `int(4x^3/ln(x), 0..infinity)`

Each one is different and the individual syntax of each package has to be individually remembered by each user. None of the syntax could be intuited from the actual mathematical expression, thus a user needs to learn each format.

All three of these programs also provide alternate means of entering expressions in a linear form, attempting to alleviate some of the complexity of entering expressions accurately and quickly. Whether this is a benefit overall is hard to say.

### Document processing

Unlike template and visual methods, linear user interfaces often provide output in a different form to the input. This is most commonly used to



convert a linear sequence of commands into a more readable two dimensional format. The early computer algebra systems achieved this through multi-line text output. Equation 2.1 rendered by Macsyma in this format would look like Figure 2.6.

Another common use of linear input for mathematics is document processing.  $\text{\TeX}$  [Knuth, 1984] is commonly used for typesetting mathematical documents. In fact Equation 2.1 is typeset using  $\text{\TeX}$  using the linear input `\frac{2+3^3}{1+\frac{1}{2}}`. Document processing programs are different from programs designed to manipulate mathematics, however, as they do not provide any mathematical computation and often provide multiple ways of typesetting the same formulae.

### 2.5.2 Template-based user interfaces

Template-based user interfaces are the most common mathematical user interfaces. They are simple to create and extend well to incorporate a wide range of mathematical notation without any difficulty. A template editor has been a part of Microsoft Word since 1993 [Microsoft, 1993], and many computer algebra systems such as Mathematica now provide template interfaces. LyX [Quill, 1999] provides a similar template-based interface for mathematical expression entry for  $\text{\LaTeX}$ .

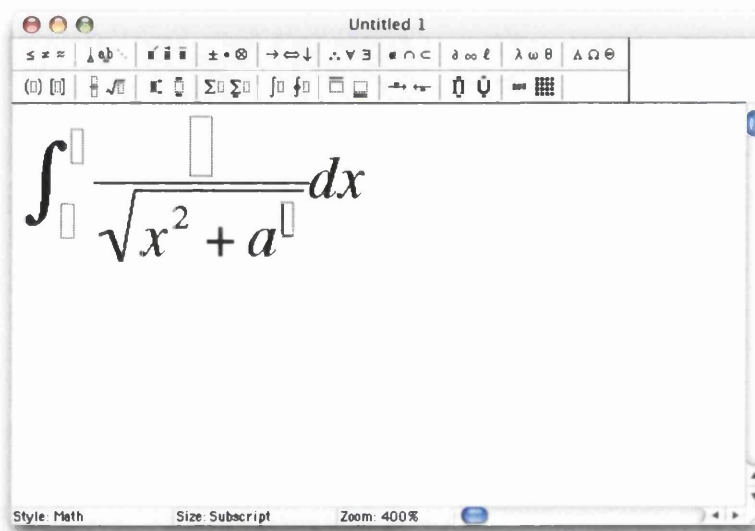


Figure 2.10: Screenshot of Microsoft's Equation Editor

Template-based editors allow mathematical expressions to be built up, combining basic building blocks together with more complex templates for different mathematical operators.

Figure 2.10 is a screenshot of Microsoft's Equation Editor showing a partially

entered expression. At the top of the window is a toolbar that provides the templates for the editor. The remaining part of the window shows the current equation. The grey boxes in the equation are the placeholders for further templates or simple expressions. In Figure 2.10 the limits for the integral, the numerator of the fraction and the exponent of  $a$  are all placeholders. The cursor can just about be seen in the placeholder for the exponent of  $a$ , and this is where any new mathematics will be inserted by default.

Basic linear operations such as multiplication and addition can be entered from the keyboard. Two-dimensional operations like fractions, exponentiation and integration are entered using templates. Templates contain placeholders for further building blocks. An example is the fraction template which has placeholders for a numerator and denominator. A more complex template example is a summation that has placeholders for the subscript, superscript and the sum, or a matrix template which has  $n \times m$  placeholders, depending on the size of the matrix.

Templates are usually added from a palette or by menu selection and inserted at the current cursor location, which is controlled by the mouse or arrow keys. The final complete expression is built up by adding templates within templates. In Microsoft's Equation Editor the toolbar at the top of the window provides access to most of the templates.

These user interfaces rely on the cursor position for adding new mathematics. The cursor can be moved by clicking using the mouse to any valid point in the mathematical expression. The arrow keys also provide some of this functionality but can be confusing to use when the mathematical structure being navigated is complex. The mouse can also be used to select portions of the expression which enables additional actions such as copying and pasting mathematics. This is restricted to either linear textual selection or selection of entire templates.

### 2.5.3 Visual methods

Offline recognition has traditionally been used to digitise mathematical documents that have already been typeset and printed. Digitising mathematical documents has been an area of research for some time, Anderson [1968] was using syntax-directed recognition in 1968 to recognise typeset mathematics. Typeset mathematics usually have a far more structured and consistent layout than handwritten mathematical expressions. Recognition of typeset mathematics therefore tends to be a simpler task and can provide better accuracy.

Pen-based user interfaces have now become far more common, in devices like tablet-PCs and handhelds. As a result, online pen-based mathematical recognition is now a more active area of research.

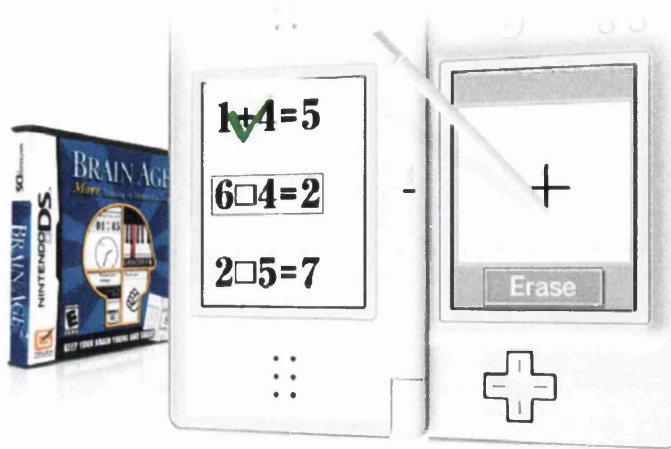


Figure 2.11: Nintendo's Brain Age

An advantage of online pen-based user interfaces is their potentially natural and intuitive interface. The majority of users are accustomed to writing mathematical expressions on paper with a pen. A pen-based interface utilises this familiarity by providing a similar user interface. Users are therefore able to use their existing experience, reducing their need to learn new user interfaces. The advantage over real paper is obvious, as a computer provides the power to compute, manipulate and solve mathematical expressions — while the pen-based user interface provides a natural method for entering mathematics.

Pen-based systems also allow a greater flexibility in how mathematics are entered. The ability to enter mathematics anywhere, and the lack of a cursor makes the user interface simpler. However, pen-based systems are rarely foolproof and users will often have to correct recognition errors. Compared to typeset mathematics there are lots of inconsistencies in how users write mathematical expressions that makes them extremely hard to recognise. A handwriting mathematical system has to deal with an arbitrary order of entry, the diverse nature of the same symbols, and a rough positioning of the various elements of the expression.

The use of visual methods in mathematical expression entry covers a broad range of capabilities from simple augmentation of linear or template entry to complete expression entry and editing using a pen.

Nintendo's Brain Age, shown in Figure 2.11, does not calculate mathematics but tests your mathematical skills by using a pen to input the answer using handwriting. The answer is always a simple symbol.

Maple provides simple character based handwriting recognition. Figure 2.12

shows Maple's character recognition palette. The typical use of this feature is to draw the symbol with a pen or mouse then click the recognise button; the suggested symbols that match the handwriting are then shown below the hand-written character for the user to select or to insert into their mathematics. As can be seen in Figure 2.12 the recognition is fairly basic and the utility of this interface, excluding writing rare symbols, is questionable.

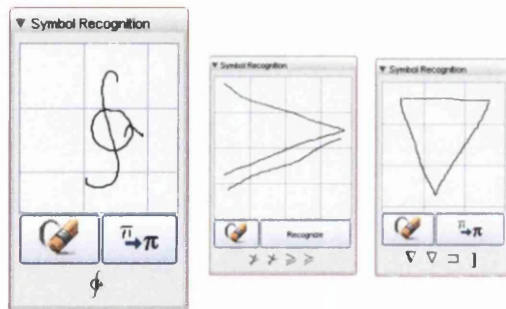


Figure 2.12: Screenshot of Maple's handwriting recognition

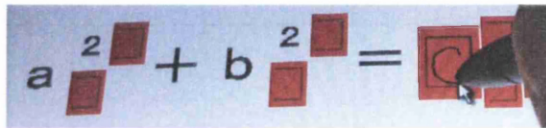


Figure 2.13: Screenshot of MathBox [Kasuya and Yamana, 2007]

MathBox [Kasuya and Yamana, 2007], shown in Figure 2.13, provides a hybrid visual template-based method. MathBox allows the user to write symbols using a pen inside defined boxes, this means MathBox does not have to perform syntactic structural analysis, and the recognition problem is simpler and therefore more accurate. MathBox trades the user's flexibility for accuracy of recognition; it also does not perform calculations or support editing.

xThink allows entire multiple expression input but provides its output as a linear string. It also allows some editing and alteration of expressions and adding notes.

FFES [Smithies et al., 1999] allows the freeform entry of complex expressions and provides some morphing of the handwriting symbol's position. FFES provides  $\text{\TeX}$  output of the handwritten mathematics.

MathPad [LaViola and Zeleznik, 2004] is not designed for mathematical manipulation or computation, but links mathematics to sketches, allowing the mathematics to animate drawings.

The Microsoft Equation Writer, shown in Figure 2.9, allows input for mathematics by handwriting or by template. It also provides an imitation of

a handheld graphing calculator, along with a small screen and multiple themes.

## 2.6 Summary

This chapter has outlined the context of previous research and applications of mathematical entry and computation. The three main user interfaces for mathematical entry to computers: linear, template and visual cover a wide spectrum of user interfaces, and all provide different benefits and disadvantages.

Visual methods of expression entry using handwriting would be the most natural for the user, however the recognition problems mean that these interfaces can be slow or error prone. Interacting with mathematics using a pen also provides many new and interesting possibilities for interaction, such as MathPad's linking with sketches and easy input of mathematics on small devices.

## Chapter 3

# Design & development

This chapter describes the principles and development of a handwriting pen-based calculator, and how they progressed and built on each other. Neither the principles, described more fully in Chapter 4, nor the implementation came first, they both developed concurrently. What started as a principle, when implemented, was altered, refined and improved. What started as pure implementation or design, later was extracted and distilled to be described in a principle.

A full description of this calculator is provided in Chapter 5. This chapter provides a context for the design decisions and how the user interface was created. Figure 3.1 shows a brief snapshot of the calculator's final user interface being used.

The calculator has provided an enjoyable user interface for a calculator and an easy interface for enjoying and exploring mathematics. For example, it was invited to be exhibited at the Royal Society Summer Science Exhibition where it was used by users both to do and explore mathematics. Thousands of people tried it: some people played with it, some did advanced mathematics on it.

Success comes in many forms. The calculator is clearly very attractive for first time use. This *might be* because it is novel and innovative, rather than really better. Its benefits for, say, long term use in a classroom is unknown — what happens when children get bored with its novelty? But the success of the reaction and enjoyment users get out of the interface is unmistakable. From all walks of life users have laughed, smiled and grinned whilst doing mathematics, even if they did not realise it.

Calculators are often not fun or enjoyable and the extent to which users enjoyed using the calculator was a surprise. The obvious question that follows is: Why? Why is the calculator a good interface? Are there principles and ideas embodied in the calculator that can be utilised in other applications, or instead do these ideas only work as part of the greater whole of the

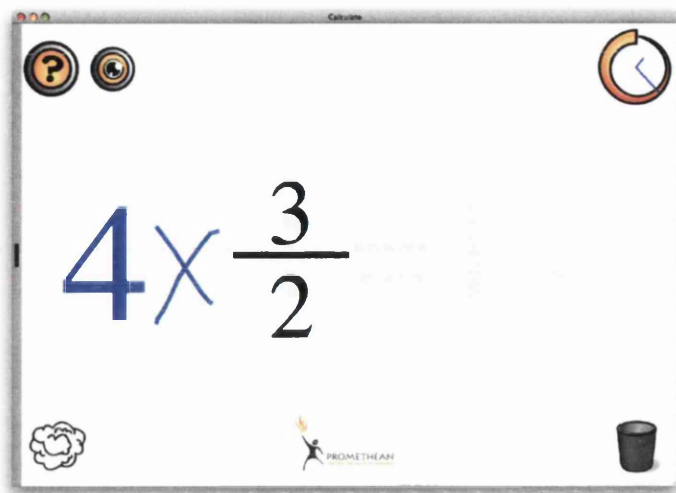


Figure 3.1: Writing  $4 \times$  in front of  $\frac{3}{2}$  using the calculator

calculator?

The calculator was designed and developed in an *ad hoc*, intuitive way and at the same time parallel to its development, principled ideas about the user interface interaction were created, refined, and incorporated. Lots of ideas and concepts were built into the calculator as part of its design to create a good interface. Gradually the ideas were reduced to principles, which then informed further development. The concepts and principles that were initially intuitively built into the calculator, are now identified in the final product. The purpose of this chapter is to rationally reconstruct that process of principle-led development.

### 3.1 The development process

The initial prototypes and pilot studies of the calculator's design used an off-line (not in real-time) recogniser, used to recognise type-set mathematics which was scanned and digitised as images. The algorithms developed further into on-line handwritten mathematical recognition, using a Wacom graphic artist's pad. Finally, the idea of computation for the mathematics, in addition to the recognition, was built on top of the recognition system.

#### 3.1.1 Like paper

Iterative design is widely recognised as being essential in interactive systems design [Gould and Lewis, 1985] and the same process was employed with the calculator. Thus, as the initial stages of the implementation of this calculator

took shape, the broad principle that guided the developing design became “it should work like paper, but with answers”. Here, the iterative design identified a *principle* which emerged through reflection on how to generalise and develop an off-line recogniser.

This principle is intended to convey the idea that if you swapped a user’s pad of paper and pen for the computer or graphics pad, then the user should be able to write mathematics in the same way (regardless of order or style) without trouble. The motivation for the principle was the hope that by providing an interface that works in a consistent and closely related way to that which users are already familiar with, then users will find the new user interface easy and intuitive to use. An interface like this also exploits the fact that a pen and computer input pad afford the same interaction as pen and paper for mathematical entry: they suggest and encourage the same style and flexibility of use.

That “the calculator should work like paper, but with answers” is an ambitious goal and it is still something that the design and implementation of the calculator aspires to, at least when strictly interpreted.

The first place where this principle impacted the actual implementation was the character recogniser. A recognition system like Graffiti [Fleetwood et al., 2002], where each symbol is written using a single pen stroke, has the potential for more accurate symbol recognition but requires users to write differently to the way they do normally. In order to work like paper, which the principle proposes, the calculator should accept all normal handwriting. In Graffiti, symbols are written using a single stroke without lifting the pen from the ‘paper’, which involves a special alphabet the user has to learn, but makes the recognition problem much easier. For example, the letter F is written in Graffiti as Γ.

The decision to behave like paper affects how the mathematical recognition system should be designed and what gestures are appropriate to be used. To work like paper the mathematical recognition needed to work without any restrictions on the order of input, like FFES [Smithies et al., 1999], or on the location of input, like MathPad [LaViola and Zeleznik, 2004].

The calculator was designed so that adding additional symbols is done by writing on top of the existing expression in the same way one would add symbols to paper. Paper also supports corrections, through the use of an eraser, and it was similarly important for the calculator to allow corrections so that users could fix or edit mathematics. To support the initial design, there were some deviations from the principle’s ideal; one gesture was provided, a single stroke X shape, which deleted the symbols that it was written on-top of. (This gesture can be used on paper of course, where it has a similar meaning.) Gestures are relatively underused by users [Long et al., 1998], and because this gesture is not an obvious part of writing normal mathematics, it was displayed in the bottom-left hand corner of the screen at all times to remind the user how to delete.



### 3.1.2 Affordance

The perceived affordance, as defined by Norman [1988], of an object suggests how that object can be used. A useful affordance is one that suggests a valid way in which an object can be used. A pull handle on a door that needs to be pushed to open it is an example of a false affordance, one that suggests the impossible and hinders the user.

Affordance helps refine the guiding principle. The calculator should draw on the similarities with the traditional methods of writing mathematics using pen and paper and two dimensional notation. The calculator is thus designed such that these similarities suggest to the user that they can use the calculator like a piece of paper, and that it is designed to allow this interaction.

### 3.1.3 Feedback

The early prototypes of the calculator simply annotated the recognised handwritten symbol with a box around the user's pen strokes, and provided an overlaid typeset symbol in the corner. This proved in use not to provide enough feedback for the user to easily notice when a recognition error had occurred. Annotation as a form of symbol recognition feedback was therefore superseded by clearer feedback: replacing the user's strokes by a typeset character scaled and positioned to have the same bounding box as the handwritten character.

Replacing the user's handwritten symbols, while initially jarring, provides very clear feedback of what recognition has actually happened.

The need for feedback that is unambiguous and clear is evident in how conventional calculators are used: users trust calculators and often only make a cursory check on the what the actual calculation is. It became very obvious in the early development that good visual feedback was critical to a user's comprehension and use of the calculator. This became the second principle — that “the calculator's state (effectively what it thinks it is doing) is always obvious to the user”.

The second principle implies that the actual mathematical expression, not just the symbols, that the calculator is computing is readily apparent to the user. To provide this, the expression needs to be shown in a way that is obvious to the user. This principle rules out any feedback that is not directly or strongly associated with the user's input.

Other methods of symbol feedback often only provide partial information leaving the user unsure about the exact state or computation. Two examples are: symbol colourisation, which is suggested as feedback for baseline information of expression structure [Zelevnik et al., 2007a], and style preserving morphs that morph the user's actual handwriting to provide feedback on

the expression [Zanibbi et al., 2001]. Style-preserving morphs are claimed to provide a better form of feedback than typeset symbols, because users prefer rough-looking sketches; and typeset input connotes an undesired authority and immutability. Neither of these approaches provide clear feedback.

At every point during the recognition process the user is informed by the visual and sound feedback of what the calculator is doing.

The principle of continual feedback was later refined to the *projection* principle, which is described in further detail in Chapter 4. Subsequently in the design process this principle came to dictate how the dock thumbnails should appear and how the undo system would work as well.

There is a tension with recognition times, between recognising symbols quickly and providing the user with sufficiently quick feedback, and with recognising slowly but providing the user with lots of time to write multiple stroke symbols and time to finish their writing. Ultimately it depends on the user, of course: a school pupil will have a different requirement to a practised adult use. To allow for individual differences, the actual timings of recognition were made adjustable for each user.

### 3.1.4 Morphing

Another important aspect of feedback is the final recognised expression, without clear feedback a user might misunderstand what expression is being calculated. The *projection* principle encouraged the immediate replacement of the user's input with a correct and typeset mathematical expression. However this was a very jarring experience and users found it hard to make the connection between their input and the resultant expression.

Morphing between the user's input and the calculator's result provides this continuity and eliminates the harsh and sudden replacement of the user's input. Littin [1993] suggested morphing as a suitable method for retaining continuity between an entered expression and a recognised expression. He describes a method that replaces the stroke data with a vector font similar to handwriting which is then morphed to the correct place.

By using gradual changes, morphing minimises the disruption of the user's mental understanding of the state of the calculator. It allows the user to easily keep track of what is happening by providing visual continuity. Morphing also provides useful feedback on the accuracy of the ongoing interpretation. This is summarised in the *continuity* principle which is more fully explored in Chapter 4.

Once the symbols have been recognised the calculator morphs them to the correct positions, such that the end result the user sees is a neat typeset mathematical expression. This feedback is designed to both provide clear

understanding of the mathematics and continuity such that the user can follow and understand the feedback.

### 3.1.5 Editing

The “works like paper” principle implies that editing mathematics should be no harder than writing on top of the current expression. From this principle the calculator was designed such that new handwriting should be recognised and incorporated into the expression being edited.

As users used the calculator it became clear that users expect to be editing what they see. Although the mathematical recognition system made mistakes the users treated the calculator’s user interface like paper and expected their additions to the current expression to be interpreted in the context of everything that was visible on screen. The principle, *What You See is What You Edit*, which is described in the next chapter, is this principle of simply editing the ink but generalised to a broader range of user interfaces.

Implementing this led to a nice solution that provided both straight forward editing and predictability: the mathematical expression is re-recognised every time the user edits it. After each edit the current mathematical structure is thrown away and the previous typeset symbols and the new input are treated as a whole. This “ink editing” allows the user to edit in whatever way they want.

It was also observed that users would often expect individual symbols to be editable, users often attempted to correct symbols by drawing on top of them. This results in a new symbol being created, and if the user attempted to correct a symbol multiple times, as did happen, this could result in a large number of unexpected symbols. Unfortunately the symbol recognition system is not able to handle these type of edits. A benefit of not allowing symbols to be changed is that it is much easier for the user to add new symbols on top of the mathematical expression, without the worry that they might by accident end up changing the underlying symbols.

### 3.1.6 Drag and drop

Drag and drop fitted neatly into this implementation of editing, rather than rearranging the mathematical expression in complex and confusing ways, a drag and drop simply moves the selected symbols to the drop location and then re-recognises the whole expression again. This leads to straight-forward implementation and interaction for the user. Feedback using a colour change and sound was added to drag and drop to help distinguish it from normal writing in order to clearly distinguish the mode because users would occasionally get confused.

Drag and drop works well with anything selected, even if what is selected is not valid mathematics in and of itself. Chapter 5, section 5.5 provides some good examples of how powerful this flexibility is. The implementation of drag and drop moves the dragged symbols and shrinks them down to fit into the expression where they are being dropped. It is therefore impossible to drag an expression, e.g. a square root sign, over the top of an expression such that it will contain what it is dropped on-top of. Edits like this need to be performed in reverse fashion, such as dragging the expression underneath the square root sign. This restriction allows the implementation of drag and drop to be very powerful, predictable and understandable.

### 3.1.7 Erasing

To reduce the number of gestures and make the calculator simpler the initial erase gesture 'X' was replaced with a drag and drop to a waste basket, utilising the fairly universal computer metaphor. A benefit of this deletion method is that the user can be specific about what is erased, they can ensure they have the correct parts of the expression before dragging it to the waste basket.

An initial observation of how users interacted with the calculator was that they liked to start from scratch if they made an error in a simple mathematical sum, rather than fixing the error. To facilitate this an erase button was added in the lower left corner of the screen that wipes the entire expression and gives the user a clean sheet to start again from.

The entirety of mathematical editing is provided through two interactions, adding new symbols to the mathematical expression by writing on top of it and rearranging or deleting symbols by using drag and drop. The *What You See Is What You Edit* nature of both of these means that the user's interaction is simple, predictable and powerful.

### 3.1.8 Computing

When the calculator provides the solution it morphs the user's input, too much change was found to be annoying because it felt like the calculator was interfering with the what the user wanted. So the calculator was designed to "alter the user's input as little as possible".

After recognising and morphing the user's input to a neat typeset expression the calculator provides the correct answer. In order to provide an experience like that of paper the answer is provided in context, appended on the right-hand side of the user's input. This appending does not rearrange anything the user wrote, it just adds the correct answer to the side.

This provided a clear way to connect what the user wrote to the answer the calculator provided. Inserting the answer in-line with the user's input works

well because that is where users would write the answer if they performed the calculation on paper themselves. The answer provided does not exist, it is just temporarily appended to the input to make it mathematically correct and changed when the expression changes. Unfortunately this can lead to users getting confused when they attempt to drag or delete something that is not there in the same way that their input in black is. To mitigate some of this confusion as the user begins to draw the current answer is faded out to discourage the user from attempting to edit it.

### 3.1.9 Partial expressions

Users often write partial expressions, or mathematics that are incorrect. To allow for this the calculator was modified to provide a mathematical solver similar to Harold Thimbleby's [1986, 1996] text based calculator. This calculator fills in all the missing parts of any expression such that the mathematics are always correct, which provides the huge benefit that the mathematics the calculator shows are always correct. That is in the mathematical sense, any final expression shown is mathematically correct. This fits in very nicely with the *projection* principle, the answer is up to date and correct at all times.

When this is combined with the user being able to write equality signs then the calculator provides a powerful way of writing half finished expressions to get the correct answers filled in for the user. This allows solving lots of simple problems like  $\frac{?}{3} = 5$  very easily. This powerful interaction is in a sense declarative, the calculator corrects everything such that it is always correct. The way it is always correct means that even if the sum is not what the user expected the mathematics is still correct. By handling partial mathematics sensibly the calculator can be used in stages and by providing an equality sign the user can solve all sorts of interesting and useful mathematics very simply. This is described and extended as the principle of *declarative interaction* in Chapter 5.

When the calculator corrects an expression by providing computer 'answers' in-line with the user's input it involves some rearrangement of the user's input by necessity. However because it is designed to change the users input as little as possible, any computer correction is inserted as much as possible as coherent chunks and in predictable places.

Harold Thimbleby's [1996] text based calculator could calculate missing exponents and symbols from the user's input. For example  $2^?=100$  has a missing exponent to the right of the caret symbol. However, in a handwriting, two dimensional calculator there is no obvious way to signify a missing exponent because there is no symbol like  $\sim$  to indicate a missing operand. Originally, therefore, the calculator provided a question mark symbol, which signified a missing number: thus a user could write  $2^? = 100$  to solve the same equation. However, the question mark symbol is not correct mathe-

matics and was removed on principle. Without the explicit symbol, the same effect can be achieved by using a left open bracket,  $2^{\text{(`}} = 100$ , or by placing a decimal point (as in  $2^{\cdot} = 100$ ) and so on. While this ability is initially fairly obscure, it becomes second nature: it is an idiom for ‘place-holding’ an unknown number. In the case of exponents, a user can use the technique to solve an exponential equation without resorting to logarithms.

Compare: “tap a dot where you want the answer if you can’t see it already” (which is a general instruction that works for any calculation) as opposed to “to solve  $a^x = b$  rewrite it as  $x = \log b / \log a$ ” — which involves rearrangement and only works for this specific class of problem.

### 3.1.10 Storage

As the calculator was developed as a real user interface and could be used for mathematics, the core of mathematical manipulation and calculation required more user interface support. The two main features that were introduced for this supporting role were the dock, which is used for storing and recalling multiple expressions, and the clock, which is used to undo mistakes or, as it turned out, to review earlier calculations, for which it is also well-suited.

A storage mechanism was needed to allow multiple expressions at the same time, and to let users save mathematics or numbers, to provide functionality much like the memory function on conventional handheld calculators.

The storage user interface started life as simulating the “affordance” of Post-it notes, so users could drag mathematics onto a Post-it note, which could then be moved and stuck anywhere on the screen. However, even after only a few equations were stored in the notes, the screen became very cluttered and made the calculator hard to use. This interface feature was therefore replaced with the idea of a more organised dock. Compared to Post-its, which cluttered the interface both visually and interactively, a dock keeps the storage interface consistently in one place, and physically separate from the mathematics being edited.

The dock, with any number of items in it, could also be hidden and revealed in a single consistent gesture, whereas managing lots of Post-its would have required the user to do lots of gestures to organise them.

### 3.1.11 Undo

Undo is one of the major benefits of using a computer. It is easy, if not eventually inevitable, for a user to accidentally make an edit or delete or add something they did not want to do.

Most undo and redo systems are discrete, providing steps backwards (or forwards) in time. However when conventional undo was piloted in the calculator it broke the continuity of flow that morphing and feedback achieve so well. In order to provide a smooth interaction that fit the calculator's interface a slider was introduced that scrubbed (i.e., animated) through the history like a movie player. As the slider was moved, it animated the creation of the current expression.

This approach animated the mathematics smoothly, just like the user was winding back time: the typeset mathematics morphs backwards as time is rewound, eventually reaching to their original places and eventually disappearing. The morphing provided the user with a clear continuity through all their edits and writing. The *projection* principle meant that the slider had to be directly linked to the mathematics, so at no time can it be inconsistent with the expression.

However, although a slider is a good interface for this sort of interaction it does not scale well. When a user has been using the calculator for a while, the distance the slider has to be moved to rewind to some exact place becomes too small a target for the user to be able to hit easily. One solution to this is to use a non linear slider, but this has its own problems. The solution used was to implement the slider as a 'clock', and this provides lots of additional benefits. Firstly, the appearance of the clock-style control associates it with time and thus naturally with 'going back into the past,' and hence undo. The circular motion of winding a clock scales well and can easily be repeated as much as the user wants. The motion also works very well for a pen based interface, where drawing circles is very easy for the user. The user can also see the time move onwards each time they make an edit, and this is very natural behaviour for a clock.

The time shown is linked to the current state, always consistent, so to get back to 'now' the user only has to rewind till the clock shows the current 'time'.

Like many interactive ideas, it is much easier to see it than to understand it from a static written description!

### 3.1.12 Teaching applications

As an aid for using the calculator for teaching, it was requested by users that the calculator had some way of posing mathematical questions. The "hide answers" feature was a direct result of this user feedback: this allows the answers provided by the computer to be toggled on and off by clicking an on-screen button.

When the answers are hidden, the computed output, which is usually shown in red typeset mathematics, is replaced by a red empty box. A teacher can use this feature, say, using a whiteboard in a classroom, to ask questions like

‘what is two times four?’ or ‘what do I divide 20 by to get five?’ When the answer is needed, the teacher can click the on-screen button and the answers are toggled on so that all the students can see them. The new feature also allows for a student to write in their own answer, and for the calculation to be readjusted around the new input.

A large green tick is shown when a user fills in all the blanks such that there is no need for any mathematical correction. In fact, the tick makes sense even when there are no questions, and for consistency it is always provided. When a user provides an answer in a box, the box disappears (for it no longer can indicate a missing answer); the screen shows a calculation with *no* box and a tick if the answer is correct. In the normal mode, the result is exactly the same.

## 3.2 Summary

Many of the features in the final user interface originated from different sources. Some features are the result of principled design, others derive from the implementation and still others from user observation or suggestions. However despite these disparate origins the process of refinement that occurred during the design and implementation was the same.

Many of the concepts and ideas that emerged during the development of the calculator were codified into more solid principles. These principles which guided the calculator’s design are described in the next chapter.





## Chapter 4

# Principles

Mihály Csíkszentmihályi [1990] describes *flow* as the state in which people are most happy, a state in which people are fully absorbed and engaged in the task at hand. This characterises the same concept as the colloquial term of being *in the zone* — a state of extreme and natural productivity.

Csíkszentmihályi has identified several factors that can accompany flow:

1. Clear attainable goals with clear rules.
2. A high degree of concentration on a limited field of attention.
3. A loss of self-consciousness, the merging of action and awareness.
4. A distorted sense of time.
5. Direct and immediate feedback so that behaviour can be adjusted as needed.
6. Balance between ability level and challenge.
7. A sense of personal control over the situation.
8. An intrinsically rewarding action.
9. Focus of awareness narrowed down to the activity itself.

A system that is designed to allow flow should incorporate some of these as design goals. The main principles behind the calculator's design are in some way practical concepts that are meant to direct design, in order to inspire user interaction flow.

### 4.1 Principles

This section provides a methodical description of the more important and general principles that shaped the development and design of the calculator. It seeks to identify and explain the principles that developed alongside the

calculator and which informed and directed the design of the novel user interface.

As described in the preceding chapter, the primary principles of the design developed to be:

1. *Projection* — Changes to the system’s state are immediately visible everywhere.
2. *Continuity* — Continuous feedback and morphing provide the continuity between state changes.
3. *What You See Is What You Edit* — Only what is visible in the user interface determines how the system can be edited.
4. *Declarative interaction* — There is no distinction between input and output.

Here, the terminology has changed and been refined from the previous chapter; we will describe the principles in more detail in what follows.

Each of these principles relates to an aspect of a user’s interaction with a computer system. The idea is best communicated in Figure 4.1, which shows the circular interaction cycle of a computer system and a user interacting with a user interface. Each of the four principles is focused on a different component of the interaction cycle, and together they combine to describe a system that as a whole engenders *flow*.

Each of the four following sections discuss these main principles in turn. Finally this chapter finishes by describing other principles that have been a useful part of the calculator’s design.

## 4.2 Projection

The first principle is *projection*, which could be described as “consistent and immediate changes everywhere”.

In the real physical world an object exists in only one place. A cup on my desk exists nowhere else. This means that talking about and interacting with that cup is simple. There is no ambiguity when talking about it and if the cup is moved then it is in a different place.

Simple! The real world works like this, and as residents of the real world it is how our brains are wired to work, and perhaps how we expect computers to work. Interacting and referring to objects in the real world is easy and natural because these simple rules are followed.

Unfortunately computers do not work like this! Computers have none of the physical constraints on what they can represent that the physical world has. How we use and interact with computers is mostly limited by the designer’s

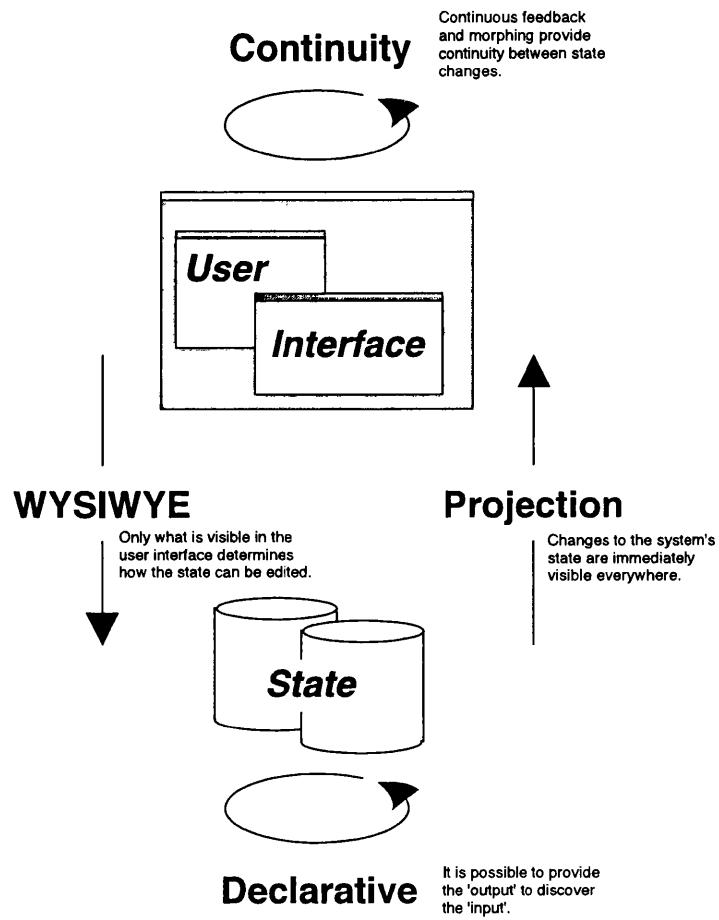


Figure 4.1: The interaction flow cycle

imagination and abilities. There is no need for computers to interact in a similar way to the real world. Thus it is now possible on a computer to show a multitude of different representations of the same thing at the same time. On a computer the back and the front of the cup could be visible at the same time or there could be a detailed view and an overview of the same cup in different windows.

By creating multiple versions of the same object we are creating a potential confusion about how to interact and refer to the object. The complexity caused by the existence of various different versions of the same object is a problem that occurs in many different situations. It is a well recognised problem in different fields, and there are many guidelines for avoiding it. For example, in almost all forms of record keeping the duplication of records should be avoided. In database design the well known guideline for “minimising the duplication of information” is a part of normalisation. In programming the same guideline takes the form “minimising the duplication of code”. In mathematics there are very clear rules and in  $x + x = 2$ , the  $x$ s refer to the same number. Other solutions for avoiding this complexity also appear in different forms in many other domains.

The guideline against multiple versions or duplication is common and widespread in many different domains because duplication can cause lots of problems. Any duplication of information that is not kept up-to-date causes information consistency to break down. If one instance of a duplication becomes different to another, a database will quickly be corrupted and a computer program will start to perform wrongly. And because the information is partially correct, this corruption is often very hard to spot. The problem with duplicate information is not that the duplication exists but that it needs kept up-to-date and is instead more often than not forgotten and left to become inconsistent.

Frequently a programmer will replicate a bit of code because it is faster than creating a more general abstraction. Then later a fix is added to the original code but the programmer forgets to correct the duplicates. The program almost works, except when the duplicated code is used. Thus the programmer spends many many fruitless hours trying to track down why the program occasionally does not work. From personal experience, duplication of code is immediately easier, but often very painful later.

### 4.2.1 User interfaces

Unfortunately the duplication of information happens all the time in computer user interfaces, and the duplicated information is often not kept up-to-date.

User interfaces provide the user with the output data from the computer. This data is often duplicated and it happens in lots of little places where

it is not always obvious, particularly in the main parts of user interfaces where we take it for granted. All this duplication of information in the user interface increases the amount users have to remember. Many computer systems fail to remember the duplicated information in the user interface so users have to remember themselves, and if they do not, the duplication is forgotten and things go wrong.

### 4.2.2 Multiple views

Yet despite the obvious problems duplication creates, it also provides a lot of flexibility and power. A desirable part of this is the ability to have multiple views of the same data.

Multiple views of the same data can show different and useful aspects. For example, a graph and a best-fit line can be thought of as different views of the same data and each provide a different insight. Each view displays a different transformation of the underlying data. A graph in a spreadsheet complements the raw data. The two different views of the data together provide a greater understanding and ability to utilise the data than each does individually.

In a similar way both our eyesight and our inner ear canals are very useful for balance and both can be thought of as providing different representations of the same orientation data to the brain. Their combined information allows us to be much more agile than we could with only one source of orientation. When the two sources are combined and their duplicate orientation data match they provide a greater ability to balance. Yet when the orientation data is inconsistent we get dizzy, can't stand up and can be sick, in a response which is thought to be a reaction to what might be hallucinations due to poison [Triesman, 1977]. The multiple views of the data, from eye and ear, are very useful but only as long as they are consistent (although potentially the discord could also be useful, in this case for detecting poisoning).

There are ways of providing different views of a single object without using duplication, for example in the physical world it is possible to use mirrors to show different views of a single object. These are multiple views of the same object yet the object has not been duplicated. When the object is manipulated then it changes instantly (for all normal purposes) in all the mirrors at the same time. There is no possible way for the mirrors to ever, even for a split second, be inconsistent.

This is not duplication: it is providing multiple, consistent and different views of the same object. This concept is the principle of *projection*.

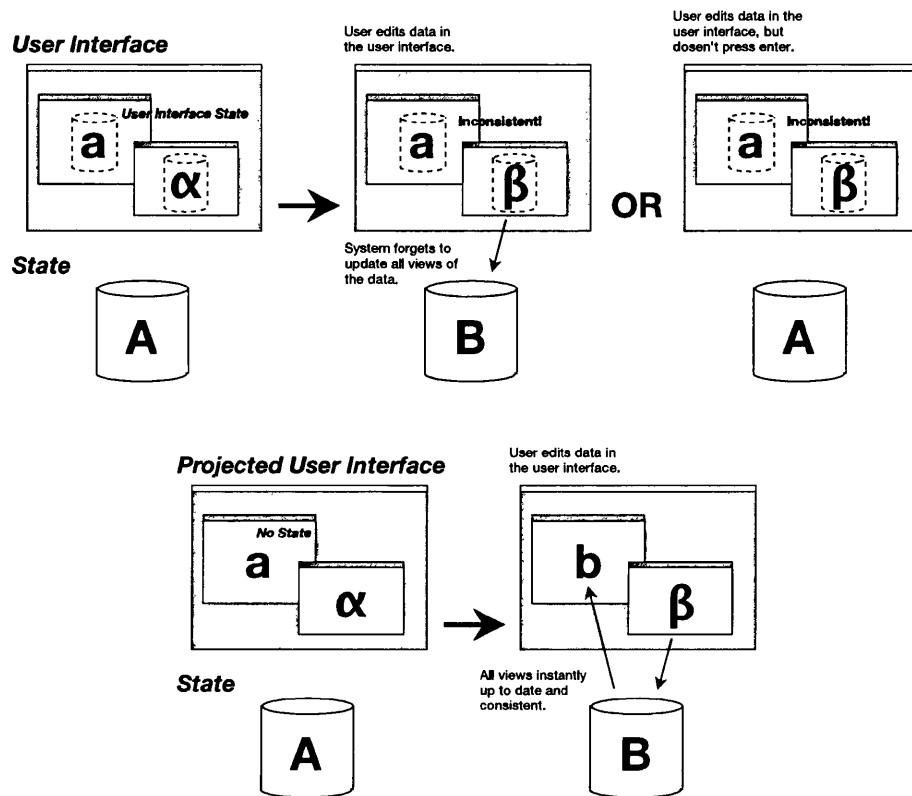


Figure 4.2: Normal and projected user interfaces

### 4.2.3 Projected user interfaces

This idea of *projection* is similar that of projecting a film in a cinema. The picture is a projection of the film strip in the projector, if the film frame changes so does the image, if projector is switched off then the cinema screen goes black, without the projector the cinema screen does not contain a picture.

A projected view in a user interface is one that is projected outwards from the underlying data. That is, separate from the data the view has no form. As the data in the system changes, projected views in the user interface are automatically updated instantly. This ensures that the user interface is always consistent. Figure 4.2 shows how a normal user interface can often become inconsistent and how a *projected* system is always consistent. The *projected* user interface contains no state and thus cannot cause a consistency error in the same way.

#### 4.2.4 Visibility of system status

The system should always keep the user informed about what is going on, providing appropriate and timely feedback. Changes in the behaviour of the user interface should be reflected in the appearance of the program. The visibility of the calculator state is encompassed by the projected user interface, which ensures that any views of the system state are always up to date.

#### 4.2.5 Similar concepts

The term *projection* is inspired by the phrase *projecting editors* [Simonyi et al., 2006] which is used to describe a user interface separation of concerns as part of an integrated development environment. Simonyi describes his Domain Workbench IDE that is designed to provide multiple different views of a program in order to allow multiple users with different needs to access and edit the program data in different ways.

Shneiderman [1983] outlined some basic guidelines in terms of user interfaces, called direct manipulation, a phrase he introduced. Shneiderman's definition of direct manipulation included the "Immediate and continuous display of results" which contains aspects of the immediacy of direct manipulation. However this is generally confined to one-way physical interaction, usually using the mouse to manipulate a control in two-dimensional space. Research into the effect of dynamic updates [Ahlberg et al., 1992], found users to be significantly faster, less error prone and tending to enjoy the dynamic interface more.

Projection is also similar to a key concept of mathematics, *referential transparency* which means that a name or expression (a view) and its value (the data) are interchangeable. For example, in the expression  $x + x = 2$ , the name  $x$  and the number 1 can be used interchangeably for each other, without altering the value of the expression.

#### 4.2.6 Editing

The tricky part of implementing *projection* is handling editing. A projected view has no state of its own, thus edits are performed through the view directly on the projected data. This happens immediately and keeps consistency with the data and other views.

For some views of data such as sliders providing immediate editing is easy, when the slider is moved the corresponding data is updated live. The slider value is always valid. Other views like text-boxes are more difficult because they allow partial and invalid data to be entered. This obviously cannot be transferred to underlying data-model or translated sensibly to display in



other views. A consistent and understandable scheme is needed to handle invalid input.

The best solution to invalid input is to interpret it intelligently and provide the user feedback about the interpretation. For example, a common interpretation of non-numerical input in a text-field would be 0 and the users textual input could be replaced with the numerical interpretation.

Another solution to handling invalid input data is to clearly highlight the user's edits as not having been committed to the underlying data, the highlighting shows the user that the view is currently inconsistent with the other views and that it is currently breaking the projection. A lot of web-based forms do this, although only highlighting the problem after the form is submitted.

Simonyi's [2006] solution is to make the underlying data model "comfortable" with erroneous states, although in practice this seems to be the most complicated solution.

Finally a lot of non-projected user interfaces do nothing. The edit might not be made or the user might not be able to remove the focus from the current control until they have figured out what the problem is. Both are cumbersome and frustrating user experiences.

#### 4.2.7 Key concepts

*Projection* describes how the relationship between views in the user interface and the underlying data should work. There are several aspects to this, which can be summarised in these key concepts:

- Immediate — Update all views immediately and continuously as the underlying data changes.
- Consistent — Ensure multiple views of the same projected data are always consistent. Multiple views showing different aspects of the same projected data are very useful, but only when consistent.
- Editable — Immediately perform user edits on the underlying data and *project* any changes back to all other views. Feedback in response to user input should be immediate.
- Lenient — Handle user input errors leniently providing immediate feedback or highlight the edit as not having been made.

#### 4.2.8 Example: Internet search

Our meaning of the *projection* principle can be illustrated with reference to internet search, as follows.

An example screen-shot taken from Firefox performing a search using Google is shown in Figure 4.3. The same sort of user interface exists in other web browsers such as Internet Explorer and Safari and with other internet search engines like MSN and Yahoo, so this is a common user interface example.

The three different views of the same data, the search terms, are highlighted circled in red. You can edit the search terms in the toolbar (1), in the web-page above the search results (2), and you can view the search terms that relate to the results just above the search results (3). The search results are also another view (albeit a more complex view) of the search term data, and these are always consistent with the search terms displayed at (3). In a *projecting* user interface all three versions of the search terms should be consistent at all times, but in this case all three different views are different. This inconsistency can needlessly confuse the user. Users generally do not notice this inconsistency because they have been trained, by bad user interfaces, to remember the duplication. User interfaces, such as Figure 4.3, place the unnecessary burden of remembering the duplication on the user.



Figure 4.3: Inconsistent views when searching

A scenario in which this might occur is this: I'm using my web browser to browse the internet, I want to look for some fruit so I type in 'apples' into the tool bar (1) and press return. The web browser goes to the search engine and shows the web page showing the search results for 'apples'. I change my mind and want some pears, so I type in 'pears' into the main large search field (2) in the web page and press return. The web page now shows the search results for 'pears'. I change my mind again and type in 'oranges' into the main search field. At this point the user interface will look something like Figure 4.3. As the user I have only used the interface in a simple fashion, yet the user interface now has three consistency errors<sup>1</sup>. If I hit return, what should happen?

A *projected* search user interface would keep all four views (the three views of the search terms and the search results) of the same data up-to-date at

<sup>1</sup>There is also a second editable text field at the bottom of the web-page. So in fact there could be four different and inconsistent views of the same data!

all times. The search terms between the web-page and the toolbar would be consistent, so that it would seem like a user was typing in both views the same time. The immediacy of projection also implies search-as-you-type, because any editing of the search terms is immediately reflected in the search results. Only recently computers have been fast enough to implement search-as-you-type and the lag and delay of the internet would still make this hard. An alternative solution, as previously suggested, would be to highlight the search results as out date as soon as the user types, this way it is clearer that the results are no longer projecting the current search terms.

#### 4.2.9 Example: Calculators

The calculator is a projected user interface. It might seem like the calculator only has one view of the mathematics, the “paper” that the mathematics is written on, but the calculator combines several different aspects of the same data into this view.

- The handwriting recognition is immediately reflected in the canvas by the replacement of the user’s handwriting with typeset characters.
- The equation recognition is immediately shown by the morphing of the characters to their correct locations.
- The result or answer of the calculation is immediately shown as corrections to the mathematics.

At no point are any of these three different aspects of the calculation inconsistent. All three are *projected*, immediately visible and always reflect the same underlying data. That the calculator is never inconsistent is a very important part of its usability. If the calculator was ever inconsistent then the user could be confused or misled (which is worse).

Another example of *projection* is the calculator’s undo clock which is linked to the expression displayed. The “time” the clock shows and the mathematics are projections of the same underlying data. As the user winds the clock forwards or backwards the mathematics similarly morphs, always staying consistent with the clock. Conversely, after the user has made an edit the symbols morph and the clock ticks forward, remaining consistent. For the user the interaction is fun, and it is easy to rewind to a certain point or scrub back and forth to view the creation of the mathematical expression. Without the immediate response of a *projecting* user interface the clock would be very awkward to use.

### 4.3 Continuity

Physical objects in everyday experience move in predictable and defined ways. As an object moves from one position to another it does so by passing

through all intermediary positions. A person can visually track an object as it moves and interact with it easily.

This movement and transition between states is essential to interacting with the real world, without it the world would be a jarring set of unpredictable sudden changes. Without any transition or *continuity*, understanding what is happening and predicting what will happen is very hard. A lack of *continuity* can be like trying to catch a ball with your eyes closed, you only know where the ball is once it has hit you. With your eyes open the *continuity* the visual arc of a ball creates in flight means that it is simple to predict the ball's destination and to catch it.

In contrast to the real world, user interfaces which are free from physical constraints often provide instant state change. Visual changes in user interfaces are often sudden and unexpected, this provides no way for the user to visually track the changes and create a connection between the old state and the new state. A user interface without *continuity* provides no opportunity for the user to follow the state changes and relate their actions to the visible changes. This is an experience not too different from catching a ball with your eyes closed.

For example, opening a file in a file browser often instantly draws a new window over the top of the existing windows. There is no connection between the two states, in one moment without any indication of why so much of the screen has changed. Examples like this are exacerbated by the large and multiple display setups users have, if the visual change happens far away from the place where the user initiated the action it is even harder for the user to connect the two.

#### 4.3.1 Animation

Animation in contrast is extremely successful in engaging its audience and providing a connection between the cause and effect of an action. Animation or *continuity* provides the connection between cause and effect, reducing the user's cognitive effort and replacing it with a simple perceptual task. Figure 4.4 shows the contrast between a user interface that provides this continuity and one that does not.

Modern operating systems like Mac OS X 10.6 use animation to attempt to alleviate this sudden disjointed visual change. For example Mac OS X animates a folder (or directory) opening by morphing a smooth expansion from the icon where the user initiated the opening action to the new window location, and animates the opening of a file by expanding the icon in place (this is a substitute for animating to the location because that position is unknown).

Cartoons offer an exemplary use of animation for providing fun action that provides *continuity*, the principles of cartoon animation are well covered

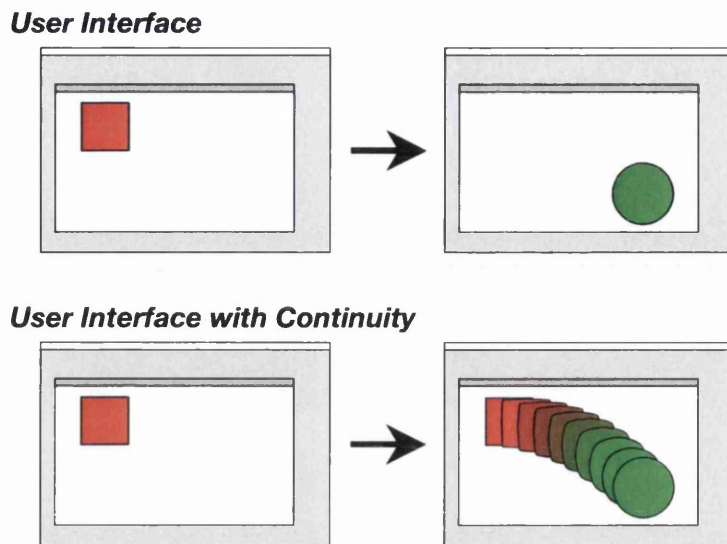


Figure 4.4: Continuity in user interfaces

in *Disney Animation: The Illusion of Life* [Thomas and Johnston, 1981], which encompasses ideas like solidity, exaggeration and reinforcement which underpin the success of cartoons. John Lasseter [1987] at Disney defines principles such as “Squash and stretch” distorting the shape of an object to define its rigidity and mass, and “Arcs” the visual path of action for natural movement. These principles of cartoon animation have been applied to the user interfaces with success [Chang and Ungar, 1993].

Continuous feedback and morphing provide *continuity* between state changes.

### 4.3.2 Key concepts

Without *continuity*, users are surprised and disconnected from a user interface, they need to work out the connections between their action and what the resulting effect was. A user interface that provides *continuity* uses animation and morphing to provide the user with the clues to follow what is happening and to join up the state changes. Changes between states should provide feedback and continuity such that a user is able to easily follow them.

- Animated — Use a smooth visual change from one state to the next to provide continuity to the user.
- Focused — Start the animation from where the user’s focus is, or from where the state change was initiated.
- Physical — Utilise some of the principles of cartoon animation to achieve a solidity and an enjoyable physical movement.

- Instructive — Inform the user about the state change using animation, what, why and how.

### 4.3.3 Example: Calculators

To provide the user with continuity when entering a calculation, the calculator morphs between states. When the calculator's state changes, the user interface catches up by morphing, this provides smooth rearrangements of mathematical equations and a seamless connection between different states. By using morphing it is easier for the user to understand the state changes of the calculator.

Continuous feedback always provides the user with a clear idea about what is happening [Shneiderman, 1992]. For example, the user's hand-written input is morphed into a typeset sum, this provides a clearer knowledge of the mathematics being calculated and how the output relates to the input.

The smooth morphing of feedback is also very visual and intuitive and is a big part of making the calculator visually fun and enjoyable.

## 4.4 What you see is what you edit

What You See Is What You Get (WYSIWYG) is a well known principle and acronym. WYSIWYG is used to describe an interface that allows the user to view the document in a similar way to what the end result would (or should) look like. Often this is in reference to what the end result is after printing, so what you see on the screen is what you get or should get from the printer. As a user interface, then,  $\text{\LaTeX}$  is not a WYSIWYG interface, because the source code the user edits looks completely different to the final typeset document. In contrast Microsoft Word is more WYSIWYG, because the document the user edits in Page Layout mode (and less so in other view modes) looks very similar to the document that is printed.

WYSIWYG is now almost taken for granted. It was important new concept when computers were usually used through command line programs, as users often had little idea what output they were going to produce until they saw it. Today most programs that produce some form of output, often via a printer, primarily utilise a WYSIWYG user interface. Non-WYSIWYG modes, like the source code mode in a HTML editor or the outline mode in Word, are still used and provide benefits for more complex tasks.

WYSIWYG is thus narrowly defined to describe the resulting output of a program. The corresponding twin of this idea would cover input into a program, the user's input in contrast to printed output, *editing* what you see. Editing what you see could be thought of as (at the cost of inventing new acronyms) What You See Is What You Edit, or WYSIWYE. Instead of

describing, as WYSIWYG does, the process of output, like printing, WYSIWYE describes the process of input, such as creating and editing data. This is not to be taken superficially, where every graphical user interface is changed or edited through a visible user interface, but the key concept is the lack of hidden constraints and surprises as the user edits. In other words, in the same way as WYSIWYG is primarily about the lack of surprise when printing, WYSIWYE is about the lack of surprise when editing.

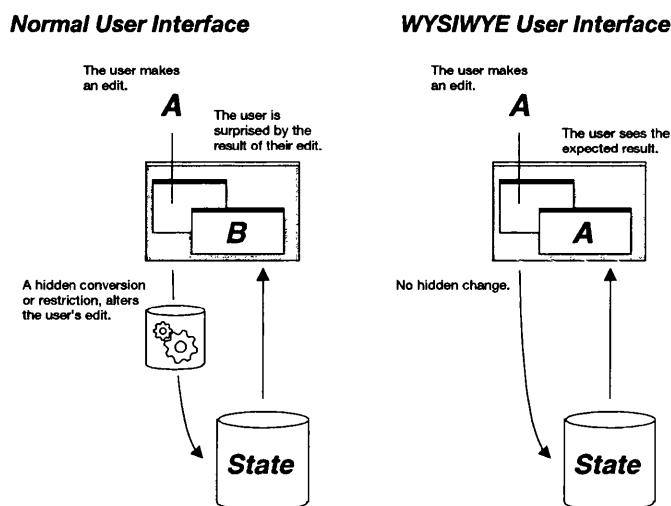


Figure 4.5: WYSIWYE in user interfaces

Figure 4.5 shows the difference between normal and WYSIWYE user interfaces. The unexpected and hidden constraints in the normal user interface causes the result of the user's action to be unexpected. In contrast the constraints on how the user edits are minimised in a WYSIWYE user interface. Constraints on the users actions usually have to be inferred from what the user can see and their knowledge of the system's behaviour. The more constraints there are the more the user needs to know about the system and the less they are just simply editing what they see, but using what they know. Any constraint or rule about interaction that the user cannot see is something they have to learn or something that trips them up.

Another acronym: Things Are Exactly As They Appear, or TAXATA [Boeve et al., 1993] encompasses some of the same ideas about the visibility of the underlying model but does not describe the user's interaction.

#### 4.4.1 Different models

User interfaces often layer a simplified editor user interface over the top of a complicated data model. This creates a similar problem to the issue of having different user mental models and system models [Norman, 1988, 1987], which is a conceptual problem when the user understands a different

model to how the system actually functions, causing unexpected results occur in response to their actions. In the case of a simplified user interface, the model the user is actually presented and interacts with is different from the underlying system model. Thus the user has a much more difficult time understanding the underlying model and predicting the correct outcomes from the actions.

Paragraph styles in Word are a simple example of when the difference in models causes confusion. Normally when text is deleted, the text after the deletion is moved upwards, filling in the space left by the deletion, but when the preceding text has a paragraph style the text after the deletion is not only moved up but also acquires the style of the preceding text. (Word does not allow easy forward error recovery from this.) Up to this point, where the unexpected happens, there has been nothing to identify the preceding text as having a paragraph style or that it will behave differently. What is visible to the user as part of the user interface is different to the underlying model, which quickly becomes a problem when it affects the result of the user's actions.

Most simple text editors in contrast are WYSIWYE editors, where the user interacts and edits with what they see, which is the characters of the text. Text editing involves a basic model that includes several simple concepts and constraints like characters, lines and selections. Once these are understood the user edits what they see and rarely needs to stop and think about these concepts because the user interface is consistent, simple and completely predictable from what they see. The constraints have been minimised to a few concepts and the user's edits are predictable and expected. No part of system model that effects the user's actions is invisible.

#### 4.4.2 Modes and hidden state

In a user interface modes are states where the user can interact with the interface in different ways depending on the state of the system. A mode changes the user's whole interaction with a program. A hidden mode makes it impossible for the user to predict the outcome of an action based on what they see breaking WYSIWYE.

It is a good user interface guideline to not use hidden modes [Raskin, 2000]. When modes are hidden, or obscured, they create a hard to use and confusing user interface. An invisible mode can mean that the results of the user's actions are completely different to what they expect. If the mode is entered accidentally the user has no idea to expect the different interaction. If the user enters the mode on purpose the user still has the burden of remembering the mode. If they forget (or another user interacts with the interface) the user has the same problem of unexpected results.



### 4.4.3 Key concepts

- Predictable and visible — Present the entire underlying model that affects the user to them. Make anything that effects the result of the user's action visible.
- Lack of constraint — Minimise the number of constraints on the user's actions.
- Consistent models — Allow the user to edit in the user interface model presented to them without the need for any understanding of the underlying system model.

### 4.4.4 Example: Syntax-directed editors

Syntax-directed editors [Teitelbaum and Reps, 1981, Reiss, 1984, Lunney and Perrott, 1988] are editors that are designed to keep the syntax of what the user is editing correct at all times. Syntax-directed editors were created with the aim of aiding and improving the process of writing program code. They were once championed by some to be “the great new way” of writing computer programs. Syntax-directed editors offered many useful properties such as reducing errors, fast refactoring and manipulation, and easier navigation. Unfortunately the editors had very restrictive user interfaces and nobody enjoyed using them [Khwaja and Urban, 1993].

Today there are virtually no syntax-directed editors in general use, although some of their benefits have been integrated into normal text editors. Examples are code completion which completes a partially typed term or structure, code folding where structures in the code can be hidden or folded out of the way, and automatic indentation. These provide some of the benefits that syntax-directed editors offered but do not restrict how the document as a whole is edited. The lack of constraints these additional features enforce mean that the code editors can primarily be WYSIWYE. This allows the user to edit the document as if it was plain text, which users have found much more appealing than the highly constrained syntax-directed editors.

### 4.4.5 Example: Template editors

Most equation editors use template-based methods that constrain how the user edits an equation. Mathematical expression template editors are good examples of non WYSIWYE user interfaces. The mathematical expression the user edits is stored in a hidden underlying syntactic tree (although the tree is implicit in the structure of the mathematics). Edits the user makes are not on the actual visible symbols but actually on this underlying tree.

This hidden structure means that the user has to understand the implied underlying syntactic structure to be able to edit the mathematical expres-

sion. Not only does the user have to understand the hidden structure in order to edit the expression but they are also restricted, by the structure, in how they are able to edit the mathematical expression. Any edits the user makes are limited by the hidden syntactic structure.

These limitations on the user's interaction make editing the mathematics in simple ways very difficult. For instance if you wanted to change an expression from  $\frac{\sqrt{a}}{b}$  to  $\sqrt{\frac{a}{b}}$  it involves at least three cut and pastes, for example:

- The square root outside the  $\sqrt{\frac{a}{b}}$  on  $\sqrt{a/b}$ .
- The fraction inside the square root  $\sqrt{a/b}$ .
- Then finally  $a$  into the numerator  $\sqrt{\frac{a}{b}}$ .

#### 4.4.6 Example: Calculators

A large part of the design aims for the calculator was to make use of the user interface's paper-like similarities. The interface should act as if the user is just drawing, deleting or moving ink on a page. The 'ink' on the screen is all the information there is, and there is no restriction to how that ink is added or moved.



Figure 4.6: An unconstrained single drag changing  $\frac{\sqrt{a}}{b}$  to  $\sqrt{\frac{a}{b}}$

The calculator enforces no constraints on the order or position of what the user writes. By editing the 'ink' the user is able to write a mathematical expression without any regard for the underlying structure of the mathematics or computer representation. The calculator interprets the mathematical input from what is visible, the same thing the user sees, using expression recognition. This permits a very powerful, and natural, interaction style called *ink editing* and plays a large factor in making the calculator easy and natural to use. Ink editing is essentially WYSIWY for a pen based interface. Figure 4.6 shows the single drag and drop using the calculator that the same transformation from  $\frac{\sqrt{a}}{b}$  to  $\sqrt{\frac{a}{b}}$  in a template editor requires at least three steps.

In the calculator the mathematical expression is always recognised from what is seen and the user only edits what they see. There are no constraints applied to how the user edits the 'ink', the mathematical constraints are

enforced after an edit is made by fluidly morphing the system into the correct state. This means that the user is free to input or edit their expression in any way they want and are rarely surprised by the interpretation of their action. The calculator's behaviour is entirely predictable if the user has the right conceptual model of its behaviour. That is the calculator is entirely predictable without any knowledge of the history of the calculator state.

The principles of ink editing, or WYSIWYE, dictate that the interpretation of input should be solely determined by what it looks like. Thus, the interpretation of the  $-$  symbol is determined by its context, and the interpretation can change if the context changes. For example, if the user writes  $-2$  it is recognised as a minus sign followed by 2. If the user then continues to write a 3 above the minus sign, the minus sign is reinterpreted in the new context and becomes a division bar, which results in  $\frac{3}{2}$ . The ink editing of this expression is natural to the user and the result looks very similar to what the user wrote. Without ink editing this would be impossible.

The calculator has a few modes of interaction, for example, selection, and dragging. Both these modes are directly linked to the users interaction and are visible on the screen. The user cannot start dragging without explicitly clicking on a selection and cannot finish dragging without removing the pen. This makes it much less likely that the user gets confused about how to interact with the calculator.

## 4.5 Declarative interaction

Often processes and user interfaces make a clear distinction between input and output. The user provides input to the computer and then the computer responds with some calculation as output, but by blurring the distinction between the two, an interface can provide interesting and powerful interactions. That is, letting the user change the output to get the input.

A programming language that works like this is sometimes called a declarative programming language. Languages like this are different to the traditional imperative languages that describe how to get to the solution. In a declarative language the programmer states the relationship between the input and output. The computer then works out the details of the individual steps to get the output. Which means that just as the computer can logically work out the steps to go in one direction it is possible for the computer to work out the reverse steps to go in the other direction.

Declarative programming is not a popular paradigm for several reasons, but partially because it involves a very different way of thinking to how most programmers are used to. Prolog is an example of a declarative language that has found use in the real world. In Prolog the programmer writes statements, declaring facts, and then Prolog backtracks to solve the equations (these are sets of Horn clauses in Prolog).

Figure 4.7 shows some examples of how the single Prolog function **append** can be used to append lists and how it can also be used in completely different ways utilising the lack of an input output distinction. Here **append** is used to test for a prefix of a list, to decompose a list into all the possible pairs of sublists, and to test for the existence of a decomposition. That is four different and useful possibilities three of which are normally not possible! These are all possible because Prolog does not distinguish between inputs and outputs to the **append** function. The **append** function can also be used to test for list membership (**append**(\_, [b|\_], [a,b,c,d,e]).) and to enumerate the members of a list (**append**(\_, [X|\_], [a,b,c,d,e]).)!

```
?- append([a,b],[c,d,e],X). % Just append
X = [a,b,c,d,e]

?- append([a,b],X,[a,b,c,d,e]). % Testing for a prefix
X = [c,d,e]

?- append(X,Y,[a,b,c,d,e]). % Breaking into pairs of sublists
X = []
Y = [a,b,c,d,e]
X = [a]
Y = [b,c,d,e]
X = [a,b]
Y = [c,d,e]
...

?- append([a,b],[c,d,e],[a,b,c,d,e]). % Testing decomposition
yes
```

Figure 4.7: Using the **append** function in Prolog

#### 4.5.1 User interfaces

The problem with making a process declarative is that many processes are inherently one way. The same problem occurs in user interfaces, which are at the basic level just ways of viewing and interacting with the input or output data from processes. A lot of what we see in user interfaces as output data would be hard to interpret as input. For example, how would altering a best-fit line change the data, or how could changing the search results affect the search terms? The usual and simple solution implemented by most user interfaces is to provide non-interactive views of this output data.

Despite the difficulty of making some user interfaces declarative, doing so can provide a lot of useful interaction potential. One of the most powerful and persuasive user interaction possibilities is exploration. By being able to

change the best-fit line, which is normally just an output only view of the underlying statistical data the user can very easily explore the relationship between the graph data and the best-fit line. By changing the best-fit line the user could explore and experiment with the relationship between the line and the data and emerge with a good grasp of the how the line and data are linked.

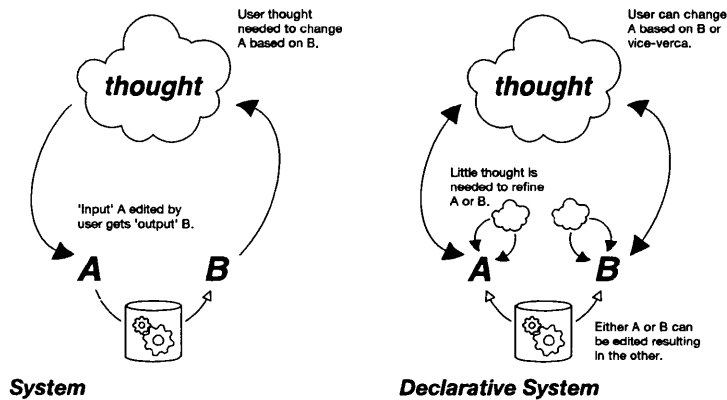


Figure 4.8: Declarativeness in user interfaces

Figure 4.8 captures the expanded interaction possibilities a declarative system offers. With a normal system the user is restricted to editing the input, seeing the result then interpreting it so that they are able to refine the input. In contrast, a declarative system allows the user many more possibilities of interaction. The user can still interact with it in the same way as a normal system but also performing the reverse, interpreting the 'input' to refine the 'output'. They can also simply refine the input or output without having to interpret anything, thus both reducing the cognitive effort and making the interaction faster and easier. Figure 4.8 reflects these many options that allow a user to work much more flexibly and freely.

#### 4.5.2 Similar concepts

A user interface that does not distinguish input from output is *declarative*, it provides aspects of query by example [McLeod, 1976], programming by example [Cypher, 1993] and even macro recording systems [Lieberman, 1993]. Using a *declarative* interface, the user, even without any concept of the underlying process, is able to show the computer what they want it to do and the computer does the rest.

Applying this sort of paradigm to user interfaces has been called 'equal opportunity' [Runciman and Thimbleby, 1986, Thimbleby, 1990] and has been explored as part of other user interfaces. Other systems [Ahlberg et al., 1992, Ahlberg and Shneiderman, 1994] also utilise the concept of 'equal opportunity' by building and extending it with their own principles.

Tight coupling [Ahlberg and Shneiderman, 1994] is an extension of the direct manipulation guidelines [Shneiderman, 1983], primarily for visual information seeking. A key concept of tight coupling is “query components are interrelated in ways that preserve display invariants and support progressive refinement. Specifically, outputs of queries can be easily used as input to produce other queries,” which touches on the declarative blurring of input and output.

Leogo [Cockburn and Bryant, 1996], a Logo IDE for children, provides three distinct programming environments; a direct manipulation of the turtle using the mouse, a visual programming environment of icons and sliders and a standard text based editor. Each of these are inter-linked and no one environment is considered ‘output’, so users can switch between the three different representations at ease. The evaluation was not conclusive but the authors did observe children often switching between the different environments for different tasks. Interestingly, a large part of the further work they wanted to undertake was to make the system more dynamic, or in other words a *projected* user interface.

### 4.5.3 Difficulties

A slow or unresponsive declarative user interface does not provide quite the same ability or freedom to explore and utilise the underlying relationships. In Prolog, the user has to enter a query, correctly written and terminated by a special character, until this point, the output (if any) is incorrect and it makes experimenting with Prolog code slow and tedious. However by combining a declarative user interface with a *projected* user interface we get an interface that is always true all of the time and allows the user the flexibility of editing almost anything. This reduces user confusion and makes interaction and exploration fast and easy, allowing a user to both have a deeper understanding of the underlying relationships and use the interface in powerful ways.

A lot of relationships are ‘many to one’ meaning the same output can be achieved with many different inputs. A simple example using mathematical equality as the relationship is  $1+3$  and  $2+2$  which both have the same result. Providing the reverse of relationships like this can be difficult, but as long as it is predictable it is useful. The most simple solution often works best, in the previous example the ‘output’ 4 would simply generate the ‘input’ 4. With a predictable two-way relationship the user can easily make use of the ability to incrementally build up the desired result by changing which ever representation (whether ‘input’ or ‘output’) is easier at the current point.

Since the distinction between input and output is blurred, it is better to think of these as multiple representations of underlying data. To allow real iterative interaction where the user is able to refine multiple representations in turn, the system needs to have a mechanism for changing the users input.

That is, given two representations, if the user specifies the value of one then changes the other, the first needs to be altered to ensure the underlying relationship remains valid.

A simple solution is to just override the other representations, replacing them with completely new results. A better solution is to adjust the values as little as possible such that the underlying relationship remains valid. The calculator does this by making the distinction between user input and computer ‘corrections’, as either side of an equality is adjusted by the user, computer additions are added or altered to ensure the equality remains mathematically correct while the user input is left unchanged. In this way the calculator provides the user with a powerful way to iteratively build up the desired result by adjusting both sides of the equality.

#### 4.5.4 Key concepts

- Exploration — Encourage exploration by allowing a deeper understanding of the underlying model.
- Interaction by example — Provide an example of the input needed when the user supplies the desired output data.
- Predictable — Always provide the same answer for the same data, whether ‘input’ or ‘output.’
- Projection — A projected declarative interface makes exploration and incremental change faster and easier.
- Incremental — Allow the user to incrementally construct an answer by cycling between editing the ‘input’ or ‘output.’
- Refining — When the user makes any edit alter the ‘output’ as little as possible to ensure it is correct. This supports gradual editing as each edit refines any others.

#### 4.5.5 Example: Calculators

Harold Thimbleby’s [1996] calculator implements a declarative user interface, which from both informal and empirical evaluations [Cairns et al., 2004] has been shown to have several advantages. A declarative calculator treats output and input equally, such that it can solve any basic mathematical expression by providing the correct input for the answer. Users are thus able to solve problems, such as ‘what power of 2 is 56?’ (i.e.,  $2^x = 56$ ) directly, that they might have no idea of how to solve otherwise, and which would be impossible without circumlocution (e.g. introducing logarithms) and impossible without rearrangement and generally losing the initial problem structure. Typically mathematical problems would be impossible to do

correctly without prior experimentation on a calculator, for different calculators, even within brands, do advanced arithmetic calculations differently!

When combined with WYSIWYE, a declarative calculator makes the experimenting trivial.

The ‘output = input’ concept works well in a calculator, because the output and input are both the same type of data, a mathematical expression or number, and can be combined together into one larger expression.

With a declarative calculator, the user can replace the computer output with their own and nothing will change. This means that if the user writes the correct answer in then the calculator shows no extra work, and it means that if the user writes a wrong sum like “ $3 + 4 = 15$ ” the calculator corrects it. When users use it they find that, as one individual put it, their old calculators are “nagging and pedestrian fusspots.” More examples of the possible ways in which the calculator can correct mathematical expressions are found in Chapter 2.

The flexibility of the declarative calculator allows an explorative and playful interaction. The calculator makes evident how the underlying mathematics operates and it enables easy exploration through play. Users are able to change any part of a mathematical expression to see how it is adjusted to ensure it is correct, and in doing so the user sees the underlying mathematics in action.

#### 4.5.6 Predictable

The corrections to user input and the morphing of the input into the final result are always predictable. The same input always yields the same output. Thus, once the user understands the calculator’s model they can predict the results of any particular edit (visually if not computationally).

#### 4.5.7 Refining

The calculator ensures the expression the user sees is always correct by adding corrections to the partial input. These corrections are inserted into the user’s own input. An incorrect expression could be corrected in an infinite number of ways, and lots of these corrections would cause the user’s input to move around or become more complex, thus making it harder to follow what the calculator was doing.

Therefore the calculator inserts the least amount of additional corrections to ensure the mathematical expression is correct. The calculator *always* corrects  $4+$  to  $4 + 0 = 4$  not any of other possibilities like  $4 + 10 = 14$  or  $4 + 4 = 4 \times 2$ .



Contiguous blocks of corrections and user input are also preferred, this means that the user's input is broken up as little as possible. Corrections are also always placed on the right-hand side of any user input. For example,  $2 = 3$  is corrected to  $2 = 3 - 1$  rather than  $2 + 1 = 3$  or  $1 + 2 = 3$ .

## 4.6 Synergy

These four principles, *projection*, *continuity*, *What You See Is What You Edit*, and *declarativeness* all work together in an unusually coherent way. Yet each principle is distinct and describes how one aspect of a user interface should operate. Figure 4.1 shows the four principles applied to the different parts of a system:

- *Projection*— state to user interface relationship
- *Continuity* — user interface interaction
- *WYSIWYE* — user interface to state relationship
- *Declarative interaction* — state interaction

The interaction *flow* framework shown in Figure 4.1 enables a user to interact with a user interface with lots of feedback, no discontinuity, clear functionality and few restrictions.

### 4.6.1 Flow

These principles match up to several of Csíkszentmihályi's factors for enabling flow. While not sufficient to "get in the zone" these principles do enable a more seamless experience when using a user interface that aids flow.

1. Clear attainable goals with clear rules.

*WYSIWYE* provides clear rules for attaining the desired goals.

2. A high degree of concentration on a limited field of attention.

*Continuity* helps ensure concentration by removing jarring discontinuities.

5. Direct and immediate feedback so that behaviour can be adjusted as needed.

*Projection* provides immediate feedback so the user can adjust their actions.

7. A sense of personal control over the situation.

*Declaration* gives the user more control to achieve their results.

### 4.6.2 Reducing the user's cognitive workload

When using a traditional calculator, before the user can do any calculation they have to translate the sum they have in mind into key presses that the calculator will understand. The new calculator removes much of this step, changing the role of the user and can reduce the user's cognitive workload. This means that the user can spend more effort on the important things and are generally less taxed by using the system.

The calculator reduces the user's workload in several very effective ways:

- *Projection* means that immediate feedback and partial input let the user build an expression up stage by stage, instead of having to enter a fully formed complete expression at the start.
- *Continuity* reduces the amount of effort to keep track of the many state changes that happen.
- *WYSIWYE* means the user can enter and edit mathematical expressions using standard two dimensional mathematic notation without constraints. This reduces the need to translate the desired mathematics into a linear format or into specific user interface actions.
- *Declaration* ensures that the user does not have to rearrange expressions to have the answer on the right-hand side.

### 4.6.3 Error recovery

Errors will always happen, so a good user interface helps users to easily recognise, diagnose, and recover from errors. Errors should be precise, indicating the problem and constructively suggesting a solution.

Errors while using the calculator, either from input or from recognition, will happen. However, the calculator's immediate feedback and visible expression from *Projection* helps users immediately recognise errors [Thimbleby, 2004] and thus correct them. The *WYSIWYE* editing abilities of drag, drop and deletion allow easy and logical forward error recovery. Undo provides simple backwards error recovery.

The calculator avoids any error messages or inconsistent state by supporting partial user input. No matter how bad the user input is the *declarative* calculator will form a valid expression and the user can always use forward error recovery from the current state or backwards error recovery using the undo clock.

#### 4.6.4 Breaking principles

We have claimed that a set of principles improves a user interface design, and we have given detailed discussion in the context of a novel, highly interactive calculator. This should have been a demanding context to test the ideas. However, methodologically it is good practice to explore the “reverse result.” If a principle  $p$  is claimed to improve an interface, absence of  $p$  should make an interface worse: just as these principles provide a good user experience, the lack of them should cause errors and problems.

We do not have space to give worked examples of the consequences of the lack of the principles, but it would be routine to do so. Here is a brief “thought experiment” of the problems that would be readily anticipated if the principles were flouted:

- *Projection* consistency errors — “this can’t be reliable.”
- *Continuity* continuity errors — “how did I get here?”
- *WYSIWYE* expectation errors — “that wasn’t what I expected.”
- *Declaration* editing restrictions — “why can’t I edit this?”

It may be thought of as trivial; that if a good design principle is broken then the interaction is also broken. But the sharper insight is that breaking the proposed principles would clearly break the user interface in *specific* ways. On the contrary, if the principles were vague (say, “make it nice”) then their opposites (say, “make it nasty”) would have an arbitrary effect.

### 4.7 Summary

The calculator’s design has been guided by four precise principles, as piloted in the previous chapter and now as refined in this chapter. The four main principles described in this chapter: *Projection*, *Continuity*, *What You See Is What You Edit* and *Declarative interaction* are all critical to the design and fun interaction that calculator provides. These principles aided the design and implementation of a calculator that has a fluid, smooth, easy, and powerful user interface, and which is enjoyable to watch and interact with.

Together the principles form an interaction *flow* framework, which is able to provide real generative ideas for making innovative and satisfying user interfaces.

It remains to substantiate the claims throughout this chapter that the user interface is effective. The next chapter reviews the user evaluation for the calculator. Then, in subsequent chapters, we will test the higher-level claims about the principles by applying them and same generative processes to a

---

very different style of user interface and showing that it, too, is remarkably effective.

## Chapter 5

# User interface overview



Figure 5.1: Royal Society exhibition. © Will Harwood

It is very instinctive and fast. It's great, I feel like Tom Cruise in *Minority Report* — Bravo!

— a PhD student at the Royal Society's Summer Science Exhibition

### 5.1 Overview

In the film *Minority Report*, John Anderton [Tom Cruise] uses a gesture-based interface to view and manipulate clips of the future so that he can stop murders. The interface is built from several large screens which he interacts with by using hand gestures. The interface displays (fictional!) precognitive visions of the future, dreams replayed through a computer, on an interface similar to that of a timeline or a video editor. Using this interface Anderton

performs panning and zooming and manipulates this ‘video’ using various hand gestures.

The interface is typical of film and CGI interfaces, it looks good, it looks fun, and it looks like it works. However, it is all faked and none of it is real. It probably took many months of computer graphics and prop design to achieve. The interface appears to be successful because of how it *seems* to interact. The fluid motion and gestural interaction is persuasive within the film. Would the interface, if actually built, be successful? Does the perceived usability work in real life, or does it just look good on the big screen?

Imagine writing a calculation down on paper, and the paper magically working out the answers. The calculator works like this, using an approach that is ideal for gesture-based user interfaces, from handhelds with pens to interactive whiteboard use in classrooms. The calculator developed resembles the interface from *Minority Report*. The quote at the start of this chapter was from a visitor to the Royal Society’s Summer Science Exhibition, where they used the calculator on a 6 foot interactive whiteboard that can be interacted with using fingers. The fluid nature of the mathematics and the smooth morphing of the symbols that the calculator uses are similar to the fluid interface of *Minority Report*, and are probably what prompted the quote. When the calculator is used on an interactive whiteboard, such as a SMARTboard, users manipulate the calculator using their hands in a natural way using gestures and handwriting, in a similar fashion to the interface of *Minority Report*. Figure 5.1 shows a user using the calculator on an interactive whiteboard at the exhibition.

The calculator is written in Java and runs under Windows, Linux and Mac OSX, and it works with standard hardware such as Mimio, SMARTboard, or Wacom tablets. It is somewhat difficult to use it with a trackpad or mouse, because creating normal handwriting movements, the calculator’s normal mode of input, with a mouse is unfamiliar and awkward.

It is fun and engaging, users enjoy using it and it also works as a calculator computing mathematics. Fun and mathematics is an unusual combination. This raises the questions: Why? Why is it fun? Why is it engaging? Are these inherent in a *Minority Report*-esque interface? Is the fun part the gestural interface? Or are there other factors that make the calculator user interface successful?

This chapter provides an overview of how the user interface works and functions. It provides an overview of the new calculator’s user interface, and how it operates from the user’s perspective.

Describing a user interface in such detail can make it seem quite dull. This level of detail is provided for clarity. In reality the calculator’s user interface is not restricted by the medium of five frame cartoon strips (such as Figure 5.3) or paragraphs of text, and thus it is much more fluid. To re-

ally grasp how the calculator works and interacts, it is probably far more productive to play with it instead of reading the comparatively dull and non-interactive text of this chapter.

## 5.2 User interface sections

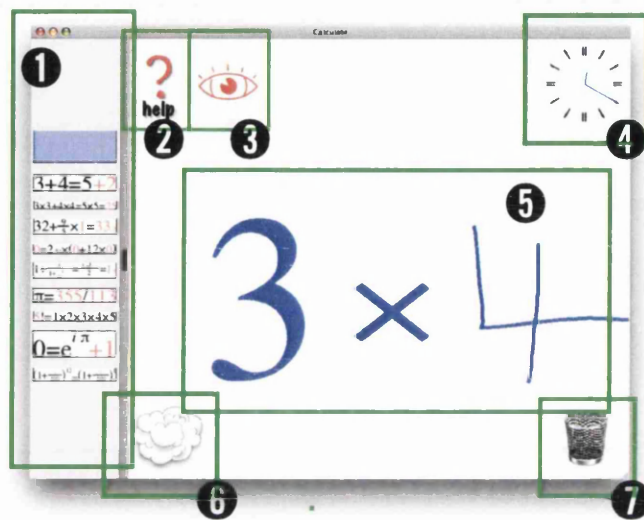


Figure 5.2: User interface overview

Figure 5.2 shows the entire visible part of the calculator's user interface, the different numbered sections of it are enumerated below:

### 1. The dock

Equations can be dragged here from the equation editor, which are recorded and saved in the dock for later use.

### 2. Help

The help button brings up a help screen that shows the user what symbols are recognised and how the calculator is used and what can be achieved with it.

### 3. Hide/show answers

The button that looks like an eye toggles the showing of answers to calculations.

### 4. History clock

The history clock records the changes made to the mathematics, and can be used to review what has happened or to undo changes.

### 5. The mathematical expression

Mathematics are written here as handwriting, which the calculator solves. Figure 5.2 shows the calculator in the process of solving  $3 \times 4$ .

### 6. Clear

Erases the whole screen for a new calculation.

### 7. Trash

Saved equations in the dock and parts of the mathematical expression can be deleted by dragging them to the trash.

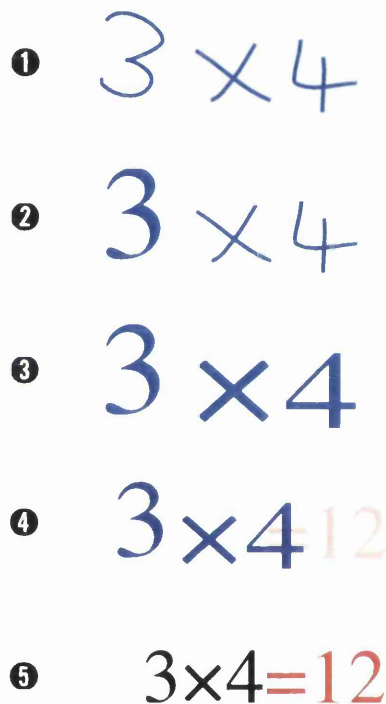


Figure 5.3: Simple calculator use

## 5.3 Simple mathematics entry

The entry of a simple mathematical expression is shown in Figures 5.3.1—5.3.5. These show a sequence of screen snapshots of the calculator in use as a user enters the simple sum  $3 \times 4$ . These steps are typical of entering a straightforward mathematical expression without any mistakes or editing into the calculator. These interactions and the principles that are the basis for them are now described in more detail.



#### 1. User input

Here the user has written by hand the mathematical expression  $3 \times 4$ . The user's handwriting appears in blue on the screen as they write, creating a rough mathematical expression. There is no special syntax or interaction, the user writes the same way of writing they would on paper. This is the first part of the original guiding design principle: that "it should work like paper, but with answers".

#### 2. Symbol recognition

After a short time delay, the calculator begins recognising the user's handwriting and replaces the rough hand-written strokes on the screen with typeset characters that are stretched to the same size and location as the handwriting.

The user wrote  $3 \times 4$ ; the calculator is "catching up" with them and has already rendered the 3 in a typographically neat font. This feedback is grounded in the *projected editing* principle, the calculator provides direct, timely and clear feedback to the user about the symbol recognition.

#### 3. Finished symbol recognition

In Figure 5.3.3 the calculator has recognised all the user's input and replaced the handwriting with the typeset symbols.

#### 4. Morphing

After all the user's hand-written symbols have been recognised the calculator identifies the mathematical expression and computes what is missing to ensure it is mathematically correct. Then the calculator begins morphing the distorted typeset symbols into their final positions as part of a neat typeset expression. The answer, or corrections, are shown in red and fade in during the morphing of the user's input. In this case the answer ' $=12$ ' is shown.

In Figure 5.3.4, the calculator is in the process of morphing all the user's input and has combined it with the output displaying a nicely typeset equation. User input that was written in blue 'dries' black, this is to help distinguish new user input as it is written with what is already on the screen.

The feedback provided by the typeset expression provides direct feedback about the equation recognition and is part of *projected editing*. The answer provides the last part of the original goal "...like paper, but with answers".

#### 5. Finished

The final calculator output is shown in Figure 5.3.5. The equation is neatly formatted, typeset and the calculated answer is shown in red.

①  $12^3=12$

②  $12^3=1728$

③  $\sqrt{12^3}=1728$

④  $\sqrt[3]{12^3}=41.56\cdots$

⑤  $\sqrt[3]{12^3}=12$

Figure 5.4: Editing a calculation

## 5.4 Editing

The next few sections take individual aspects of interacting with the calculator in turn and describe them more fully. Editing a mathematical expression is as simple as writing on top of it. The What You See Is What You Edit (WYSIWYE) principle is core to the editing experience of the calculator, it means that users need not be concerned with the hidden structure of the mathematics.

Figure 5.4.1–5.4.5 shows the user editing a simple expression, starting with  $12 = 12$ . Each step shows adding new structure to an already generated typeset expression, resulting in the final expression  $\sqrt[3]{12^3}$ .

1. Starting expression: 12

Once an expression has been recognised, it is possible to write over the top of the typeset expression to edit it. Figure 5.4.1 shows new user input in blue over the top of the typeset expression. The computer generated answers in red are faded out as the user starts to write, these answers are only temporary, and the fading out is to stop the user from making use of them.

2. New typeset expression:  $12^3$

The new expression is then morphed and typeset neatly with the correct answer.

3. Further editing

The individual steps show how an expression can be built up bit by bit while the calculator provides error checking, recognition and the answers at every step. Figure 5.4.3 shows the user adding a root on top of the original expression.

4. Additional further editing

The user can then continue to add more to the expression. The way in which a calculation is edited is flexible, the user could have written the 3 first before the square-root symbol. As a WYSIWYE user interface there is no hidden order or structure the user has to conform to, this makes it much easier for the user to edit an expression.

5. Finished:  $\sqrt[3]{12^3}$  The final result is typeset neatly incorporating all the user's additions to the original expression.

## 5.5 Drag and drop

Drag and drop allows users to easily rearrange an expression, any part of the expression can be moved anywhere else. Drag and drop is also useful



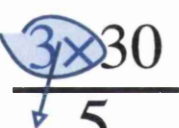
A	B
❶ $\frac{3 \times 30}{5} = 18$	❶ $2^3 = 8$
❷  $\frac{3 \times 30}{5} = 18$	❷  $2^3 = 8$
❸  $\frac{3 \times 30}{5} = 18$	❸ $3^2 = 9$
❹ $\frac{30}{3 \times 5} = 2$	

Figure 5.5: Drag and drop

for correcting recognition errors, for example when a symbol is positioned where the user did not want it to be, they can easily move it to the correct position. The ability of the user to select symbols without restriction on what or where is selected is driven by the WYSIWYE style of the user interface. Without this principle shaping the nature of the interaction, the experience would be much more restricted and complex for the user to use. Figure 5.5 provides two examples of drag and drop.

A.1. Starting expression:  $\frac{3 \times 30}{5}$

A.2. Selection

To select part of an expression the user draws round the symbols they want to select. A selection is initiated when the user has drawn a loop that contains other symbols. The selection mode is signified by a “whoop” sound and displayed as a blue transparent filled loop with the selected symbols drawn in blue, both providing valuable feedback to the user. Figure 5.5A.2 shows a fraction where  $3 \times$  is selected from the numerator.

Note that  $3 \times$  is not a mathematically complete expression but the user can still select in the WYSIWYE user interface and move it however they want.

A.3. Dragging

By clicking inside the highlighted blue selection the user can drag the selection to wherever they want. While the user is dragging an arrow indicates the drag that will happen. Figure 5.5A.3 shows the user dragging the selection down into the denominator.

By clicking outside of the highlighted selection the user could also cancel a selection and re-select a different portion of the expression or continue editing the calculation as normal.

A.4. Finished:  $\frac{30}{3 \times 5}$

When the user finishes a drag the selected portion of the expression is inserted at the end point of the drag and the equation is recalculated and morphed to a new typeset expression. In Figure 5.5A, moving the  $3 \times$  from the numerator to the denominator has altered the expression from  $\frac{3 \times 30}{5}$  to  $\frac{30}{3 \times 5}$ .

The second drag and drop example is shown in Figure 5.5B.

B.1. Starting expression:  $2^3$

B.2. Dragging

The user here is dragging a base into its own exponent. This is not a problem for the calculator, the expression is simply re-recognised with the dragged components of the expression at their new locations.

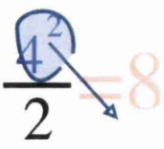
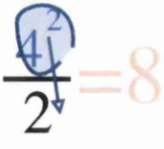
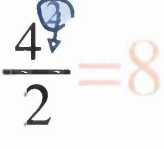

Drag	Result
	$\frac{1}{2} \times 4^2 = 8$
	$\frac{1}{24^2} = 0.002$
	$\frac{42}{2} = 21$
	$-42^2 = -1764$

Figure 5.6: More complex dragging

B.3. Finished:  $3^2 = 9$ 

The new expression is now morphed into a neat typeset expression, swapping the base and its exponent, giving  $3^2$ . The same result could have been achieved by dragging the exponent down and left below the base.

Figure 5.6 shows the result of more complex drag operations on the mathematical expression  $\frac{4^2}{2}$ . Notice how versatile the drag operation can be in rearranging the mathematical expression, there are very few limitations and a single drag can move and rearrange complete or partial expression. The calculator reinterprets the new expression after the drag, making sense of whatever the user did. To understand how the resulting expression was arrived at, thinking about what the drag and drop achieves is useful. For example in last case of Figure 5.6, which may seem to have a counter intuitive result, the drag and drop changes  $\frac{4^2}{2}$  to  $-\frac{4^2}{2}$  which is interpreted in a sensible way as  $-42^2$ .

## 5.6 Deletion

Deletion is performed in much the same way as drag and drop. Figure 5.7 shows how part of an equation can be deleted by dragging it to the trash. The selected part of the equation is removed and the equation is re-recognised without the deleted section. A cloud of smoke is used to provide feedback to the user that they have deleted part of the expression. By utilising the same gesture and metaphor of drag and drop for deletion the calculator makes economical use of the few gestures the calculator supports, the user interface is less complex and the user has fewer gestures to learn.

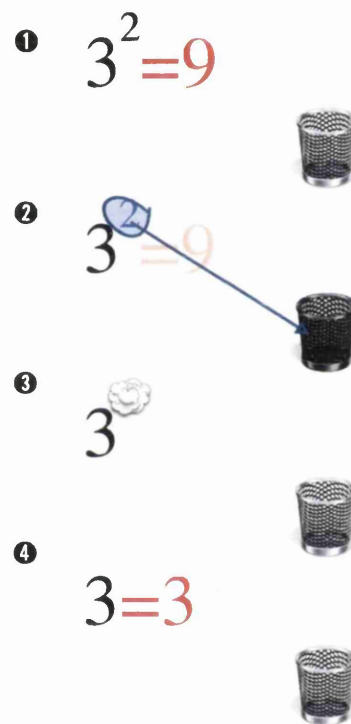


Figure 5.7: Deleting part of a calculation

1. Starting expression:  $3^2$
2. Dragging to the trash

To delete a selection the user simply circles part of the equation and drags it to the trash in the bottom right of the screen

3. Deleted

The user is provided with a smoke cloud and a “poof” sound as feedback from the deletion.

4. Finished: 3

The final expression is recalculated and morphed after the deletion.

There is also a clear button (shown as a cloud of smoke) in the bottom-left corner of the user interface that clears the whole current expression, resulting in a completely clear screen.

## 5.7 Partial expressions

Part of what allows the calculator to compute answers as the user enters the expression, is the ability of the calculator to handle partial expressions smoothly. When a mathematical expression is recognised, missing components are filled in with placeholders, which appear in the user interface in red. These placeholders are then adjusted intelligently so that the mathematical expression is correct. This ability is a part of the *declarative interaction* the calculator provides, which blurs the distinction between input and output and allows the user to utilise this potential for ease of use, like completing simple incomplete expressions, and for more powerful solving capabilities.

Figure 5.8 shows four examples of partial completion, each with a black number and a red symbol added to complete the expression:

- 1.  $3 \times 1 = 3$
- 2.  $\frac{1}{2} = 0.5$
- 3.  $(\frac{4}{1} + 2) = 6$
- 4.  $\frac{3}{2} \times 4 = 6$

Figure 5.8: Simple partial completion

Figure 5.8 shows some simple examples of partial completion. In each case the red symbols have been added to the black mathematics written by the user to ensure that what the user sees is mathematically correct. An answer on the right-hand side is also added ensuring the entire expression is mathematically correct.



1. The multiplication is completed with its identity, converting  $3\times$  into  $3 \times 1$ .
2. The incomplete fraction is completed with an added numerator, converting  $\frac{\phantom{1}}{2}$  into  $\frac{1}{2}$ .
3. Both the denominator and the missing closing bracket are added to the user's input.
4. A multiplication symbol is added to disambiguate a fraction followed by another symbol. This is for consistency so that  $\frac{3}{2}4$  and  $\frac{3}{2}\pi$  are both clearly multiplications. This stops  $\frac{3}{2}4$  being confused with  $\frac{3}{2}+4$ , which is how it is sometimes written.

This facility of handling partial expressions is extended by allowing the user to write an equals sign. By writing an equals sign, the user is able to add mathematics on both sides of the equality and get the calculator to correct the partial expressions, providing answers to mathematic calculations that would otherwise need to be rearranged.

The calculator in a declarative fashion (i.e. a lack of distinction between input and output) completes all the expressions such that the user can easily compute the answer to questions like “what power of 2 is 100?”. Figure 5.9 shows some examples of how this ability can be used to compute more complex sums very easily without rearranging the expressions. These examples are described below.

1. A simple multiplication is completed to correct the mathematical expression. The user wrote  $3\times = 12$  which is then corrected to  $3\times 4 = 12$ .
2. In the same fashion, an incomplete fraction that is missing a numerator is completed. This is an example of a common problem “what divided by  $x$  is  $y$ ?” that users can solve without rearranging.
3. An exponent is completed, without the user knowing the correct logarithm rules to get the answer. In this example the fact that there is a missing exponent needs to be indicated (in this case by using a bracket). Without any symbol in the exponent the calculator assumes there is no exponent. Thus  $2^{\phantom{()}} = 100$  and  $2 = 100$  are completed very differently. The latter is completed as  $2 = 100 - 98$ , and the former as a solution to “what power of 2 is 100?”.
4. An incorrect sum  $3 = 5$  is corrected with the addition of a  $-2$  on the right-hand side. All similar corrections happen on the far right-hand side to minimise confusion about where corrections appear.
5. Some mathematics with unknowns can be completed in an infinite number of ways. The calculator always picks a “simple” answer. In this case a multiplication is completed using an identity.
6. A factorial is completed as close as it can get to the answer and the remainder is added in. The user could be trying to solve “What fac-

- ❶  $3 \times 4 = 12$
- ❷  $\frac{18}{2} = 9$
- ❸  $2^{(6.644)} = 100$
- ❹  $3 = 5 - 2$
- ❺  $12 = 12 \times 1$
- ❻  $50 = 4! + 26$
- ❼  $\sqrt{81} = 9$

Figure 5.9: More complex partial completions

torial is equal to 50?", the calculator provides as close as it can get, and the mathematics the user sees is still correct.

7. The inverse of a square root is calculated to correct the input. The calculator also handles the similar expression  $\sqrt{?} = -9$  in a sensible way correcting it to  $\sqrt{1} = -9 + 10$ .

## 5.8 Hiding the answers

As a teaching aid the calculator allows the user to hide the answers. This allows a teacher to pose questions and to get responses to a mathematical problem before the answer is shown. It can also be used to allow users to enter possible answers to see if they are correct.

Figure 5.10 shows two examples of how the calculator interface displays hidden answers. In the first example, the expression is  $3 \times 4 =$  followed by a dashed red box. In the second example, the expression is  $\frac{\text{[dashed box]}}{2} = 9$ , where the numerator is inside a dashed red box.

Figure 5.10: Hidden answers

Figure 5.10 shows two examples of how this appears to the user. The red numeric corrections are replaced with dashed red boxes. By clicking the toggle in the top-left corner of the user interface the user can quickly switch between this mode and the standard display.

If the user writes an answer in, the expression it is recalculated as usual, any new corrections are then displayed in the same style with a red outline box. If an equation is written that does not need any corrections, then this is highlighted by a big green tick, shown in Figure 5.11. This happens, for example, when the user fills in the correct answer for a hidden correction.

For teaching and group-use the ability to hide the answers until required provides a richer interaction both with the calculator and the class. A teacher can use the calculator to show a mathematical relation and then

by hiding the answers can pose questions like “What happens if I alter this number?” A missing number is rendered as a box, prompting the user to enter a number in it. When this is combined with the ability to undo, it allows users to try an answer, rewind and try a different one, providing a useful tool for a teacher using it with a class or group.



$$3 \times 4 = 12$$

Figure 5.11: A correct answer

## 5.9 The dock

The dock gives the user the ability to save and reuse and work on multiple mathematical expressions. It contains the thumbnails of the calculator’s stored expressions that the user can access and edit. The dock sits on the left hand side of the user interface and can be grown, shrunk or hidden to the user’s liking.

Each expression stored in the dock is selectable by clicking on the tile in the dock. The current main expression is then switched to the one that was selected in the dock. Each expression in the dock is self-contained and has its own data and history. As an expression is edited, the corresponding dock thumbnail is updated in real time in keeping with *projection*, ensuring that there is no inconsistency between the dock thumbnail and the current expression.

Figure 5.12 shows how mathematics are stored in and used from the dock.

1. The dock user interface is shown on the left of the screen. The current expression is edited and interacted with on the right, a thumbnail of this larger mathematical expression is shown highlighted inside a white tile in the dock.
2. The user saves part of an expression by dragging the selection in the current mathematics to the dock, creating a new tile in the dock corresponding to the saved expression.
3. In the dock the new saved expression is shown with a grey background, differentiating it from the current expression being edited. The answer shown in the thumbnail has already been automatically calculated and filled in, meaning the mathematics even within the dock thumbnails are consistent or *projected*.

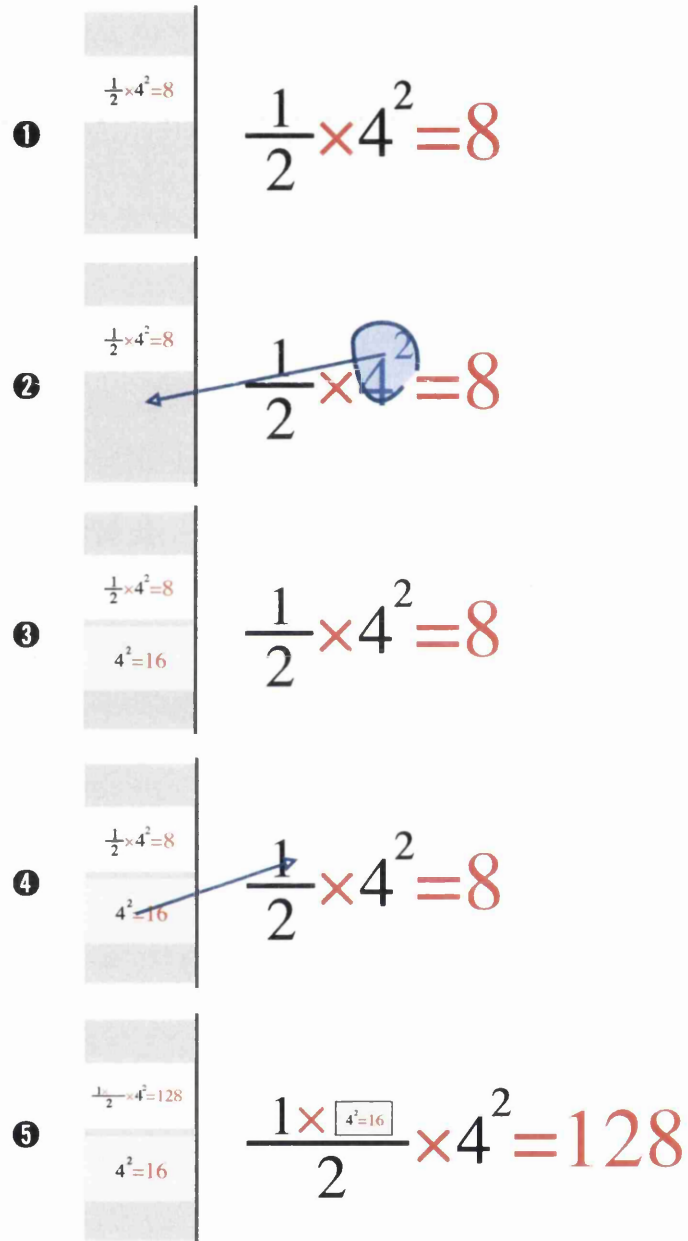


Figure 5.12: Using the dock

Saving parts of an expression to the dock is as simple as finishing a drag and drop action in the dock. The selected part of an expression will be copied and inserted into a new saved expression.

4. Saved expressions can be dragged back into the current expression or be deleted by dragging them to the trash.
5. When a saved expression is dragged into the current expression being edited, it is inserted as an immutable box with the value of the contained expression. The main expression is recalculated and the dock tile is also kept up-to date. The final sum with the inserted saved expression is  $\frac{1 \times [4^2]}{2} 4^2$ .

## 5.10 History

Instead of providing a discrete undo similar to most modern user interfaces, the calculator provides an undo that is continuous and smooth. The interaction with the undo system provides feedback that is similar to the morphing feedback used to provide continuity between the user's writing and the type-set calculation. The interface is presented as a clock (seen in the top-right corner of the user interface) that allows the user to manually "set the time." As the user edits the expression the "time" ticks forward, about a quarter "hour" for each edit. More recent versions of the clock have been made more abstract in order to not be mistaken with an actual clock.

To undo, or go back, the user grabs the clock and rotates the hands backwards. The clock "time" and the calculator state are linked, providing a *projected* user interface. To undo to a previous point the user only has to remember the "time" they want to go back to and rotate the clock's hands back or forward to get to that time, or the user can scrub back and forth reviewing the history of the mathematics until they find the point they want.

As the hands of the clock are turned in either direction the equations morph from state to state, backwards and forwards in time. The fluid morphing of the equation as this happens provides a continuity through time and through the history of the equation that is easy to follow. The morphing contains all the edits, drags and deletions that have happened to the expression, including some of the effects like the "poof" of smoke for deletion. (Imagine Figure 5.3 but animated and scrubbable through like a movie.)

Scrubbing backwards and forwards in time allows the user to undo or review their previous work. If the user rewinds the clock this undoes previous edits and the user can continue editing from the current visible state. The user is able to review all their previous work without undoing by rewinding backwards and then winding forwards to the most recent state.

## 5.11 User interface concepts

### 5.11.1 Handwriting

Handwriting is an important component of meeting the “it should work like paper” guiding principle. By providing a user interface that works like paper, users are able to make use of their existing knowledge and skills for manipulating mathematics on paper.

The calculator provides a modelless interface, which resembles a sheet of paper. Any writing causes “ink” to be drawn on the screen. The user is able to write everywhere. The handwriting recognition works with standard handwriting as written on paper, thus there is no need to learn a special set of gestures or to learn a different method of handwriting, such as a simplified alphabet would require.

The user can write on-top of and over current typeset mathematics which allows for easy insertion into and modification of the mathematics. A selection is created when the user’s gesture forms a loop around several symbols. This selection can then be dragged using the pen to anywhere on the screen. Other than this selection gesture every pen stroke and touch of the screen is assumed to be writing.

Cursor manipulation can be the most tedious aspect of editing an equation in 2D equation-editing systems. A benefit of a pen-based system is that there is no cursor, therefore the user can write anywhere without awkward cursor movement or placement. This is especially useful for mathematics where editing and writing happens in lots of different places and often not sequentially.

### 5.11.2 Morphing

Providing continuity to the user with continuous feedback, is a key part of the user interface, this is encapsulated by the *continuity* principle. Feedback is provided through typeset annotation and animated morphing of the mathematical expressions. This feedback provides the user with continuity as the calculator’s state changes and makes it clear what the calculator is doing and how it relates to the user’s input.

Annotation provides immediate feedback about the success of the character recognition, replacing the hand-written characters with typeset characters stretched to the same size and bounding box. Animated morphing provides the continuity between what the user has written and the final typeset expression. After the characters have been recognised and replaced with typeset versions they are still in the rough position and shape that the user wrote them. The calculator then smoothly morphs these characters into neat positions to form the final neat typeset expression.

### 5.11.3 Feedback

Immediate feedback is continually provided to the user so that they are always aware of what is happening and what mathematics is being calculated. The calculator performs several state changes based on timers, that provide the user time to finish any edits they make. All state changes are *projected* and provide immediate feedback. *Projection* also means that the user interface is never inconsistent, as soon as the user edits the mathematics the current answer disappears, and as soon as the user finishes editing a new answer fades in.

Without the *projected* feedback and the *continuity* of morphing the user would face sudden state changes and an often inconsistent and confusing user interface.

### 5.11.4 Freeform editing

The calculator provides a very flexible way of editing mathematics. Adding new, deleting, or rearranging symbols is performed in an unrestricted and natural way. Editing is not restricted and any part of the expression can be changed. Once the user has selected something, they can move it wherever they want, without having to worry about any constraints on their interaction. This is the result of the *WYSIWYE* principle, because of the lack of restrictions the user interface is both easier to use and more powerful. Many of the examples of editing mathematics earlier in this chapter would not be possible in a non-*WYSIWYE* user interface.

### 5.11.5 Calculation

The mathematics of the calculator are based on the *declarative interaction* principle. The central idea being that the input and output are treated with equality. The calculator non-destructively completes the user's work, simultaneously correcting or solving any arithmetic mistakes or omissions. Thus ensuring that the expression the user sees is always mathematically correct.

The calculator provides a sensible answer at any point in time for any input: partial, complete or wrong. This allows the calculator to provide immediate *projected* feedback to the user without delay. Thus the user is continually kept informed of what the calculator is doing and the current mathematical solution to what they have entered.

The user is also able to utilise the declarative nature of the calculation to do mathematics in a more natural way. They are able to solve simple sums, such as  $4 = 27$ , without having to rearrange the sum. Not only are they able



to solve simple sums more easily, but they can also solve complex expressions that they would not have otherwise been able to do.

#### 5.11.6 Exploration

There are many different aspects of the calculator that encourage exploration and play. All of the *flow* principles engender a fluid and playful user interface. *Projection* ensures that what the user sees is always correct and can quickly and easily respond to the results. *Continuity* means that the user rarely loses track of what is happening and does not lose their flow. *WYSIWYE* allows the user to change anything as they see it, without the extra cognitive effort of dealing with constraints. Finally a *declarative* interface makes the underlying mathematics more immediate and concrete. These all contribute to a user interface that is great for exploring and learning about mathematics.

Undo is controlled through an interface that is linear and smooth, this provides *continuity* so that when undoing or redoing, the user does not experience any jarring transitions between the different states. The smooth undo is similar to the morphing feedback that is used to provide continuity between the user's writing and the typeset calculation.

### 5.12 Summary

The user interface of the calculator has been described in a step-by-step manner, showing how the pen-based user interface can be used in flexible and powerful ways. These descriptions, though comprehensive, do not convey the fluid, interactive nature of the calculator. Video demonstrations of users interacting with the calculator and of the user interface are available online<sup>1</sup> are more persuasive. However, nothing compares to the experience and enjoyment of using the user interface yourself, preferably with a pen or finger user interface.

The combination of the simplicity of the pen interface, the *projected* immediacy and visibility of what is being computed, the *continuity* and fluid morphing between states, the *WYSIWYE* ease of editing, the *declarative* correctness of incomplete mathematics transforms the way users interact with the calculator. The new calculator enables mathematics to be explored in a highly forgiving and experimental interface that is fun to use.

---

<sup>1</sup><http://www.cs.swan.ac.uk/calculators/>

## Chapter 6

# Implementation

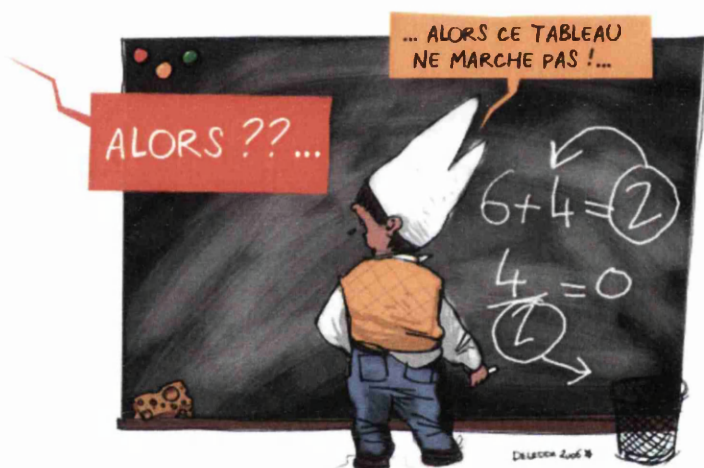


Figure 6.1: “RealCalc la calculette du futur” © J. A. Deledda

This chapter provides an overview of the implementation of the calculator. For the actual implementation, the Java source code and data files as well as a version ready to run and use as compiled Java JAR file, are available online at <http://www.cs.swan.ac.uk/calculators/>.

A short literature overview of the specific components of the calculator’s implementation is provided. However this thesis is not intended to primarily contribute in these areas and the literature overviews have been shortened.

### 6.1 Tools

The calculator was programmed in Java, an object orientated language which produces cross-platform runnable applications. However, originally the calculator also made use of *Cocoa*, a specific Mac OS X technology,

as this provided a simpler and more familiar way for the author to create graphical user interfaces. This has been removed to make the calculator completely cross-platform. The calculator is now written solely in Java. By programming the calculator in a cross-platform language, it was hoped that the system may find wider usefulness, beyond a research project. Indeed the calculator is now published by Promethean.

Effort was made to keep the design of the whole system well structured using object orientated techniques. As such, part of this the code uses the Model-View-Controller pattern [Gamma et al., 1995] extensively, which keeps the user interface separate from the model. The use of this pattern meant that the replacement of the Cocoa user interface with a Java user interface was a straight forward task, because the interface between the user interface and the core structure had already been abstracted.

## 6.2 Overview

The core of the calculator is summarised in the remainder of this chapter. The calculator consists of pen-based interaction with a single mathematical expression. Omitted in this description of the calculator's implementation is the majority of the actual code-level or Java-specific implementation details. This core of pen-based interaction with mathematics consists of four distinct processes, shown in Figure 6.2. Processes A to C are stages within the mathematical engine and occur sequentially. The user interface, process D, is layered on-top of these core stages and provided continuous interaction and feedback with the core processes.

These processes are:

- A. Symbol Recognition — A symbol recogniser that reads the strokes written by the user on a pen-based device and determines which symbols have been written.
- B. Expression Recognition — An expression processor that takes the symbols with their relative positioning and size and recognises the mathematical expressions they represent.
- C. Calculation — A calculator that provides the completed mathematically correct expressions.
- D. User Interface — A user interface that provides seamless user interaction, from the initial handwriting input to displaying the result, including providing the feedback during this process to the user, so they can understand what is being computed.

The remainder of this chapter is divided into four sections that each discuss the individual stages of the process described above.

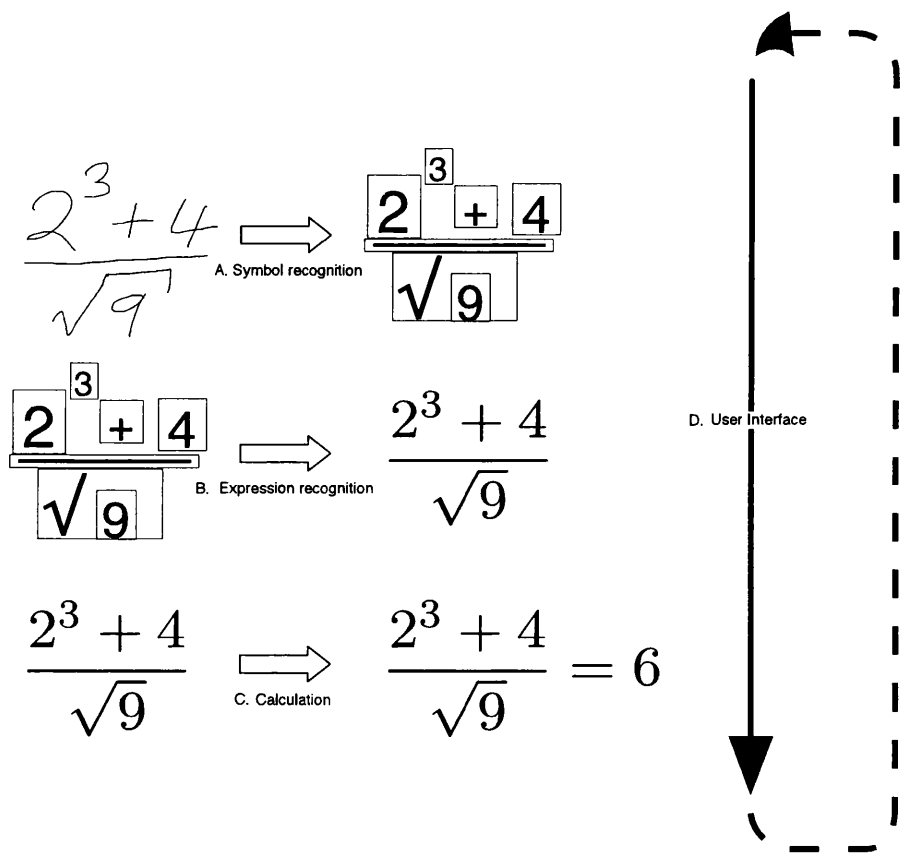


Figure 6.2: The calculation process

### 6.3 Symbol recognition

The first stage of any calculation using the calculator is handwriting or symbol recognition. This is the process where the user's writing, which is provided to the calculator as vector data from pen movement, is recognised as individual symbols. These symbols are then passed onto the next stage which recognises mathematical expressions. During this recognition process, the user interface provides immediate feedback by replacing the handwritten strokes with typeset symbols.

Handwriting recognition is a well-researched area. Many implementations provide very good recognition rates and there are lots of different approaches to the symbol recognition problem. Plamondon and Srihari [2000] provide a comprehensive overview of the area. Some of the different approaches to the handwriting recognition problem are highlighted below.

Persoon and Fu [1977] uses signal processing, viewing symbols as a signal processing problem, where the closed contour of a pattern is considered as a periodic signal. Odata et al. [1982] describe a simple statistical method for online character recognition. Starner et al. [1994] use hidden markov models, used for recognising continuous speech to recognise online cursive handwriting. Shaw [1969] uses a syntactic method or Picture Description Language. PDL uses straight line segments as primitives and grammar rules to describe how line segments can join together. Chan and Yeung [1998] use a structural method in which the unknown pattern is repeatedly deformed if it does not match any of the classes. Pavlidis et al. [1996] use a physics-based shape metamorphism method and casts the problem as an energy minimization problem. Tappert [1984] uses elastic matching and dynamic programming.

Instead of using a handwriting recognition library the calculator implements its own system for symbol recognition. There are several reasons for this, not least that mathematical symbol recognition is a different problem to textual handwriting recognition (handwriting recognition for mathematics has several special features that mean it is a different problem than normal handwriting recognition). The main reasons are summarised below:

- A bespoke system is free of restrictions, whether legal or technical, that hinder other libraries.
- Mathematics is not typically written in a joined-up cursive style. Therefore the recogniser does not need to address the more difficult problem of recognising cursive writing.
- Mathematical writing contains many odd symbols, including symbols that vary in scale and shape, like the  $\sqrt{\quad}$  symbol. Mathematics also uses a restricted character set, which makes the recognition problem simpler. A specialised recogniser can be tweaked specifically for mathematics.

- The relative location of symbols and sizes vary greatly in mathematics, whereas text is generally uniform and written in a line.
- In order to provide better affordance with paper, the symbol recogniser recognises handwriting as written, instead of using a single pen stroke based system like Graffiti [Fleetwood et al., 2002].
- Technical reasons of integration. Using my own system, as opposed to say Microsoft's handwriting recognition engine, means that the entire calculator is cross-platform.

Handwriting recognition has been a usability issue for the calculator. If the handwriting recognition fails then the whole user experience is undermined. Currently the handwriting recognition has an accuracy of 90–95% depending on the user. Unfortunately this is still the largest cause of users not finding the calculator easy to use. It is a shame that such a critical component of the user experience is not part of the whole positive experience. In retrospect relying on a third party library might have been better, despite all the valid reasons for not doing so.

However, the symbol recogniser is simple, fast, and fairly accurate. It uses a simple model matching method, where sets of strokes are recognised by comparing them to models.

Model matching is based on the assumption that hand-written characters are distorted realisations of ideal models. At the training stage, templates or ideal models are recorded. Then at the recognition stage these are compared to the data to be recognised. A distance measure is then generated between the data and the template, using features from the data. This distance metric is used to generate a likelihood value indicating the closeness of the match.

### 6.3.1 Character models

Each symbol has many different models. The character models are stored as stroke data, each model possibly containing several strokes. For model matching to work well, the ideal models are critical. The clearer and more distinct the models are, the better the recognition works.

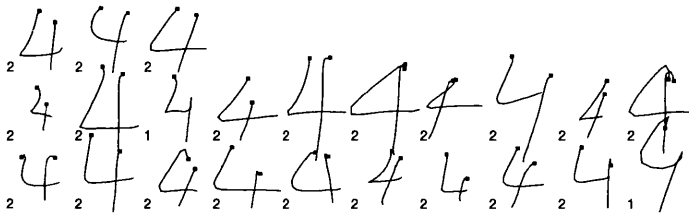
The calculator currently uses a total of 237 models that cover the 22 different symbols the calculator recognises. Most models are made from one or two strokes. The maximum number of strokes per model is 3 (which is for the  $\pi$  model). Figure 6.3 shows the 23 models used to recognise the symbol '4'. The dots show the start of the individual strokes, and the numbers bottom-left of the model show the number of strokes used. As can be seen the symbol '4' can be written in several different ways using one or two strokes.

All the models are normalised to make the matching algorithms faster. The data points along the strokes of the models are resampled so that they are

spaced evenly in distance instead of time.

When hand-written input data is matched against a model, the input data is scaled uniformly so that the largest dimension of the template is matched by the size of the same dimension of the input data. Note that normalising to a specified height or width as Tappert [1984] suggests causes problems with symbols like  $-$ ,  $1$  and  $.$ .

Normalising in this way for matching does not solve all problems, for example small dots still cause problems (any symbol can be scaled to a small dot and still match with a low distance metric). Dots are therefore treated as a special case and solved by recognising any stroke drawn under a certain size as a dot (or semantically as a decimal point).



**Figure 6.3:** The models used for the character ‘4’

### 6.3.2 Segmentation

The majority of symbols a user writes are single stroke symbols like  $6$  and  $-$ , but some symbols like  $4$  and  $=$  can be composed of two separate strokes of the pen. Segmenting these strokes into separate characters is the first part of recognising a symbol.

The segmentation stage of the symbol recognition is a brute-force exhaustive search through all combinations of grouping the input strokes into symbols. Symbols are assumed to be composed of sequential strokes: that is, strokes are only combined into a symbol if they are temporally consecutive, thus a user cannot go back and add an extra stroke to a previous symbol (for example to dot an  $i$ ).

To segment the strokes, a combination of temporal ordering segmentation and a simple spatial check are used. The symbol segmenter holds a queue of the strokes entered by the user. As a new stroke is written by the user it is added to the end of the queue of strokes to be segmented. When the size of the queue of unrecognised strokes reaches twice the number of maximum strokes in any symbol model, it is in principle possible for the segmenter to segment the first two symbols without error as there is no combination of two symbols that could require more strokes. When this condition is

reached, or if a time delay triggers first, the initial symbol is segmented and recognised.

To do this, the segmenter recursively tests all possible combinations of strokes in the queue that can be segmented to create two symbols. The segmented combination with the lowest total sum of distances between segmented symbols and models is chosen. The strokes used for the first symbol are then removed from the queue and are visually replaced in the user interface with the recognised symbol, appropriately positioned and scaled.

This is a limited version of brute force temporal ordering, but by restricting the segmenter to two symbols, the symbols can be recognised as the user enters them, providing valuable feedback while the user is writing.

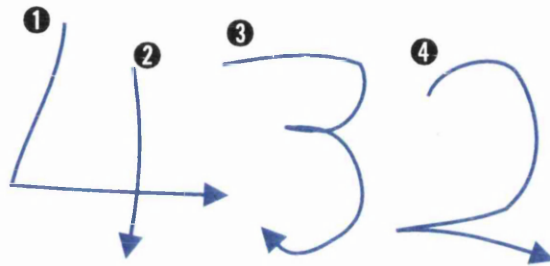


Figure 6.4: Input strokes for segmentation

Figure 6.5 shows the debug output from the segmentation and recognition of the strokes '432' from Figure 6.4. The recursive test of all possibilities for the first two symbols is shown, segmenting up to a maximum of two strokes per symbol.

The lowest total sum for recognising two symbols is for the symbols 4 and 3 (a total combined distance of  $162 + 919 = 1081$ ). These symbols are the most likely combination of two symbols from any segmentation of the first four strokes. This segmentation groups the first two strokes into one symbol and the third stroke into a second symbol. A different segmentation (the first segmentation tried) of one stroke per symbol would provide the symbols 0 and 1 (a total combined distance of  $5653 + 64 = 5717$ ).

In the debug output the best symbol and its cost for each segmentation is shown on the right side of the output. As an example, 0 is the most likely symbol that matches the first stroke with a cost of 5653, this is shown on the first line of the output.

Once recognised as the initial symbol, the symbol is displayed on the screen and the strokes that are part of the symbol are removed from the list of strokes to be recognised. In this case the first two strokes, which are recognised as a 4, are removed and a typeset 4 replaces the strokes in the user interface.

Once a symbol has been recognised, the stroke data is discarded and the



```

1st Symbol: 1 strokes best: 0 (5653.3955)
  2nd Symbol: 1 strokes best: 1 (64.828865)
  2nd Symbol: 2 strokes best: 4 (7712.3564)
1st Symbol: 2 strokes best: 4 (162.56503)
  2nd Symbol: 1 strokes best: 3 (919.0894)
  2nd Symbol: 2 strokes best: 4 (8252.45)

**Recognised: 4 (162.56503)

delay

```

**Figure 6.5: Segmenter debug output**

symbol is immutable. The symbol cannot be edited or altered in anyway, but it can be moved or deleted.

From the user's perspective, the only restriction they have to conform to is that any symbol must be written in one go. That is, *i* symbols must be dotted and all + symbols crossed before the next symbol is started. The symbol segmenter has one adjustable parameter, the time delay between drawing a stroke and when the recognition is started. The longer this time is, the more time the user has to enter multi-stroke symbols like = and 4. However, the longer the delay is the longer the system will take in providing feedback from the symbol recognition. After the time delay, the segmenter recognises the first symbol on the queue regardless of the number of strokes the user has written. A long delay is more suitable for children and slower writers, but slows the calculator down, the actual parameter can be adjusted as a personal preference.

### Composite symbols + and =

The segmentation of symbols is generally good, however the composite symbols + and = pose a specific recognition problem. These symbols can both be accurately recognised in two different ways. The decomposed strokes of these composite symbols are themselves symbols. For example strokes of a + symbol can also match the two symbols, - and 1. In fact the recognition of the decomposed versions of these symbols are usually more likely, because the spatial relationship between the two strokes is more flexible.

The solution used for this problem is to provide a set of symbols that each composite symbol overrides. Thus if a segmentation of two strokes best matches an = symbol but can also be recognised separately as two - symbols, the cost of the = symbol is adjusted to make the = more likely than two separate - symbols. This method is possible because of the restricted character set of the calculator. This problem and solution are unique to

mathematics recognition, as the extra structure and context of text allows easier recognition.

### 6.3.3 Recognition

Simple model matching is used to recognise all written symbols. Experimentation with training data showed that for the set of symbols the calculator needed to recognise, simple model matching provided comparable recognition results for the same number of models as other more complicated matching methods like elastic matching. With good models and the restricted character set, simple model matching provides a good recognition rate for writing mathematics. The simplicity and speed of simple model matching also allows more models to be used in symbol recognition with much less overhead.

The models and the pattern to be matched are vector path data. These paths are matched comparing each point in the path of the pattern with its equivalent point on the model. The matching distance between two points on a stroke is based on the point's spatial position and gradient in the path. After experimentation, the distance equation shown in Equation 6.1 was decided on. Matching further path information (such as curvature) did not substantially improve the model matching accuracy.

$$d(i, j) = \alpha \min\{|\phi_i - \phi_j|, 2\pi - \phi_i - \phi_j\} + \beta((\hat{x}_i - \hat{x}_j)^2 + (\hat{y}_i - \hat{y}_j)^2) \quad (6.1)$$

where  $\phi_i$  is the slope angle of the curve, and  $\hat{x}_i, \hat{y}_i$  are the normalised coordinates.

$\alpha$  and  $\beta$  were adjusted programmatically once all the models have been entered so that the distance between different symbols is maximised whilst the distance between the same symbols is minimised. The values for  $\alpha$  and  $\beta$  used to provide the best recognition for the models used by the calculator are 10 and 1 respectively when the models are scaled so that they have a maximum dimension of 1.

The symbol recogniser provides a single symbol to the expression recognition stage. It does not pass any additional information, such as probability information. The expression recognition stage therefore does not and can not provide any backtracking in order to choose different symbols using different probabilities based on context of the symbol in the expression.

## 6.4 Expression recognition

The purpose of expression recognition is to determine the meaning of the expression given the symbols and their relative placement and sizes. This is



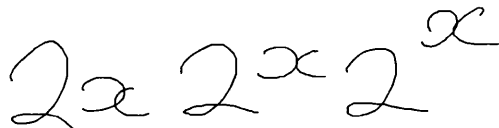
the second stage in recognising a mathematical expression.

### 6.4.1 Some difficulties

Mathematical expressions can be hard to understand. Mathematical notation is often very subtle, and continually makes use of the relative sizes and placements of symbols. Without the context of the mathematical expression it is sometimes hard, even for practised humans, to comprehend some layouts of mathematical notation.

Some of the difficulties that occur are:

- Ambiguous symbols — A dot can represent a decimal point, a multiplication or an annotation ( $3.4$ ,  $a \cdot b$ ,  $\dot{x}$ ); a horizontal line can be an infix subtraction operator, a prefix negation, a fraction bar or part of a more complicated symbol such as ‘=’ or ‘ $\leq$ ’.
- Ambiguous spatial relationships — Implied operations are hard to determine as the meaning is implied by a rough spatial positioning, as shown in Figure 6.6.



**Figure 6.6: Ambiguous powers**

- Ambiguous expressions — Sometimes even simple mathematics can be misinterpreted as a result of implicit operators or knowledge of the context. Equation 6.2 shows a simple example of how a simple sum could be misinterpreted.

$$\text{Does } 2\sqrt{4} \text{ mean } 2 + \sqrt{4}, 2 \times \sqrt{4} \text{ or } \sqrt[2]{4} ? \quad (6.2)$$

Zhao et al. [1996] discuss more determinable and indeterminable parse trees, and Martin [1971] provides several examples of ambiguities and indeterminable expressions; one of his examples is shown in Equation 6.3.

$$\text{Does } \sum_{i=5}^{10} i + Y \text{ mean } \sum_{i=5}^{10} (i + Y) \text{ or } \left( \sum_{i=5}^{10} i \right) + Y ? \quad (6.3)$$

### 6.4.2 Alternative solutions

Blostein and Grbavec [1996] and Chan and Yeung [2000] all provide good comprehensive overviews of the different approaches to mathematical expression parsers. There is a wide variety of different solutions and a brief

overview of the approaches with brief summaries of some of the key contributions to this field is now provided.

Littin [1993] uses a SLR(1) parser with additional tests on the geometric relationship of the symbols. Anderson [1977] uses coordinate grammars for both arithmetic and matrix mathematical notation recognition. In 1967, one of the earliest recognition systems, Martin [1967a], uses concatenation operators that offer a geometric approach. Chang [1970] uses structural specification schemes, based on operators that divide the pattern into one or more sub-patterns. Lee and Wan [1995] and Twaakyondo and Okamoto [1995] use a procedurally coded system that has no explicit grammar or structure. Chou [1989] uses a stochastic grammar to recognise noisy typeset equations. Graph rewriting is used by Grbavec and Blostein [1995], Blostein and Schüerr [1999]. Faure and Wang [1990] use a top-down data driven segmentation. Zanibbi et al. [2002] outline an implementation that makes extensive use of trees, tree transformations and the directionality of notation. Eto and Suzuki [2001] use minimal spanning trees to reconstruct the formula.

### 6.4.3 The solution

The expression recognition algorithm described here is based on a structure specification scheme similar to that of Chang's [1970]. It uses a structural specification scheme for special operators (such as division and roots) that divide the expression into sub-expressions, then uses a recursive descent parser to handle linear expressions when there are no more special structurally dividing symbols. The algorithm also provides a new way of allowing one operator to dominate another and a special method of handling non-explicit operators such as exponentiation.

### 6.4.4 Structure specification schemes

Chang [1970] uses a structure specification scheme to recognise the structure of mathematical expressions. The scheme could be thought of as a two-dimensional grammar allowing the specification of certain two-dimensional patterns. Each grammar rule or pattern can be composed much like templates are composed in a template-based editor but the composition happens automatically. The recognition time for this pure structural scheme is  $O(n^2)$  for an input expression of  $n$  symbols.

Structural specification schemes are based on operators that divide the pattern into one or more sub-patterns. Figure 6.7 shows two different operators and their sub-patterns as shaded areas. According to Chang, the structural specification scheme is based upon the assumption that some or all primitive components of a collection of patterns are operators, and that the structure of a pattern can be constructed by analysis and comparison of these opera-

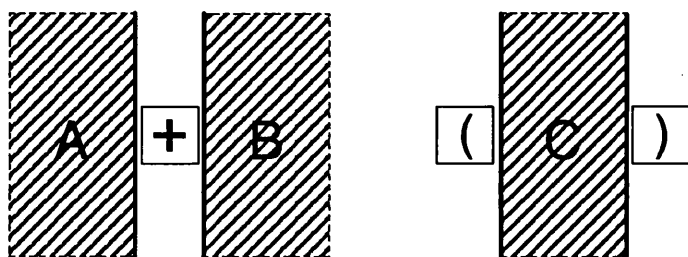
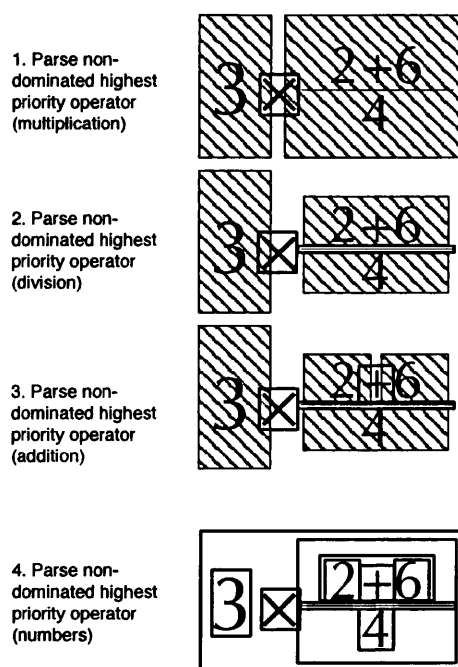


Figure 6.7: Example operators and applicable ranges

tors. Each operator has a division rule and applicable ranges. For example, the  $+$  operator in Figure 6.7 has ranges  $A$  and  $B$  as its operands.

Figure 6.8: An example of how Chang's algorithm parses  $3 \times \frac{2+6}{4}$ 

Chang uses the concept of operator *domination*, where an operator dominates another if and only if the latter is in the range of the former and the converse is false. Therefore  $+$  dominates  $-$  in the pattern  $a + \frac{b}{c}$ , whereas  $-$  dominates  $+$  in the pattern  $\frac{a+b}{c}$ . Thus a combination of dominance and precedence can be used to define an ordering relation on the operators. Any non-dominated operator has precedence over a dominated one.

### 6.4.5 The algorithm

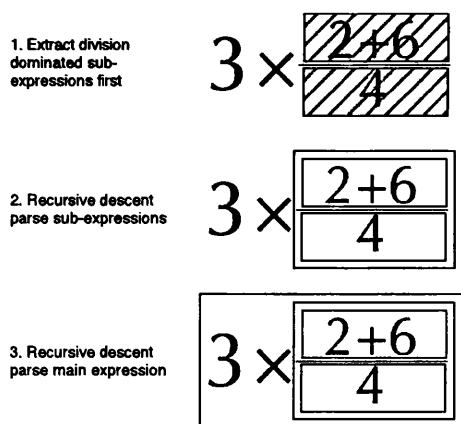
The expression recognition algorithm is implemented in two parts: the first as a top-down structural specification scheme, the second as a handcrafted recursive descent parser. The majority of written mathematics is still linear, and a traditional recursive descent parser handles all of the mathematics that could be written without any two-dimensional positioning. The limited vocabulary of the calculator means that with the exception of roots, exponentiation and fractions, all the mathematics is handled by this parser. The operator precedence is implicit in the structure of the recursive descent parser, either an operator is found and the pattern split into the appropriate sub-patterns, or the parser attempts to find the next operator.

In order to correctly prioritise two-dimensional operators the algorithm pre-parses the sub-patterns of *dominating* operators. This is a simplified notion of dominating operators that uses structural matching for only a few operators; Chang's method, while more powerful, requires a more complex parsing strategy. Correctly identifying dominated symbols allows a simple recursive descent parser to be used for the majority of the expression. Unlike Chang's method our algorithm does not need to prioritise and keep track of operators and sub-patterns in a queue, which makes the algorithm much simpler and faster.

This pre-parsing syntactic stage thus takes a two-dimensional dominating operator and removes its sub-patterns from the expression before any recursive descent parsing starts. Essentially, it can be thought of as cutting out the dominated symbols and parsing them separately. For example, a division operator parses its numerator and denominator first, removing the numerator or denominator from the main expression. Once all dominated symbols are removed from the main expression this leaves a simple expression which can be parsed by recursive descent. Only operators that make use of the two-dimensional nature of expressions can dominate others.

Figure 6.9 shows how the new algorithm parses  $3 \times \frac{2+6}{4}$ . First the dominating two-dimensional operators (such as division are pre-parsed), the dominated sub-expressions ( $2 + 6$  and  $4$ ) are separated out, and are then fully parsed using recursive descent. Finally with no dominating operators left the algorithm performs a normal recursive descent on the main expression. Compare this with figure 6.8, that shows Chang's method which has to maintain a queue of operators ready to be parsed.

Division is handled by pre-parsing because its syntax (the shape of the symbol) defines the range of the operands or sub-patterns explicitly. Thus division can be easily computed first before the rest of the expression is parsed. Horizontal line and square root symbols are sorted in order of width, as an estimate of priority, then each symbol from longest to shortest is parsed. Square roots encapsulate the symbols below the root sign, and horizontal lines capture the symbols above and below into a numerator and denomi-



**Figure 6.9:** An example of how the new algorithm parses  $3 \times \frac{2+6}{4}$

nator. Horizontal line symbols without any numerator or denominator are treated as subtraction or negation signs. This process makes use of the assumption (that is generally correct) that the fraction bar or root symbol are ordered in mathematical priority by their width.

Unfortunately the other two-dimensional operator the calculator supports, exponentiation, cannot be treated in this way. Although an exponent is explicitly defined by the baseline structure of the mathematical expression, the baseline of an exponent is dependent on the rest of the structure of the parsed expression. Thus exponents cannot be determined before any parsing has happened.

#### 6.4.6 Exponentiation

Exponentiation causes more problems for syntax directed mathematical expression recognition than for other methods such as graph rewriting. This is because exponentiation provides no explicit syntax for the parser to be directed by. So the top-down syntax direction struggles to extract powers in order to handle them first.

Chang attempts to solve this problem by limiting the area in which powers can be written. His two-dimensional division rule can be seen in Figure 6.10. However, completely structural approaches, such as Chang's, fail with even simple mathematical expressions containing exponents. The rule in Figure 6.10 assumes that only whole syntactic units are parsed and not numbers composed of multiple digits. Chang's pure structural specification scheme is incapable of parsing simple expressions like:  $\frac{1}{2}^2$  or  $12^34$ .

To solve this problem, the baseline structure is parsed left to right, once dominating operators have been calculated but before any recursive descent parsing of the expressions.

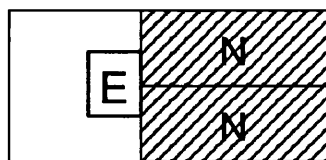


Figure 6.10: Chang's [1970] division rule for exponentiation

Exponentiation is best parsed bottom-up;  $2^{\frac{1}{2}}$  can be understood correctly only after the fraction has been parsed and its baseline determined. The recursive descent is in order of mathematical priority, but the syntactic pre-parsing is in syntactic order. Thus divisions have a higher syntactic priority than exponentiation, and are pre-parsed first. The pre-parser is, in a sense, a bottom-up syntactic parser.

Exponent pre-parsing is done by grouping symbols by baseline into a tree structure. Each exponentiation level is then recursively parsed further. Figure 6.11 shows the exponentiation levels of the expression  $2^{3^4+5}6^7$ . This expression is parsed from left to right. Symbols are grouped along the same baseline: if the baseline of the symbols is higher than a threshold (75% of the current symbol height) then another exponent parse is recursively started at this point with a new higher starting baseline. When the baseline drops the parsing exits.

Figure 6.11 shows the final tree generated from parsing the expression. This is generated left to right, as the baseline moves up, (after the 2, 3 and 6) a new sub-tree is started, after the baseline drops (after the 4 and 5) the recursion moves back down the structure tree.

This tree provides the final expressions used in the recursive descent parser. Exponent operators are added into the expression where complete sub-expressions provide the base for the exponent, and are ignored when the base of the exponent is illegal, for example  $+^2$  is parsed as  $+2$ .

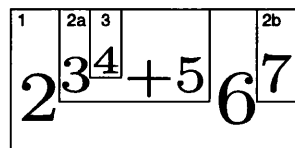


Figure 6.11: Exponentiation ordering

#### 6.4.7 Missing components

When parts of an expression are not provided but which are syntactically required, these are replaced with placeholders. Examples of missing components would be a missing numerator in a fraction, digits before or after a decimal point, or a matching bracket. These are required to make the



mathematical expression valid and placeholders are added automatically when there is no user input. The calculation described in the next section then adjusts the values of these placeholders to ensure the expression is mathematically correct.

## 6.5 Calculation

Lots of work and research has gone into providing computers with more and more powerful mathematical capabilities, from algebraic manipulations of complex formulae to signal processing. However in contrast, very little research has been done in novel methods of providing calculation.

Harold Thimbleby [1996] outlines a new *declarative* design of a calculator. This design was an attempt to correct some of the mistakes he found in existing calculator design and the main premise of this design was to show the user a correct equation all the time. The primary design components of this system are:

1. to take equations from the user, not instructions to calculate
2. to display exactly what the user has entered
3. to permit the equation to be edited
4. to fill in any missing numbers or symbols
5. to correct all mistakes, and ensure the result is numerically correct
6. and to do so at all times not requiring any “terminators”

This design for a calculator uses a linear entry of expressions, similar to that of a text editor, the user can add and edit at any point in the calculation.

The central idea of showing the user a mathematically correct display all the time is that the calculator non-destructively completes the user’s work, simultaneously correcting or solving any arithmetic mistakes or omissions. By doing this the calculator ensures that everything the user sees is always numerically correct. Figure 6.12 shows a screenshot of this calculator; the user has entered  $9 \times c = 1.5$  and the calculator has filled in additions, in an outline font what is needed to make this mathematically correct.

A conventional calculator works out  $3 + 4$  when the user *instructs* it to by pressing the [=] after [3] [+] [4]. The design described requires the output of the calculator to be an equation, such that ‘ $4 + 5$ ’ and ‘ $3 \times 2 =$ ’ are strictly incomplete. The completions, ‘ $=9$ ’ and ‘ $6$ ’ that are needed for a correct equation are provided automatically by the calculator to complete the mathematical expression and are shown in a different colour. These completions, hopefully intrinsically, provide the answer the user wants. In fact the answer is available before the user even presses [=].

If the user enters an invalid expression such as  $7 = 3$ , the calculator corrects this by balancing the equation with a '+4' on the right-hand side.

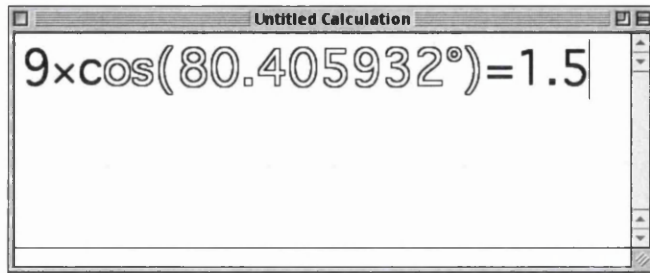


Figure 6.12: Harold Thimbleby's [1996] calculator

The completions ensure that at every point in a calculation the calculator shows a mathematically correct display. The initial 'blank' expression is not '0', as on an ordinary calculator, but ' $0=0$ '. A correct expression, such as ' $4 = 2 + 2$ ', requires no completion and is not adjusted in any way. An important part of this process is that the calculation of the completions is consistent, that is, the same calculation always has the same completion. A completion never depends on previous calculations nor on how the calculation has been edited; it depends only on the actual text of the incomplete expression. Thus for the user, the calculator's interaction is completely predictable, there is no hidden state that determines how the calculator works.

The incomplete expressions are corrected intelligently, often providing useful feedback or even answers before the user has finished entering the whole equation. The intelligent completion also allows a user to carry out "reversible" calculations, for example calculating the answers to equations like  $4 \times ? = 36$  and  $2^? = 100$ .

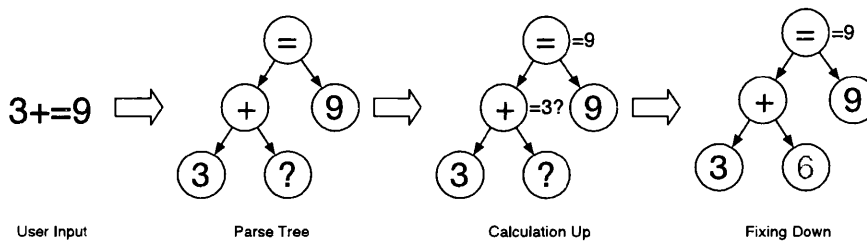
This *declarative* design for a calculator works well for a pen-based system and allows many powerful interaction possibilities. The primary benefit of this design is the ability of the calculator to provide immediate feedback as the user writes a mathematical expression. There is no need to wait for a complete expression before calculating the answer because the calculator handles incomplete equations smoothly and unobtrusively.

### 6.5.1 Implementation

The implementation of the calculator uses a novel method of performing multiple traversals of the final parse tree. Before the calculation takes place, the final parse tree is built with two considerations, firstly unknown placeholders are inserted where user input is missing, and secondly if the parse tree is missing an equality, it is added at the root level with an unknown placeholder on the right-hand side.

Once the parse tree has been built, it is traversed up to three times:

1. An upwards traversal from the leaves, calculates known values and guesses. After this pass each node in the tree has either a known or guessed value. Operators with unknown operands guess sensible and predictable values for the operands.
2. A downwards traversal computing unknowns that are not fixed. This starts from the root of the tree and using the mathematical inverse of each operator forces values down the tree into the unknown leafs.
3. A second downward traversal happens to ensure that the mathematical expression is balanced where unknown values have extra restrictions on their value, for example roots and factorials.

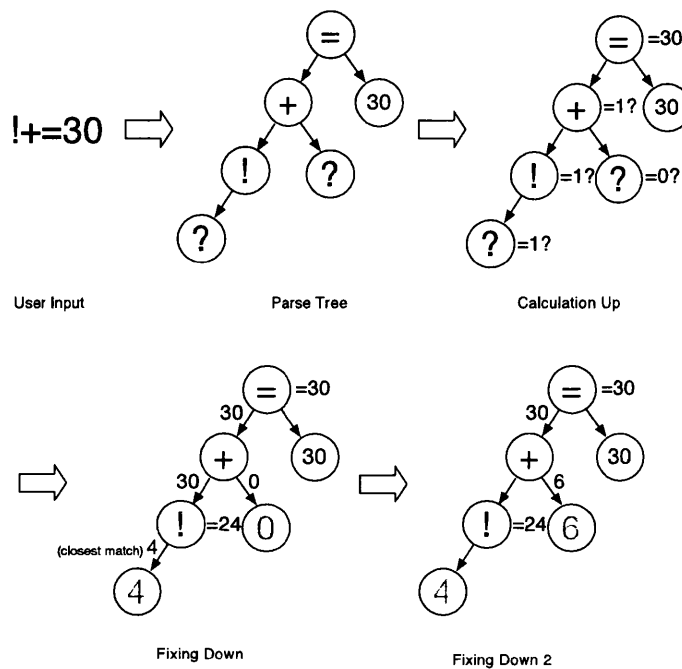


**Figure 6.13: Correcting a user's expression**

Figure 6.13 shows a simple example of this process for the unfinished expression '3+=9'. Once the parse tree has been constructed any gaps are filled in so that any missing leafs on the parse tree are replaced with computer-generated unknowns, whose value the computer will fill in when completing the tree. In this case the right-hand-side of the addition is filled in with an unknown. Then, in the calculation pass, the value of the expression is recursively calculated upwards. The value of each node is composed of a numerical value and a flag stating whether or not that value is fixed. An unfixed node's value is a guess and can be changed by future passes. Nodes that are not fixed include computer-generated unknown nodes and most nodes with an unfixed child.

An equality, the root node in this tree, chooses its value to be the value of its fixed child (if both children of an equality are fixed the left-hand-side is chosen for predictability). In Figure 6.13 the right-hand side of the equality is the only fixed side (the left-hand side is an unfixed 'guess' of 3), therefore the equality takes the value of the right-hand side. The second traversal pushes the value of the root node down the parse tree. This alters unfixed values as necessary to make the tree mathematically correct. In Figure 6.13 the value of the addition is corrected to be 9, which in turn changes the unfixed left-hand-side of the addition to 6. The parse tree is now complete and mathematically correct,  $3 + 6 = 9$ , and there is no need for a third traversal.

Figure 6.14 shows a more complex example of completing  $!+ = 30$ , which



**Figure 6.14: Correcting a more complex expression**

requires three traversals. The initial creation of the parse tree and the calculation of the values in the tree happens as normal. The value 30 is chosen as the only fixed child for the value of the equality. The  $+$  operator attempts to set its unknown operands to 30 and 0 (the default behaviour for the  $+$  operator with two unknown operands), this attempts to set the value of the factorial to be corrected to 30. This is an impossible value for a factorial ( $4! = 24$  and  $5! = 120$ ), so in this case the inverse factorial function chooses the nearest value it can get to, in this case  $4!$ .

This leaves the operands of the  $+$  not actually summing to 30, so a second correction of the right-hand side of the  $+$  is attempted with the left-hand side fixed at 24, this pushes the value 6 down into the right operand of the addition. This third traversal finally leaves the tree correct mathematically with the solution  $4! + 6 = 30$ .

Parsing of a complete expression,  $7 = 5$  that contains no unknowns, is shown in Figure 6.15. The calculation of the value leaves an equality mismatch: the left-hand side of the equality is 7, the right side is 5 and both sides are fixed. This situation is corrected by choosing the left-hand value of the equality and adding in an extra operator to the root of the right-hand side of the equality, this is either an addition or a subtraction depending on which side is greater. Always choosing the left-hand side value means that the correction of an inequality is always and consistently positioned on the far right of the mathematical expression. The third and last traversal then

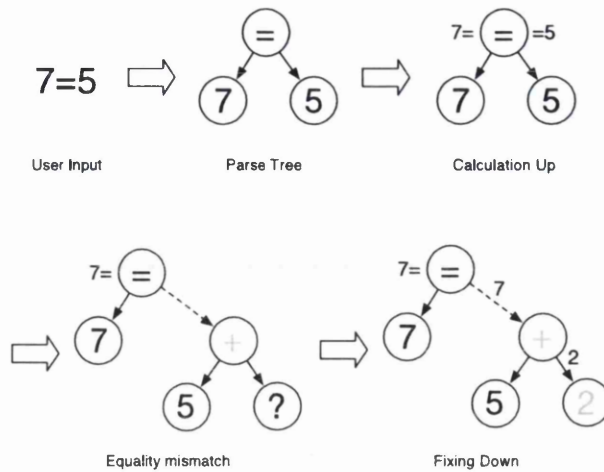


Figure 6.15: Correcting a complete expression

corrects this new parse tree, correcting the addition to have a value of 7, resulting in the solution of  $7 = 5 + 2$ .

This multiple pass correction works well for correcting most expressions and can sensibly be used to correct more complicated equations like  $2^! = 100$  which gets corrected to  $2^{3!} = 100 - 36$  and  $\surd = -10$  which gets corrected to  $\frac{\sqrt{1}}{-0.1} = -10$ .

The calculator also handles complex arithmetic easily. It is not only able to calculate  $e^{i\pi}$  and  $\sqrt{-1}$  but also to correct complex expressions like  $2^! = -64$  correctly using complex arithmetic to  $2^{(6+4.532i)} = -64$ .

## 6.6 User Interface

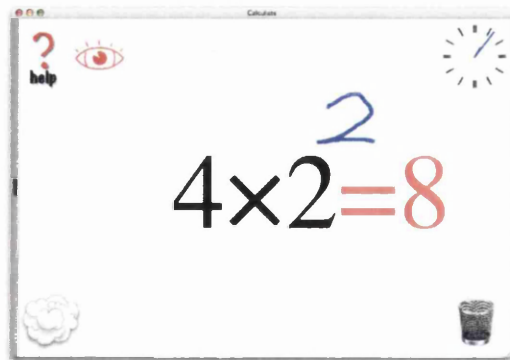


Figure 6.16: The calculator's user interface

The main portion of the user interface is the central white canvas, which

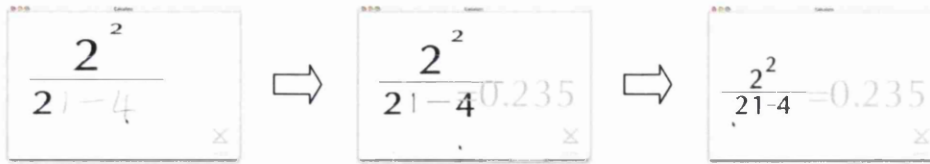


Figure 6.17: An expression being morphed

shows the current equation and handwriting, Figure 6.16 shows the equation  $4 \times 2$  being modified to  $4 \times 2^2$ . The interaction of the calculator's user interface is described more fully in Chapter 2, this section provides an overview of the implementation and how it interacts with the underlying processes.

In terms of implementation, the user interface provides a rendering of the current calculation and handles user input by passing the user's handwriting to the symbol recogniser.

### 6.6.1 Interaction

The user interface provides continuous *projected* feedback of the state of the calculator and it provides *continuity* linking between the user's hand-written input and the calculator's output by morphing between the two. How the different stages in the calculation process are handled by the user interface are listed below. This process is also shown in Figure 6.17.

- A. Symbol Recognition — The user interface replaces the user's hand-written symbols with typeset characters as the user writes them. A small delay allows the user to finish writing composite symbols. This approach provides immediate *projected* feedback about the recognition process and does not leave any doubt as to the symbols recognised.
- B. Expression Recognition — After a small delay once the symbols have been recognised the calculator recognises the equation. The user interface morphs the now typeset characters from where the user wrote them into a neat typeset equation. The morphing provides a smooth *continuous* linking between what the user wrote and the final equation. Using a typeset equation provides a clear representation of what equation the calculator is computing.
- C. Calculation — The 'blanks' in the calculation are filled in as the equation is morphed to make the equation mathematically correct. These show the answers in-place and provides immediate feedback of what the calculator has *declaratively* computed.

### 6.6.2 Pen-based interaction

The majority of potential users for the calculator will already be competent with writing using a pen on paper and also writing mathematics down on paper. The primary advantage of a pen-based system is the similarity or perceived affordance [Norman, 1988] of the pen-based interface with that of pen and paper.

The advantage of an affordance with paper is especially powerful with mathematical expressions, because a lot of sums and mathematical work is still done on paper with a pen or pencil. A pen-based system that works in the same fashion as pen and paper, means that anyone familiar with writing mathematical notation should be able to enter expressions with little or no training. There need be no restrictions on how an equation is written. Ideally if a mathematician writes an equation neatly in exactly the same way as they would on paper, it will be recognised.

Another advantage of pen-based user interfaces is there is no need for any other interface. Pens are capable of replicating the complete functionality of both the keyboard and the mouse. There is no need switch between two input devices, as the pen can be used for both. This and the ability of pens to be used on small screens are some of the prime reasons that are driving research in pen-based mobile devices.

The actual input data to the calculator software is mouse movement, pen or finger based input is solely dependent on the user input hardware. While the calculator is usable with a mouse, writing smooth symbols with it is very hard.

### 6.6.3 Expressions and ink editing

The system allows users to enter expressions as they would on paper, without any unnatural restrictions in a *WYSIWYE* way. For example, the user is not forced to enter the expression in a linear fashion as some expression recognition methods require [Littin, 1993].

Although there are small timing constraints to allow multiple stroke symbols to be written, these are rarely intrusive, so the user should not have to alter their way of writing by much, if at all, to use the calculator.

Edits can be made to an expression by adding new symbols, deleting parts of the expression or moving parts of the expression from one location to another. After each edit the current parsed mathematical expression tree is thrown away and completely recalculated from scratch using the new symbols and locations. This means that an edit, insertion, move or deletion is performed on the “ink” not on the expression tree. From the user’s point of view this is what they see and expect.

#### 6.6.4 Selection

In the user interface, the user is required to select symbols before they can be deleted or dragged elsewhere. A selection is created by drawing round the symbols the user wants selected. This gesture is recognised automatically from the context and does not make use of explicit user mode switching for gestures which limits the user's interaction [Li et al., 2005].

All other drawing on the calculator ends up as ink as part of a new symbol. The only gesture the calculator supports is the circle-drag-drop gesture. This gesture is initiated by encircling symbols in a loop, where a loop is one which is closed or one that comes within a small distance of being closed.

The creation of a selection is highlighted to the user by several visual and audible notifications: a “whoop” sound is played, the inner part of the loop turns light blue, and the symbols contained in the loop turning bright blue.

Once symbols have been encircled, the selection stays visible. When a selection is visible on the screen the calculator does not accept new drawing as normal. Drawing on the screen does not create new strokes but removes the selection. By dragging from any location inside the selection, the selected symbols can be moved anywhere within the mathematical expression. While the user is dragging, the user interface draws an arrow from the selection to the current drop point.

Dragging a selection to the bin provides a similar user interface metaphor for deletion to deleting files in Windows Explorer or the Mac OS X Finder. This combination of selection and deletion removes the need for an additional deletion gesture. When the deletion happens an animated smoke cloud is placed over the symbols and a “poof” sound is played. The symbols are removed and the expression is re-parsed without the deleted symbols.

The single encircling selection gesture and the ability to dragging the selection within the equation, to the bin or even to the dock supports all editing possibilities. Only having a single gesture makes the user interface simpler and easier to learn or use.

#### 6.6.5 Drag and drop

Dragging symbols utilises this concept of “ink editing” to provide dragging for ink rather than syntax or structure. Dragging is implemented by moving the selected symbols to the end point of the drag and drop and shrinking the symbols down to tiny proportions so they take up no room but still retain their relative positions. Once the symbols have been reinserted and moved, the expression is re-parsed and the symbols morphed into their new correct locations for the new expression.

A side-effect of the way this drag and drop process interacts with the ex-



pression parser is that it is impossible to drag and drop symbols on-top of the existing expression so that the existing expression is contained inside the dragged symbols. This is because the dragged symbols are reinserted as tiny symbols at the drop point so you can drop them into an existing expression but the symbols are too small to be recognised as containing any symbols they are dropped on-top of. For example, it is not possible to drag a square root symbol over the top of a number, but it is possible to drag the number underneath the square root.

Performing operations on the “ink” of an equation also means that edits can be made that do not make sense to perform on the expression tree. Examples would be dragging non-contiguous symbols or syntactically meaningless groups of symbols. Figure 5.6 in the previous chapter shows several drag and drop ink-edits that make no sense when thought of syntactically.

### 6.6.6 Undo

The user interface provides an undo ability, shown in Figure 6.18, in the form of a ‘clock’, positioned in the top-right hand corner of the screen. The clock hands show the current ‘time’ of the equation. Every time a user writes something or edits the equation, as the equation morphs the clock moves on a “quarter of a hour.”

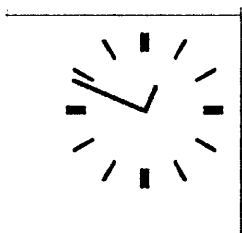


Figure 6.18: The undo ‘clock’

A user can grab the clock hands and rewind the hands by moving in anti-clockwise circle around the clock. As the clock is rewound, the equation displayed is the equation that was shown in the past when the clock was previously at that time. The undo clock allows users to undo mistakes and to rewind and scrub through the past to get an overview of how the current (or any previous) equation was achieved.

The undo system is implemented as a keyframe animation of everything the calculator has shown. The undo system records everything that happens as an animation except for the pen movement of user input. So when rewinding the calculator will display symbols morphing from location to location but no handwriting. No semantic information is kept only the sizes and positions of symbols.

Each symbol keeps a record of its creation time, all the keyframes of position and size that it was morphed through during editing and a deletion time if it was removed. Each time the equation is re-parsed and the symbols move, the keyframes for all the symbols are recorded for the current position. Keyframes are also recorded for the smoke deletion animation and for the computer answers fading in and out.

As time is wound forward or back, the symbols are morphed between these keyframes without any additional semantic information. Once the user has finished and begins to write on-top of the mathematics shown, the expression is re-parsed from what the user sees. By beginning to write in the past, the future equations are lost and the clock cannot be wound forward again to retrieve them.

The parsing is flexible enough that any state between the original input and the typeset result is always recognised as the same expression, this means that stopping the clock at a point halfway through a morph does not cause any problems.

### 6.6.7 The dock

The dock is provided so that a user can work on multiple equations at the same time. The dock sits at the left side of the screen and can be shown or hidden by the user by dragging on the dock handle. Expressions stored in the dock are shown as small, scaled-down, versions of what they would look like when edited.

Each expression stored in the dock lives in its own world and encapsulates its own data and undo history, similar to multiple open text documents in an text editor.

The dock is initialised from a text file which stores the initial expressions in a layout (ink) based syntax, and can be customised to a user's needs. The dock text file and its syntax is available in Appendix G. The dock is not saved after each session, but there is no technical reason why it need not be. Saving the dock could provide a useful memory store over multiple sessions of using the calculator. However, the calculator is currently not designed for multiple sessions. These choices were made to make it robust in multiuser situations such as exhibitions.

New calculations can be created in the dock by dragging expressions or portions of expressions from the current expression into the dock. Expressions in the dock can be rearranged by drag and drop or deleted by dragging to the bin. Selecting an expression stored in the dock is done by clicking on the expression. When an expression is selected the main expression being edited changes to the one stored in the dock.

It also possible to drag expressions stored in the dock out into the cur-

rent expression being edited. These dragged dock expressions appear as boxed expressions that are immutable, but which can be dragged around and deleted like all symbols.

## 6.7 Summary

The mathematical recognition the calculator is based on uses simple model matching of symbols combined with a hybrid structural specification scheme and recursive descent expression recognition. These provided decent recognition for the subset of mathematics the calculator handles.

The answers to the user's input are provided sensibly, using a *declarative* approach, regardless of how incomplete the user input is. The user interface provides flexibility to the user and allows additions and edits to the expression without the constraint of the mathematical structure. This “ink editing” underlies most of the implementation and computational approach to editing mathematical expressions.

The combination of flexible editing and sensible interpretation of incomplete input, combine to provide a compelling user experience even when the mathematical recognition is completely wrong. Indeed, quick and easy recovery from errors, whether human or computer, makes the calculator fun to use.

## Chapter 7

# Evaluation

This chapter begins with a brief discussion of evaluation in HCI as used in this thesis. Previous research both providing a comparative evaluation of a 2D version of the calculator and evaluation of pen-based mathematical interfaces in general are described.

The calculator and the principles described in this thesis are evaluated in a variety of different ways using different techniques:

- Comparative task-based and quantitative user testing of the pen-based calculator using exam questions.
- Feedback from unguided user interaction from a large range and number of people at the Royal Society exhibition.
- Comparative evaluation with *xThink*, a pen-based calculator that was designed without the using the *flow* principles.
- Heuristic evaluation using Green's [1989] cognitive dimensions.

Evaluating the calculator with different methods yields different insights and useful information about the design and principles. Each different technique lends its own complementary support to the design of the calculator and the principles underlying it. The difference the underlying principles make is highlighted by the comparative evaluation with *xThink* which provides a great comparison because although it is superficially similar to the calculator it lacks most of the *flow* principles. The culmination of these different strands of evaluation is then tied up at the end of this chapter.

### 7.1 Evaluation in HCI

*Readings in HCI* [Baecker et al., 1995] concludes “Given [design’s] complexity, and its mystery, how are we to proceed? The answer is implicit in the process of iterative design — *evaluation*.” Many main HCI undergraduate

text books [Preece et al., 1994, Dix et al., 1997, Shneiderman, 1997] similarly cite evaluation as a major part of HCI design.

Evaluation is important. Many techniques, with different approaches, have been developed to provide evaluation of computer systems and user interfaces. Examples include: User testing [Nielsen and Mack, 1994] which is probably the most widely used technique. Typical users are brought into a lab and use a prototype. Various data collection methods like observation, thinking aloud, tasks and questionnaires are then used to elicit data. Participatory design techniques [Schuler and Namioka, 1993] attempt to identify user design requirements. Analytic techniques like, GOMS [Card et al., 1983], cognitive walkthrough [Wharton et al., 1992] and heuristic evaluation [Nielsen and Mack, 1994] offer an evaluation without users or prototype. More recently, laboratory studies have been questioned, and there is a greater emphasis on field and ecological methods.

In *Trouble with Computers*, Landauer [1995] states the case that insufficient evaluation of computer systems and user interfaces with respect to their usefulness and usability is a major problem and part of the “productivity puzzle”. Although evaluation only address the usability half of this puzzle, Greenberg and Buxton [2008] point out that usefulness is much more difficult to evaluate.

### 7.1.1 Creating or iterating

While refining a design through user evaluation and iterative design is a successful and productive process, these methods are potentially less successful when it comes to *creating* a novel user interface design [Buxton, 2007].

When the HCI was a young field much of the literature was principle-orientated. Books like Tognazzini’s popular and much-cited *Tog on Interface* [Tognazzini, 1991], published as late as 1991, are primarily guidelines and principles. Here are two examples of different principles from Tog’s book:

“Do not attempt a 3D look in one-bit graphics.”

“Make the response time snappy. The more rapid-fire and more closely coupled the dialog, the more the user will feel and be in control.”

A simple example of this trend towards evaluation is that, while evaluation is a strong part of both the 1987 version of *Readings in HCI* [Baecker and Buxton, 1987] and its 1995 second edition [Baecker et al., 1995], the second edition spends much less time focusing on principles and more time on evaluation. The focus in HCI literature has increasingly been to focus on evaluation: quantitative evaluation (about 70% of publications) and qualitative evaluation (about 25%) [Barkhuus and Rode, 2007]. Arguably, this has been to the detriment of other forms of verification such as analyti-

cal evaluation and principled or principle-led design. Barkhuus and Rode [2007] suggest that this domination of HCI evaluation by a few methods undermines novel and ground-breaking research that does not fit into the “correct” mould.

Perhaps the HCI field was initially concerned with creating new user interfaces, and as designs and user interfaces have proliferated, much of HCI has rightly become about refinement instead of the initial focus on creation. The focus has changed from creating new user interfaces to iterating and improving existing interfaces.

## 7.2 Evaluation in this thesis

It is not the aim of this thesis to evaluate the calculator, or Lineform described in Part II, with methods that measure error rates or task completion times. The contribution of this thesis is primarily the principles and ideas of user interface design that are incorporated in these applications, not an incremental and measured improvement of solving a calculation, or drawing a picture. The appropriate methodology for the evaluation of these principles is reasoning and argument, which this thesis provides effectively. In many ways the principles could be valuable but a conventional evaluation of them turn out negative, for example when evaluating the principles incorporated in a poorly implemented prototype. The purpose is not to dismiss evaluation as a useful and important tool but to position the main contribution of this thesis as primarily situated in the design and principle space.

This thesis does not stress the traditional evaluation of the calculator or Lineform and their respective design principles. These are primarily design and conceptual innovations, and as Greenberg and Buxton [2008] suggest, traditional evaluation stresses measurable contribution at the expense of design and engineering innovations. While this thesis does not avoid evaluation it attempts to escape the “tyranny of evaluation” [Lieberman, 2003] by focusing on the principles and design innovation.

Despite the lack of extensive formal evaluation, the designs in this thesis have been tested and used by thousands of users. The majority of evaluation in HCI is formative iterative evaluation. Just as the feedback from users of successive versions of Lineform has helped guide its design, iterative evaluation is part of a process of improving a design.

Both designs, of the calculator and Lineform, are also comparatively evaluated, by analytical comparison with similar user interfaces. Formal comparative user testing is neither needed nor productive, given the nature of the user interface designs. Section 7.7 compares the calculator with a superficially similar user interface and shows how the principled design of the calculator avoids many design problems.

### 7.3 Declarative calculators

The mathematical engine the calculator uses is based on the declarative calculator first described by Harold Thimbleby [1986, 1996]. Harold Thimbleby's calculator used a simpler "traditional" one-dimensional input like a text editor but provided partial expression handling and completion that is similar to the calculator described here.

Cairns et al. [2004] evaluated this calculator comparing it with a software simulation of the Casio HS-8V. The Casio HS-8V is a standard four function calculator, which was simulated to provide control over the implementation and to remove the difference between physical and computer-mediated interaction.

Twelve subjects took part, each using only one of the declarative calculator or the HS-8V. The subjects were then asked to answer five GCSE mathematics exam questions (i.e., exam questions for 16 year olds), chosen to avoid the strengths or weaknesses of either interface.

The result of this study was that the declarative calculator took on average a third longer to use. They did see reduced error rates with the new calculator, and concluded that user familiarity with traditional calculator user interfaces and that the new user interface was so different that it left room to be optimistic about the new calculator's performance.

### 7.4 Pen-based mathematics

Studies have found pen-based user interfaces to be slower than typing [Brown, 1988], although compared with soft-keyboards on small screens, handwriting may have the advantage [Lewis, 1999].

Nevertheless there are several reasons why mathematics may benefit from pen-based user interfaces. Mathematics makes use of higher dimensional layouts, for example exponentiation, that are directly accessible from a pen-based user interface. Template based user interfaces do provide higher dimensional representations but these have to be constructed from templates in a top-down manner. Published comparisons with handwriting have focused on paragraphs of English text, but mathematics, in comparison, is more structured and contains many symbols like  $\sqrt{\quad}$  and  $\sum$  that are not directly accessible from the standard keyboard.

An evaluation of pen and speech input for mathematics, [Anthony et al., 2005], compared the entry of mathematical equations of varying complexity using the keyboard and mouse with Microsoft Equation Editor, pen-based handwriting, speech, and handwriting plus speech. Having empirically tested 48 participants the conclusion was that handwriting was significantly faster, less error prone and more preferred than using the keyboard and



**Figure 7.1: Wacom Graphire2 tablet**

mouse with Microsoft Equation Editor. In addition, they found that the more complex an equation (e.g., longer and including special symbols) the more the keyboard entry slowed down, while handwriting did not see such a sharp decline. Speech was also found to be a good method but worse than handwriting.

## 7.5 Initial evaluation

The calculator described in this thesis combines aspects of both the declarative calculator's mathematics [Cairns et al., 2004] and the pen-based handwriting interaction which was evaluated by Anthony et al. [2005].

An evaluation of an early version of the new calculator was first published by Will Thimbleby [2004]. This initial evaluation was performed with an early prototype of the calculator and later studies in this thesis have been performed with a much more capable and complete program.

The initial prototype was primarily hindered both by poor handwriting recognition and an unfamiliar user input device that users had some problems with. Despite this, the results were favourable and provide interesting data.

The accuracy of the prototype's handwriting recognition was 81.1%. That is, on average one in five characters were miss-recognised. This significantly lowered the usability of the overall system, as users repeatedly had to correct the handwriting recognition.

The user interface was based on a Wacom Graphire2 tablet (shown in Figure 7.1). Graphics tablets have to be used by looking at the screen and drawing on the tablet with the pen, and are typically usually used by artists. The disparity between where the user is writing and where they are looking



- Practice calculations:
  - Calculate  $3 + 62$
  - Calculate  $7 \times 4$
  - Calculate  $\frac{4}{5}$
  - Calculate  $9 - 5$
  - Calculate  $3^2$
  - Calculate  $3^? = 64$
- Simple calculations:
  1. Calculate  $2 \times 3 + 4$
  2. Calculate  $\frac{2^2}{21-4}$
  3. Calculate  $9 - 2/3$
- Mathematical problems:
  4. What is the average of 21, 34 and 56?
  5. What multiple of 32 equals 50?
  6. What power of two is 28?

**Figure 7.2: Tasks used for the prototype evaluation**

is unusual and it can take a while before users are comfortable with this kind of interaction. None of the users in the usability study were familiar with using a tablet, and they predictably found using an artist's graphics tablet awkward.

### 7.5.1 User studies

A total of nine participants took part in the testing (2 female, 7 male undergraduate students). All the participants had used standard calculators at school, and were studying a wide range of subjects including mathematics and art history.

Before the test began, users were allowed to familiarise themselves with the pen and tablet. This involved suggesting that they try to write words, numbers and draw pictures with the pen and tablet interface. The observer then gave a short demonstration of the calculator, showing how an example sum would be entered. When the user announced that they were ready, the observer started the test by giving the user a list of tasks on a piece of paper (see Figure 7.2). Some of these tasks were based on old GCSE mathematics papers.

The tasks were split into six practice questions, three simple mathematical questions, and three worded mathematical problems.

The thinking aloud protocol was used [Lewis and Rieman, 1993]. Participants undertook the tasks in Figure 7.2 while the observer watched them and helped when they had problems. On conclusion of the test, the observer discussed any issues that arose during testing. These were supplemented by an anonymous questionnaire (shown in Appendix A). The tests were also recorded as a live video of the user's interaction and the interaction on screen was recorded using screen-capturing software.

The anonymous questionnaire was used in addition to the think aloud and discussion so that participants could freely express their thoughts about the system. Providing a discussion afterwards also allowed the observer to ask additional questions resulting from issues that arose during the testing.

After discussing the calculating system with the observer, users were asked to perform the same calculations again on either their own pocket calculator or a calculator provided for them (a Sharp EL-531GH DAL). These tests were also recorded.

Sharp's DAL technology means:

Until the introduction of SHARP's D.A.L., keying in equations had been a complicated process making scientific calculators difficult to use. Introduced in 1992 and an industry-first, SHARP's D.A.L. allows symbols and numbers of an equation to be entered as they are written. Instead of wasting energy on difficult calculator operations, users are free to concentrate on mathematical concepts.

— Sharp D.A.L. marketing<sup>1</sup>

For example, the Sharp web site gives  $10 + 2 \sin 30$  being keyed as  $\boxed{1} \boxed{0} \boxed{+} \boxed{2} \boxed{\times} \boxed{\sin} \boxed{3} \boxed{0} \boxed{=}$ . This would contrast to calculators where sin is a postfix operator, which confuses users as  $10 + 2 \times 30 \sin$  would almost certainly find  $\sin 60$ , etc.

### 7.5.2 Results

Upon completion of the test, the video recordings were reviewed and information on error rates and time on task was extracted and logged.

In general, participants found the interface and concepts of the new calculator easy to learn and use, despite many users struggling with handwriting recognition problems.

When asked to rate the system in terms of ease of use compared to other systems they had used, on a scale of 0 (worse) to 5 (better), all the answers were above 3 and had an average of 4.1.

<sup>1</sup><http://sharp-world.com/contents/calculator/features/standard/dal/index.html> (viewed Feb 27, 2010)

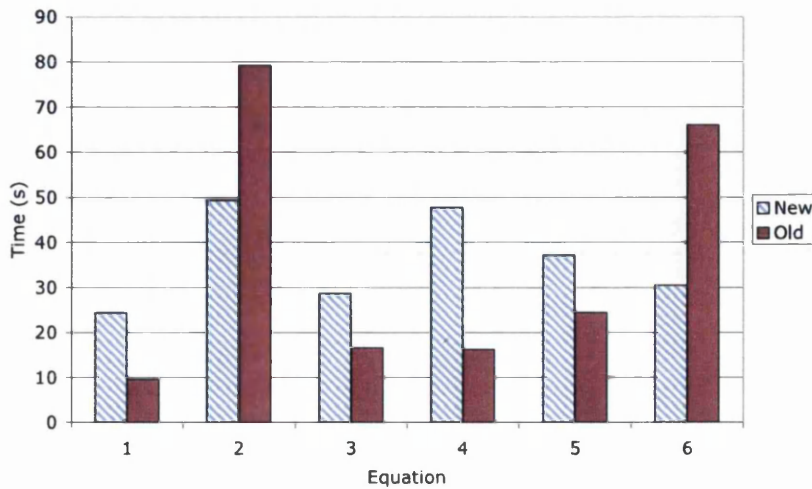


Figure 7.3: Average time for users to complete tasks

The results presented here were extracted from the video tapes of the user tests and from the questionnaires. Comprehensive results from the anonymous questionnaire, and some of the discussions, are in Appendix B.

### 7.5.3 Time on task

Figure 7.3 shows the average time for the users to complete each of the tasks in Figure 7.2. This figure shows a comparison in seconds of the average time for the users to complete each task using this system and using their own calculator. The last two tasks were left incomplete by several users when using a normal calculator, so these results are averages of those users that completed the tasks successfully.

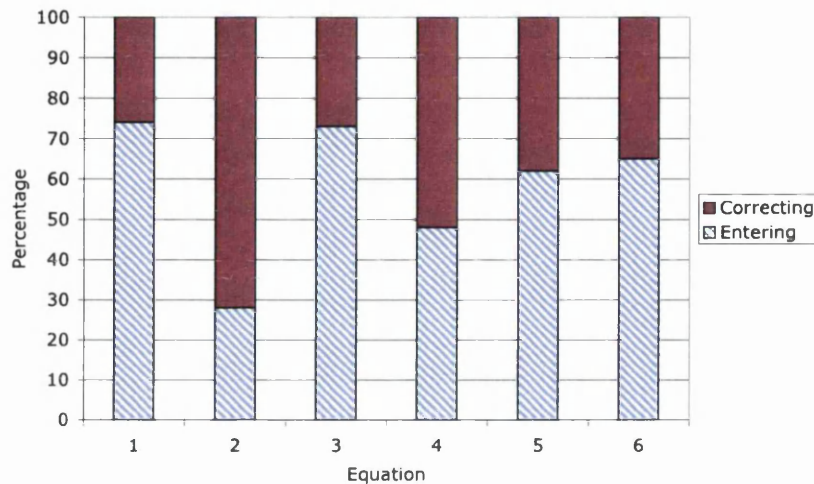
The last two tasks that users struggled to complete were:

5. What multiple of 32 equals 50?
6. What power of two is 28?

Both of these problems require some rearranging to find the result using an ordinary calculator. Only three users knew enough mathematics to find the answer to task six, using  $\frac{\log 28}{\log 2}$  to find the power of 2 equal to 28. However, all of the users successfully managed to arrive at the solutions using the new system.

For the simpler sums, like  $9 \times 2/3$ , the handheld calculator was much faster than the new system. This was expected. All users were familiar with their own handheld calculators. Handwriting and handwriting recognition also slows down the new calculator. However two of the tasks were actually faster on the new system.

Task 2:  $\frac{2^2}{21-4}$  was almost certainly performed faster because users could enter it “as they saw it,” as one participant described it, rather than having to search for buttons and to think about brackets on a handheld calculator. Secondly, task 6: “What power of two is 28?” was performed faster; again users could write the problem with minimal rearrangement or use of equations.



**Figure 7.4: Average percentage time entering or correcting tasks**

Figure 7.4 shows the average percentage time spent by users entering a formula and correcting mistakes when using the new system. For each task the percentage of time spent correcting is large, on average 42% of the time users were using the calculator was spent correcting errors. Several users had trouble entering task two, often because of bad segmentation errors (an implementation problem), these users spent a long time trying to correct the expression, often restarting from scratch when a symbol was miss-recognised. The timing results here reflect more on, or at the least are obscured by, the accuracy of the handwriting recognition of the prototype and the interaction problems of users unaccustomed to an artists tablet input device.

#### 7.5.4 Ease of use

In general, all of the users expressed their enjoyment of using the system at the end of the tests. After the quick demonstration of a simple mathematical expression, not a single user asked a question regarding the use of the system, excluding problems with the handwriting recognition. Every user found that the system worked as they expected it to.

Feedback in Appendix B from the questionnaires reflects this. Comments were made by all participants praising the simple and intuitive user interface.

Users liked the lack of buttons and that they did not need to think about extra things like brackets.

However, several users also expressed some frustration with the symbol recognition, especially when the system repeatedly miss-recognised certain symbols. Most users also commented that they found the Wacom pen and tablet awkward to use. This was primarily due to the fact that writing happens in a different place to the screen. This is in contrast to later developments which provided much more accurate symbol recognition and touch based user interface of tablets or SMARTboards.

Every user liked the morphing, and they particularly liked the fact that they could instantly see what the calculator was calculating.



**delete**

**Figure 7.5: Delete gesture from early prototype**

The initial prototype tested here used only one additional feature to handwriting recognition. This was a delete gesture, similar to a joined up X, as shown in Figure 7.5. This gesture has since been replaced with dragging to the trash.

Some users found that the delete gesture was difficult to use over a large area and several users suggested the addition of a `Clear` button, which has now been added and is in the current version of the calculator.

### 7.5.5 Accuracy

The large amount of time spent correcting errors suggests that better handwriting and expression recognition could dramatically reduce the time on task. A large part of the time taken to complete the tasks with the prototype calculator was taken up with specifically recovering from symbol recognition errors. There is a significant correlation ( $r = 0.78$   $\rho < 0.05$ ) between the time spent correcting symbol recognition errors and the time taken, which suggests that improving the recognition will improve the time spent entering mathematics. The poor average symbol recognition accuracy percentage of 81.1%, was a significant factor in the input error rate and did not aid the overall usability of the system.

However, when calculating mathematics, input accuracy is not the most

important consideration as these errors can be corrected; the accuracy of the output is often far more important.

With the new calculator *no* user got the wrong answer for any question. That is, although there were a high number of errors in the input, these were all intermediate errors, errors that were noticed and corrected before the user finished. Errors that are unnoticed or uncorrected are a far bigger problem. Users never arrived at an incorrect answer with the new calculator, but when using a handheld calculator they made several simple mistakes that went unnoticed, resulting in a final value that the user thought was correct but was in fact wrong.

Crucially, by displaying the computed mathematics in an easily understandable two-dimensional format, the calculator provides the feedback necessary for the user to understand what is being computed. Users knew if and when their calculations were wrong and when they had to be corrected. This is an aspect of WYSIWYE, by making the user interface predictable and visible the calculator provides enough information for the user to show exactly what expression resulted in the answer and for the result of any action to be completely predictable.

Handheld calculators, on the other hand, usually do not provide this feedback. Several users got some of the answers wrong and did not realise that they were wrong until prompted. Some of the users even got some of the simpler sums wrong (like  $9 - 2/3$ ) without noticing their mistakes. This inaccuracy in the expected *output* is far more concerning than poor *input* accuracy, especially when most users trust calculator answers implicitly over their own judgement.

### 7.5.6 Summary

Users found the new calculator more intuitive and easier to use than traditional calculators. The new system was also faster in some cases and allowed users to complete problems they could not otherwise complete.

The new calculator was faster for some mathematical problems even though users were unfamiliar with both the system and the graphics tablet input device.

Of the two tasks that were faster, task 6 is an unfair time-on-task comparison: it was specifically added as a task to see if the new calculator enabled users to compute answers that they were unable to with a standard calculator. In the study, six users were able to complete the task with the new calculator that they failed to do on their own calculator. Thus the new calculator enables users to perform mathematics that they could not do before, which is enabled by its *declarative interaction*.

Typesetting and feedback through morphing successfully allowed the user

to understand what the calculator was doing. Importantly, users never arrived at an incorrect answer with the new calculator, however long it took, compared to several simple mistakes that went unnoticed using conventional handheld calculators. The immediate feedback of *projection* and the *continuity* of the user interface were key part of the user's comprehension of what was happening.

The new calculator provides an improved system that users produce less errors with and therefore can place more trust in. With practice users should be able to use it faster, without having to recheck their formulae. It is clear that the two-dimensional typesetting, and morphing provide good feedback that communicates what is happening to the user very effectively. The WYSIWYE nature of the user interface was central to making the interaction and result from the calculator obvious in how the user got to the current result and how to change it.

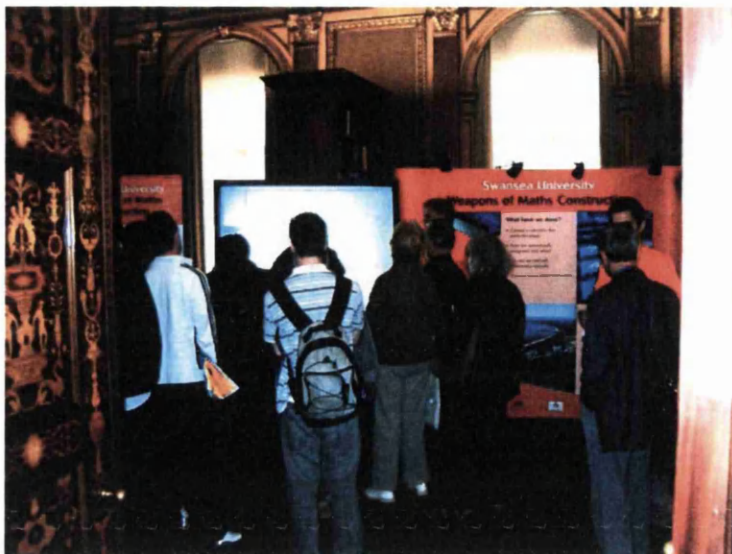
When viewed in the context of the unfamiliarity and symbol recognition problems these results are surprisingly good. The new calculator was faster for some problems, let users solve mathematics they could not have done otherwise and produced fewer errors. The concepts and ideas implemented in the new calculator were shown to be a success.

## 7.6 Royal Society evaluation

The calculator was exhibited at the *Royal Society's Summer Science Exhibition*, 2005. This exhibition is held to showcase top UK science annually at the London premises of the Royal Society, the UK's national academy of science. The calculator was one of 24 exhibits that were competitively selected from universities and companies throughout the UK. The purpose of each exhibit was to present science, engineering or technology, through visually engaging and interactive displays to the public.

The Royal Society's exhibit normally runs for 5 days, and with several evening events tailored to teachers, business leaders, politicians, royalty and others. Unfortunately the week included July 7, 2005, with its acts of terrorism across London, which had very unfortunate direct effects as well as a reduction in travel and visitor numbers to the Royal Society, so there were fewer respondents than hoped for. In particular, our planned video evaluations had to be cancelled.

The calculator presented in this thesis was one of these exhibits. Which was humorously titled with the bad pun: "Weapons of Maths Construction." The exhibit was large, covering over 6 by 3 meters of floor space in the Library of the Royal Society, which contained only our exhibit. This included aspects of historical calculators, from abacuses and slide rules to mechanical calculators of the 1960s. A large collection of modern calculators from various manufacturers including Sharp, Casio and HP were also available. To



**Figure 7.6: Royal Society Summer Science Exhibit**

aid their use, GCSE revision notes were available explaining how calculators should and should not be used, how to ensure mistakes were not made and how to work round their inconsistencies. A photograph of our crowded exhibit is shown in Figure 7.6.

The main part of the exhibit, visible in Figure 7.6, was taken up with a 6 foot diagonal interactive white board, a SMARTboard 2000i. This board is a rear projection touch screen, that users can interact with using their fingers (or indeed anything else). This board was running the calculator continuously for the four days of the exhibit. Visitors to the exhibit were encouraged to walk up and use the SMARTboard with their fingers or, a pen. Often one of the exhibitors would provide a short demonstration and encourage the visitors to have a go themselves and experiment. Throughout the exhibition the demonstrators remained on hand as visitors played with the calculator, answering any questions that arose and prompting the visitors to try different things out. The guidelines provided for exhibitors are included in Appendix C.

The exhibit was very popular, and due to overcrowding around our event, we sometimes had health and safety staff remove our evaluation desk and facilities to encourage people to move on.

### 7.6.1 Visitors

The visitors came from vastly different backgrounds, ages, cultures, education and occupation, from 6 year old primary school pupils to retired 80 year olds, from GCSEs to PhDs and professors, from students to teachers



to accountants to civil servants and builders

The whole enormous range and variety of visitors from shy teenage girls to Fellows of the Royal Society enjoyed using the calculator; seeing everybody and especially teenagers and school pupils getting involved and excited by the mathematics was very rewarding.

More than 4,000 people came through the doors of the Royal Society over the four days of the exhibition. From those visitors that viewed our exhibit, roughly a quarter used the calculator. After visitors had used the calculator the demonstrators encouraged them to fill in a feedback form. The feedback form used is included in Appendix D.

Visitors were further encouraged to fill in forms by providing prizes of an iPod Nano each day for the most interesting/useful feedback.

In total, 436 evaluation forms were filled in and handed in during the course of the exhibition.

The results from the feedback forms provide a reasonable estimate for the demographics of the visitors. The average age of the users who returned feedback forms was roughly 30, with half the users under 20. These were evenly split between male and female. The largest group of users was students who were either studying GCSEs or A-Levels. Teaching was the most common occupation after students but only by a small margin; there was a very broad range of other occupations represented.

### 7.6.2 Results

The raw anonymised data from the Royal Society's Summer Science Exhibition is available in Appendix E. The results from the feedback forms are summarised here.

Figure 7.7 shows the percentage of respondents that used different methods for calculating mathematics. The majority of the respondents used calculators for doing mathematics and most people used several different methods of doing mathematics. Just 15% of the respondents used calculators or spreadsheets as their sole method for doing mathematics, about the same as the number of people who only used paper or mental arithmetic.

Users were asked to rate the calculator on a scale from one to five for both enjoyment (disliked-it to loved-it) and helpfulness (unhelpful to very-helpful). The summary of the results from these two questions are shown in Figure 7.8 and Figure 7.9.

Over 90% of people returning feedback forms either liked it or loved it. Put another way, they liked or loved a calculator, something that users only usually tolerate. These results suggest that there is something about this new calculator that works better, and is more enjoyable.

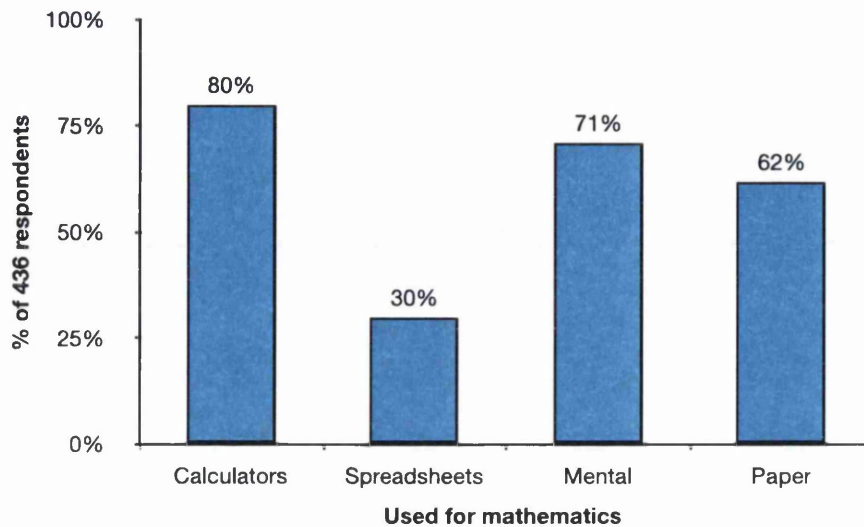


Figure 7.7: Royal Society evaluation: Usage

Users filling in the feedback forms were also asked three other specific questions on their use of calculators. The results are summarised below.

92% of respondents thought the new calculator was better

27% of respondents said they had previous problems when using calculators

34% of respondents said in retrospect they had problems doing mathematics

Part of the exhibit was about current handheld calculators and some of their problems [Thimbleby, 2000, 1996]. The last two statistics above were from two questions about problems with calculators were designed to find out if the exhibit had changed or informed visitors opinions about calculators. From a subjective point of view, it was successful, one Nuclear Engineer wrote in their feedback “[the] exhibition helped me realise how cumbersome (mentally) calculators/spreadsheets are.” However the statistical difference between the answers for the before/after questions is not significant (though we did not do a controlled before/after evaluation, as this would have been infeasible in the exhibition environment).

### 7.6.3 Quotes

Users were provided with several opportunities for providing feedback on the different aspects of the calculator. Some selected quotes are repeated here. Other quotes are available in Appendix E. Hopefully the quotes provide

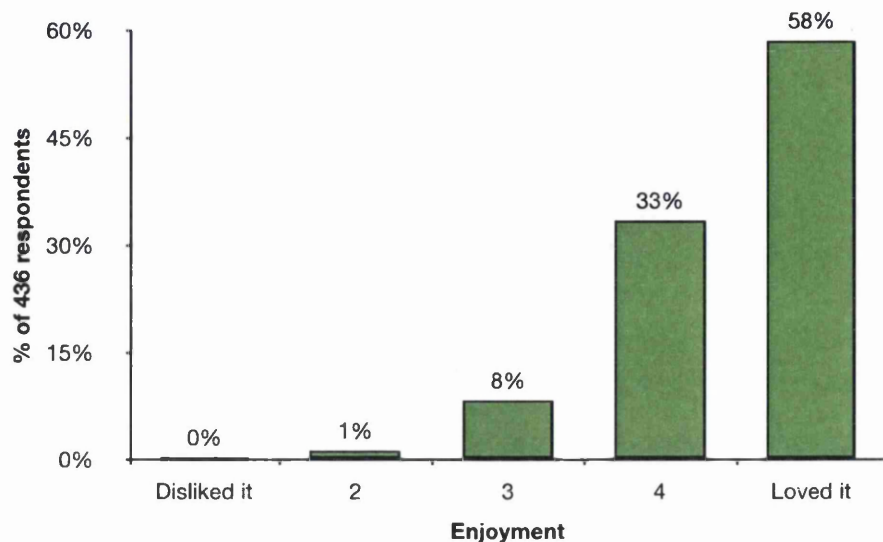


Figure 7.8: Royal Society evaluation: Enjoyment

another dimension to the results presented here, showing the enjoyment users got from the calculator.

Positive feedback came from students, who often do mathematics daily in classes and could quickly see the opportunities for using the calculator:

It stops you making mistakes. — Student

It is better because you can work out the equation very quickly, it takes less time to get the answer, you should provide it to all students at colleges and university. — GCSE Student

I loved it. — A-Level Student

The exhibition is amazing, I love it it gets me excited. Better than anything I've seen before. — A-Level Student

The most fun I've probably ever had doing Maths! A good mix of paper and calculator, more interactive and easier to learn. — A-Level Student

Calculators seem clumsy and hard to use — the new method is genius! — when can I buy one in the shops (If I had had one I would have done A level maths). — A-Level Student

It's great!! It's brilliant, better than pen and paper. — University Student

Lots of teachers and lecturers were also very positive about using the calculator both as a teaching aid in education and for use engaging students:

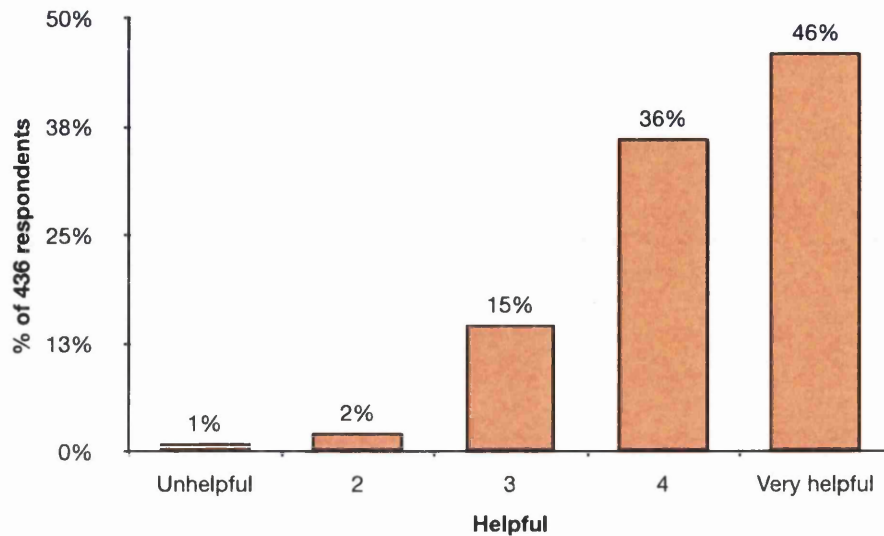


Figure 7.9: Royal Society evaluation: Helpful

It visualises the internal workings of abstract calculations, fun, as it is wonderful! Fun! Engaging and importantly visible! — University Professor

I believe this could be of real value for the education of groups of children in Maths, getting them to really interact with the equations in front of them. — RSC Higher education award winner

I used to teach with white boards and calculators. It was amazing. — 78yr old, retired teacher

Engagement, excitement, interactivity, seamless, more visually appealing and easier to use! — Teacher

It opens up endless mathematical explorations. — Teacher

It makes maths engaging and allows one to reason about the process, maths can come across as static but this enlivens things. — Senior Lecturer

Many other users from many different occupations and experiences also really enjoyed using the calculator. The whole range of users from artists to engineers all were mostly positive:

Great fun — of course it is better. — Musician

I've never seen anything that's brought a smile to my face while doing addition, but this has. For that reason alone, I want one! — Artist

GUI is fantastically intuitive. — Engineer

I didn't think there was an easier way till now, the possibilities are endless..  
— Grant Officer

Showed me some limitations of other calculators. — Actuary

Very few users had negative comments about the calculator and those that did were often commenting on only specific aspects of the calculator:

Calculator is faster, good for teaching but not on a daily basis. — IT Manager

Clunky but potentially brilliant, it leads to more questions which is great.  
— Research Manager

Probably better for school kids. — AS Student

Will kids think even less? — Science Communicator

I'm not sure whether it would be suitable for large quantities of data.  
— Accountant

There are limited functions available. — Physics Teacher

These are just a small selection of some of the more interesting comments from users. In Appendix E the comments from all the feedback forms are listed. Additional interesting feedback is highlighted there.

#### 7.6.4 Mumbai, India

As part of one of the popular exhibits of the Royal Society exhibition we were invited to also exhibit at Mumbai Institute of Technology's TechFest in 2008. An estimated 50,000 people came to the festival and thousands of students and parents visited our exhibit, which was constantly surrounded all day, everyday. Much of the feedback from this exhibit was their enjoyment of it and also highlighted the calculator's lack of support for higher mathematical functions, such as logarithms, integrals and trigonometric functions. This observation perhaps sheds more light on India's education system or, more specifically, on the educational attainment levels of the students visiting the TechFest exhibition than on the calculator itself.

#### 7.6.5 Summary

While the Royal Society exhibition was not a controlled user study, it did provide an opportunity to test the calculator with a huge range of people. The results from this exhibition might not be indicative of a longitudinal study, but they do provide a good representation of how the calculator functions in an environment similar to the exhibition and how easy people find the calculator to use without any prior experience.

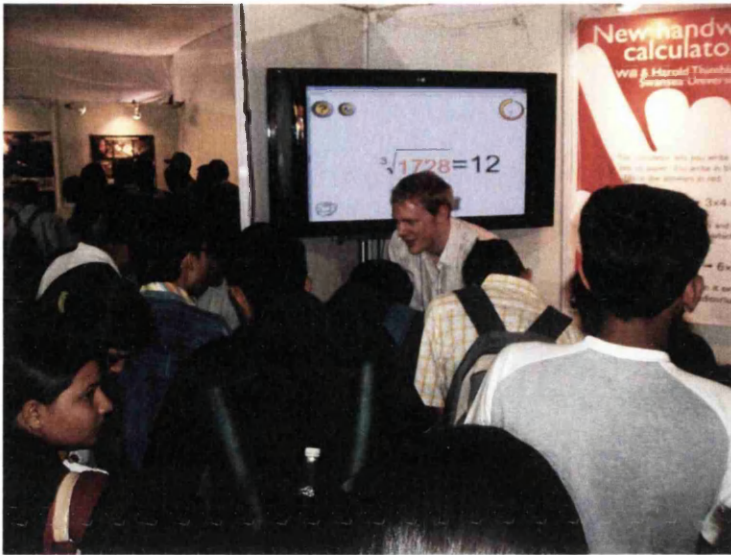


Figure 7.10: Demonstrating at TechFest, Mumbai

The feedback was overwhelmingly positive. A large majority of the visitors who used the calculator thought it was better, enjoyed using it, and thought the calculator was helpful.

Certainly whether the enjoyment of using the calculator is sustained over long use or whether users find the calculator easier to use in day-to-day situations are unanswered by this study. Indeed the form-factor of the user interface used in the exhibition (a 6 foot whiteboard) limits the situations where this sort of interaction could be representative of how users might use a calculator. However, it is reasonable to assume this kind of interaction would happen in a school classroom, where large screen interactive whiteboards are now common and the short group interaction is often typical of whiteboard use with a class.

Frequently during the exhibition demonstrators used the calculator as a teaching tool, to show users how it could be used, by explaining mathematical concepts like roots or fractions. In fact one of the comments from a student about a demonstrator was that he was “a nice ‘teacher’ he was very entertaining and whatever he was demonstrating was easy to enjoy”. Many teachers themselves were very enthusiastic about the calculator with comments like “a fantastic teaching aid”, “get it in schools” and “When can we have it?”

As a teaching aid, the new calculator on an interactive whiteboard obviously has many advantages over static whiteboards or handheld calculators. The success of the calculator in the exhibit setting suggests that there might be different uses for which the calculator might be more applicable. The use of the calculator in an exhibition or classroom is strongly supported by the

feedback from the Royal Society exhibit.

These positive results cannot be used to claim that the existence of the *flow* principles are the critical reason for the good reception of the calculator in the exhibit. However the *flow* principles are so integral to the entire user interface of the calculator, that the success of the calculator is by association also a success of the underlying principles.

## 7.7 A comparison with *xThink*

*xThink*<sup>2</sup> is a commercial pen-based calculator that recognises handwritten mathematical expressions and provides the answers to the calculations.

*xThink* provides a good comparison for the new calculator because, superficially, it appears to provide the same user interaction and capabilities as the calculator yet *xThink* does not implement any of the *flow* principles. *xThink* provides an interface that interacts in a very similar way to that of the calculator using a pen-based system and as a commercial system it potentially has the better mathematical recognition engine and mathematical functions. However, it is missing some of the principles described in this chapter, which means that the contrast in the experience of using *xThink* and the calculator can provide a useful focus on the difference the principles make. This section provides a heuristic evaluation of the *flow* principles by utilising a comparison of these two systems.

Some of the aspects of both *xThink* and *Mathematica*, as a comparison to the new calculator described in this thesis, are described more fully by Thimbleby and Thimbleby [2007]. Parts of this section are based on this paper.

A typical ‘page’ from *xThink* is shown in Figure 7.11. The advantage of this interface over other approaches, is the ease and simplicity of entering mathematics, however its interaction style retains some of the same problems that handheld calculators exhibit. There is no guarantee the ‘answers’ are in fact answers to the adjacent formulae, and furthermore *xThink* has introduced new handwriting recognition problems; that is, the formula evaluated may not *ever* be the one that was thought to have been written down.

*xThink* like the new calculator recognises user’s handwriting in the standard notational format and the computed answer is displayed adjacent to the hand-written sum.

---

<sup>2</sup>[www.xthink.com](http://www.xthink.com)



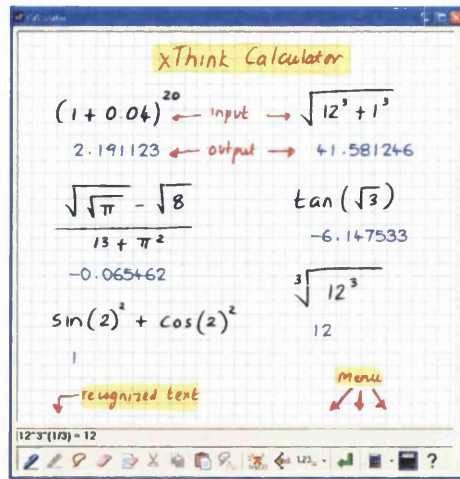


Figure 7.11: Example of *xThink*, showing natural handwriting notation combined with calculated output. Screenshot from *xThink*'s web site.

### 7.7.1 Specific differences

Both the calculator and *xThink*, from first glance, appear to do the same things. In fact *xThink* seems to be more powerful as it can handle annotation, multiple sums, and more complex mathematics. Yet ignoring a bullet point comparison and the superficial similarity of the two programs, they are in fact very different.

Both calculators provide a user interface based on handwriting recognition. But this is where the similarity ends.

The new calculator, was designed using the *flow* user interface principles of *projection*, *continuity*, WYSIWYE and *declaration*, in contrast, *xThink* seems to merely add the idea of utilising the affordance of pen and paper without escaping some of the typical problems that calculators have.

### Handwriting recognition

With the new calculator, recognition of the handwritten symbols is *projected* into the user interface. As a symbol is recognised the user interface is immediately updated replacing the hand drawn symbol with the typeset recognised symbol. The replacement of the user's handwriting with typeset symbols not only provides an immediately neat and tidy (and correct) equation but also provides immediate visible feedback of what was recognised.

The recognised equation is projected as a neatly formatted equation in the user interface. This provides the user immediate feedback about the expression recognition. The displayed typeset equation *is* the equation that



the answer is calculated for. This in-place visibility removes confusion and miss-understanding over what the calculator is doing, and whether it had handwriting recognition has occurred as the user is entering symbols.

The *projected* recognition means the user's view and the computer's interpretation are identical. There is no separate user data used to generate the display, thus the user sees exactly what is happening and has confidence in the answer. The *projection* of what equation the user is trying to solve makes any errors obvious as they are writing. This immediacy and clarification of the user's desired result (and any deviation from it) is one of the primary reasons that intermediate errors are quickly noticed and the lack of final errors.

Unlike the new calculator where the symbol and expression recognition are *projected*, in *xThink*, the handwriting is not replaced with typeset symbols and the mathematical expression is not typeset. This means that the user actually has little confidence that the recognised expression is the correct one. A handwriting or expression recognition error is not easily visible to the user and the answer given could be the answer for a different expression to the one the user believes they wrote.

### Continuity from input to output

The new calculator provides *continuity* between the user's input and the typeset output. This allows the user to easily understand how the resulting mathematical expression corresponds to the one they wrote.

In *xThink* there is only minimal feedback about the mathematical recognition and no continuity or even linking is provided between the input and that result. The feedback is a linear representation of the mathematics in a unconnected part of the user interface. If the user even notices the feedback there is no feedback providing the link or relationship of it to the user's input.

### Getting answers

The *xThink* user also has to be aware that once they have finished an equation they still have to press the **Enter** button, this time switching mental modes from "entering" to "getting the answer." In the new calculator the answer is a *projection* of the recognised mathematical expression and appears as soon as the expression has been recognised.

### Partial input

As part of the *projected* user interface, the new calculator supports partial input of mathematical expressions. This allows the user to easily form an

expression over time and provides simple feedback to the user about any incompleteness. *xThink* throws obscure error messages when the user enters an incomplete expression.

### Declarative interaction

The new calculator allows users to edit both sides of a mathematical expression, giving more flexibility and power. *xThink* does not provide anything similar; to get the same results users will have to rearrange even simple sums.

### Flexible editing

Editing in the new calculator is simple and easy, the user can edit the ‘ink’ they see by dragging, adding and deleting in a WYSIWYE way. Editing user input directly in *xThink* is impossible! *xThink* only lets the user create new expressions which are grouped and recognised when the user presses the **Enter** button. If the user does not carefully erase old expressions in *xThink* the user interface quickly becomes cluttered and new answers end up appearing on-top of old answers making any interaction confusing.

Once some mathematics is recognised in *xThink* by pressing **Enter** it is no longer editable. *xThink* provides a much poorer experience when entering any mathematical expression unless it is written and recognised perfectly the first time.

### Modelessness

In *xThink* the user has to switch both mental and physical modes many times. To erase or move parts of the equation the user has to select different tools at the bottom of the screen, then when they have finished the user has to remember they are in a special mode and reselect the original tool. The *xThink* interaction style makes this cumbersome approach unavoidable in principle.

With the new calculator there are no modes, and no user context switching. Not only are there no multiple different tools or modes but (obviously) there is no need to switch mental modes or to pause and press an **Enter** button. This greatly simplifies the user’s mental model and reduces the effort required to use the calculator. There is also no synchronisation problem where the user’s model can become out of step with the system model.

## Rearranging

In *xThink* it is possible to delete things or move them around, but it is always an awkward process involving several mode changes and it is fairly limited in what it achieves. Moreover, *any* editing in *xThink* breaks the relation between written input and calculated output.

In the new calculator the ability to drag and drop, using WYSIWYE, an arbitrary part of the equation elsewhere is synchronised by the calculator's ability to provide *continuity* by morphing the result into a new typeset equation. It is therefore possible to move parts of the equation around without regard for their size or shape, and the user *always* sees a fully correct equation.

### 7.7.2 Worked example

To better illustrate the differences between these two superficially similar interfaces, the interaction the user employs to solve a simple sum, along with the potential pitfalls is described in this section. This also provides a more concrete example of the differences between the two user interfaces by providing a step-by-step walk through of an example calculation in both user interfaces.

#### Initial input

In both user interfaces the user starts by writing the sum on the screen, using a pen (or using their fingers on suitable touch-sensitive screens). From then on the user interaction is different.

- In *xThink*, the handwriting is recognised in a separate location, which the user must read to check the accuracy of the handwriting recognition. If the handwriting is misrecognised by *xThink* then, without checking the small text at the bottom of the screen the user can easily be fooled into thinking they have the correct answer. The text at the bottom of the screen is both small and linearised, losing the benefit of the handwritten two-dimensional notation — for example Figure 7.11 shows the cube root of twelve cubed being calculated, it is printed as  $12^3 \cdot (1/3) = 12$ .
- In the new calculator, as the user writes, the hand-written characters and numbers are converted to typeset symbols and immediately *projected* without any further user action. The user feels as if they are writing typeset characters and confirming recognition is as natural as checking that your own handwriting is legible.
- *xThink*'s lack of *projected* recognition hinders the interaction. The visible hand-written equation and the parsed equation that the com-

puter understands are different. Without checking and making sure that the hand-written equation is the same as the computer's interpretation it is extremely easy for the user to be confused and misled. The two views of the equation, the handwriting and linearised mathematics are separate, distinct and different. *xThink* also provides no *continuity* between the input and output thus making it harder for the user to follow what has been computed.

### Incomplete input

During entering a complete mathematical expression the user's input is rarely complete or mathematically correct.

- In *xThink*, to determine the answer, the user's input must be syntactically complete (an expression). For example, to find the value of  $\frac{1}{\sqrt{4}}$  the user must write exactly this (and it must be recognised correctly). Anything else results in an error.
- In the new calculator, answers are provided even from incomplete expressions, as well as with expression. For example, to find the value of  $\frac{1}{\sqrt{4}}$  the user can also write  $\sqrt{4}$  the incomplete fraction is completed for the user. The user can then leave it as is, or correct the expression to exactly what they want. Answers for incomplete input are displayed as the user writes so that the user is provided with feedback as they construct a complete mathematical expression.
- By providing answers for partial and incomplete input the new calculator allows for a *projected* almost instant update of the answers whilst retaining sensible and useful answers. Without this ability the immediacy of the user interface is reduced.

### Getting the answer

Once the user has finished entering a mathematical expression they want the answer.

- In *xThink*, to determine the answer, the user must press another button, and the answer is displayed somewhere nearby the handwriting on the screen. In Figure 7.11 all such answers have been positioned under their respective formulae.
- In the new calculator, the typesetting *includes* solving the equation. When entering  $\frac{4+5}{3}$ , the user interface will show a typeset  $\frac{4+5}{3} = 3$  — the user wrote  $\frac{4+5}{3}$  and the computer inserted  $= 3$  in the correct position automatically.
- The answers and the input are inconsistent in *xThink* until the user presses the **enter** button. This lack of immediacy further confuses and

misleads the user, especially when old answers are still shown on the screen and can be located near the user's new handwriting. The new calculator is never inconsistent because the user interface is *projected*.

## Editing

To correct an expression or to modify it to a new expression the user needs to be able to edit.

- In *xThink*, the user's handwriting can be altered, but the answer is not updated, thus making the answer invalid. There is no easy way to edit mathematics other than erasing and starting again. What the user assumes are edits or corrections are treated as new expressions and additional answers are added to the already cluttered screen. It is possible for several answers to accumulate when the user evaluates formulae and old answers are not removed, confusing the expression even further.
- In the new calculator, the editing of the user's input is integrated into its evaluation. Thus the user can then continue to write over the top of this morphed equation, adding in bits that are missing.

It is possible to edit by inserting, overwriting and by drag-and-dropping symbols to a bin to delete them, or to other parts of the equation to move them, WYSIWYE. In all cases, the equation preserves its mathematical truth with *continuity*, as the new calculator continually revises it. A full undo function is also available, which animates forwards and backwards in time — also showing correct equations.

In the new calculator editing happens naturally as the user sees it. Using a WYSIWYE approach any edit is immediately *projected* and incorporated into the mathematical expression and the new expression morphed providing *continuity* and the answer given. This makes editing to alter or correct the expressions fast and easy. Combined with handling incomplete answers this means an expression can be built up easily from its component parts. For example,  $\sqrt{4}$  could be entered as  $\sqrt{\phantom{x}}$  then 4, or 4 then  $\sqrt{\phantom{x}}$ , and the user could write = if they wish. In any case, the value =2 or 2 is also displayed.

- Editing is not easy in *xThink* and it is compounded by the way answers accumulate which is very confusing. This is avoided in the new calculator because of the *projected* user interface, old answers are immediately updated as the input changes.

The comparison to *xThink*, which looks the same but lacks the *flow* principles used in the design of the new calculator shows that the principles provide real benefits. The main differentiating factor between *xThink* and the new calculator are the *flow* principles, the remainder of the interface is

Cognitive Dimension	Handhelds	<i>xThink</i>	New calculator
Viscosity	high	high	low
Visibility	low	medium	high
Premature commitment	high	medium	low
Hidden dependencies	medium	medium	none
Role-expressiveness	none	high	high
Error-proneness	high	medium	low
Abstraction	medium	none	none

**Table 7.1: A comparison of calculators using Green’s Cognitive Dimensions framework**

very similar. The comparative failure of *xThink*’s user experience strongly suggests that the *flow* principles are the key reasons that the calculator’s user interface is enjoyable and successful.

## 7.8 Cognitive Dimensions evaluation

Green [1989] proposed cognitive dimensions as a vocabulary for discussion and tools for usability evaluation or heuristics for guiding design. They are useful for discussing the calculator and allow for a heuristic evaluation of its usability.

Green’s [1989, 2000] cognitive activities and cognitive dimensions (CDs) provide a way of critiquing and comparing interfaces and their uses. They provide both a set of discussion tools, an “analytical vocabulary for design discussion” and tools for heuristic evaluation.

The table above exhibits Green’s original cognitive dimensions, comparing the new calculator with standard handheld calculators and with *xThink*. These dimensions are not consistently good attributes, for example Green’s *viscosity* is a bad attributes, whereas *visibility* is a good attribute.

*Viscous* interfaces make change difficult and hard to achieve. Changing anything on handheld calculators is usually hard. Often this is a result of having very simple user interfaces, many of them lack any ability to change any input — requiring a complete restart of the user’s actions and a loss of all intermediate work. *xThink* also makes change very hard by offering very few editing capabilities and often requiring the user to start over again. The new calculator, using ink editing, allows a very fluid (as opposed to viscous) user interface that flexibly supports editing and change, through add new symbols, deletions and drag and drop. The fluidity of the user interface is a direct consequence of the WYSIWYE principle and the ease of editing mathematics as they are seen.

*Visibility* was one of the driving design principles of the calculator stemming

from *projection*. The whole expression is visible all of the time and this seems to lead to users having more confidence in the results and being able to use the calculator faster. Compared to current calculators that are often only visible to the extent of the last numeric value squeezed into their display, the improvement is dramatic. *xThink* shows the whole mathematical expression but only provides direct feedback about recognition in a linear text string, it also does not provide any visible linking between the user input and the result.

*Premature commitment* is a problem that most handheld calculators exhibit: rarely if ever is the user allowed to undo or alter what they have calculated. Even in template editors and some pen-based entry the mathematical expression has to be built top-down in a rigid ordering. *xThink* allows parts of the expression to be erased or moved, but only before the mathematics is computed, this means that the user has to unnecessarily ‘commit’ to the expression before they can get the result. In the new calculator a mathematical expression can be constructed in any order and ink can be dragged arbitrarily, anything the user has done can be altered and is not premature. This is in part enforced by WYSIWYE, because the user is able to edit any part of the mathematics at any time, no part is ever committed or unchangeable.

*Hidden dependencies* hide the links between entities, such that a user cannot easily discern what the behaviour of the user interface will be. Standard handheld calculators often have several modes which determine how they operate, these are hidden and can confuse the user. *xThink* has hidden dependencies between input and output, it is never clear what is linked to what. This makes it tricky for the user to know how *xThink* will react to their input. The new calculator in contrast has no *hidden dependencies*, the WYSIWYE user interface means that any dependencies do not affect the user’s interaction.

*Role-expressiveness* should merely be a simple matter of doing mathematics right! However calculators disturbingly fail to do this [Thimbleby, 1996], in particular their failures to properly provide proper syntax or referential transparency are obvious (and surprising) failures of design. The new calculator and *xThink* both do mathematics correctly without unusual or confusing syntax, for example using  $\wedge$  for exponentiation.

*Error-proneness* is a problem for any calculator. Users depend on calculators to perform calculations they could not do otherwise and therefore often put misplaced trust in the answer. Errors arising in use are rarely, if ever noticed, whether they are caused by the user or by design. Current calculators exacerbate this problem by failing to provide a visible history or even error messages. *xThink* hinders the user and encourages errors by providing very little feedback about what mathematics was computed. In comparison the initial evaluation showed the new calculator was successful at reducing errors, certainly making users less error-prone. Both *projection*

and *continuity* are key to providing feedback and reducing errors.

*Abstraction* provides additional layers of complexity on top of the user interface that the user has to learn. Handheld calculators often abstract mathematical input to entering a linear sequential button presses, putting the burden on the user of converting their mathematics into button presses. Neither *xThink* nor the new calculator abstract mathematics, they both provide pen-based user interfaces that work like paper.

Green also uses more general terms for describing the type of interaction a user interface supports, these are: incrementation, transcription, modification, exploratory design, searching and exploratory understanding. Using Green's terms, most handheld calculators provide trivial *incrementation* and partial *modification*. A standard calculator's  $\boxed{C}$  and  $\boxed{AC}$  keys provide very crude modification and the changes the user can effect are almost entirely incremental. More flexible calculators do provide *modification* allowing the user to edit the mathematical expression. The same types of interaction are supported by *xThink*. In contrast, the new calculator is very flexible providing both *modification* and *incrementation*, and it provides more complex activities like *exploratory understanding* which is encouraged by all four of the *flow* principles. Making it is very easy to discover the underlying mathematical structure by using the calculator to do sums and to play.

Using cognitive dimensions within the context of activities provides us with an interesting picture. Both standard handheld calculators and new calculator provide simple incrementation and partial modification, but the new calculator supports further cognitive activities: exploratory understanding and unrestricted modification. This suggests that handheld calculators could provide a better interface for accountancy and summing numbers (e.g.,  $123+345+435.98+123+\dots$ ) when exploration and unrestricted modification are undesirable. However for performing arbitrary calculations, especially exploring and learning, the new calculator supports much more flexible and powerful activities.

The design of the new calculator is supported by a heuristic evaluation using cognitive dimensions. For all of the relevant dimensions the calculator has been evaluated against it comes out overwhelmingly positively. The heuristic evaluation substantiates both the design of the calculator and the underlying principles that informed the design.

## 7.9 A note on the philosophy of science

One might like evaluations to confirm certain design principles, but this would be poor science.

Empirical evaluations can only provide *confirming instances* of the hypothesis that certain principles are effective for design; strictly, experimental eval-



uation (when undertaken appropriately) can only refute hypotheses [Popper, 1963], since confirming instances might have been caused by other factors that were not controlled for. In this case, there is the possibility that the results are caused by, among other things, the relative quality of programming, which is not a factor that this thesis explores.

The calculator only exists as a confirming instance of the utility of the design principles, rather than proof of their validity. This is perhaps an insurmountable issue, it is impossible to evaluate these principles directly, except through their implementation which may be skewed by other factors. Thus any justification of the principles in this thesis also relies on appeal to argument rather than by evaluation alone.

## 7.10 Summary

Both the underlying mathematical engine was evaluated by Cairns et al. [2004] and pen-based mathematical entry by Anthony et al. [2005]; each was found to be successful.

The new calculator was evaluated with user tests, performing timed tasks. This evaluation found that the new calculator was faster for some problems, let users solve mathematics they could not have done otherwise, and produced fewer errors. The fact no user got an answer wrong using the new calculator is especially pertinent, as getting the correct answer is the critical part of using any calculator.

A large amount of user data was collected from the Royal Society exhibition which provided numerical data stating users found the calculator more enjoyable and more helpful. The exhibition also provided a large amount of user feedback, the majority of which was very positive and supports both the numerical data and the novel design of the new calculator's user interface.

A direct comparison with *xThink* provided a point-by-point contrast of two similar user interfaces where the *flow* principles were the main differentiator. These comparisons substantiate the arguments for both the design of the new calculator and the *flow* principles.

Finally a heuristic evaluation using Green's Cognitive Dimensions supplied a more analytic discussion of the differences between calculators and what the *flow* principles achieve. This suggests that the new calculator is more suited to exploration and learning than to tasks such as accountancy.

In summary, the new calculator has been evaluated in many different and complementary ways. Each of the evaluations has provided different supporting evidence for the success of the calculator and of the underpinning *flow* principles. Collectively, the evaluations provide a compelling case for the design principles used in the calculator.

**Part II**

**Drawing**

## Chapter 8

### Context



Figure 8.1: Cave paintings in Lascaux, France (circa 15,000BC)

#### 8.1 History

Along with calculation, drawing also has a long history. The earliest known rock paintings have been dated to 40,000 years ago and cave paintings to 32,000 years ago. Figure 8.1 shows the famous Lascaux cave paintings from France that have been dated to 17,000 years ago.

Modern drawing aids such as paper and pens have their origins in Egypt with papyrus and inks. Other drawing tools like erasers, slate boards, rulers, drafting tools and correcting fluid have all been invented for different purposes and provide different capabilities that support drawing.

Computers have always been used to generate graphics, but it was not until the 1960s with Sketchpad that they were used as a user interface for drawing. Sketchpad developed by Ivan Sutherland [1964] as his doctoral thesis, provided a graphically interactive user interface for drawing. Sketchpad used a

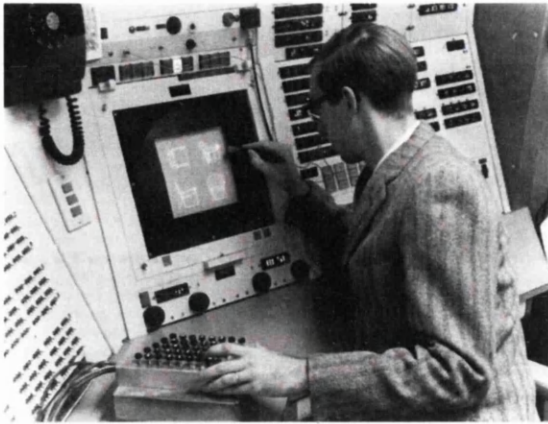


Figure 8.2: Sketchpad (1963). Source: Sun Microsystems

light-pen to point and a bank of switches to control operations like ‘move’ and ‘draw’. It is generally thought of as the pre-cursor to modern computer drawing. Since then computers have begun to be increasingly used to provide support for drawing activities. A large factor in the early use of computers for drawing was Computer Aided Design (CAD). The accuracy of the drawings and the ability to alter the drawing easily were the primary reasons why CAD took off commercially.

The first computer painting program was probably Dick Shoup’s “Superpaint” at PARC (1974–75). Superpaint introduced the distinction between vector graphics and raster graphics, this is a distinction created by how computers store images, as either mathematical vectors or as a grid of pixels.

Drawing programs were first widely used on the Macintosh in 1984, which came with both MacDraw and MacPaint, providing both vector and pixel drawing tools respectively. Since then many commercial programs have been in wide use, for example, Adobe Illustrator, which was first developed in 1986. Illustrator is currently the main professional vector drawing program and is widely used by designers and graphics professionals.

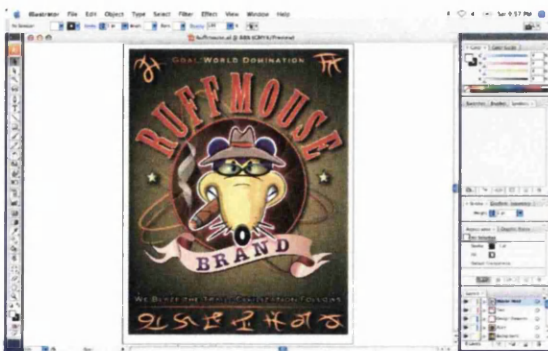


Figure 8.3: Adobe Illustrator CS3 (2007)

## 8.2 Drawing applications

In contrast to calculating, Chapter 2, there has been relatively little research on user interfaces for drawing. After Sketchpad broke ground early on, there has not been much focus on general drawing user interfaces. Instead the majority of drawing research has been focused on different areas such as: the underlying algorithms [Hsu et al., 1993], sketching for 2D and 3D input [Arvo and Novins, 2000, Zeleznik et al., 2007b], morphing vector shapes [Vronay and Wang, 2004] and more esoteric ideas such as novel user input devices for curves [Grossman et al., 2003] and representations of gradients using vector partitions of space [Orzan et al., 2008].

User interfaces for the majority of vector drawing have remained untouched by research since the early days of Postscript in 1982. In fact the backbone of vector graphics editing applications, cubic Bézier splines and their corresponding user interfaces have remained mostly unchanged for over 20 years.

Although more general user interface research is applicable such as direct manipulation [Shneiderman, 1983] and WYSIWYG.

Thus this context is by necessity defined by the successful commercial applications that have defined this domain. Those applications primarily being Adobe Illustrator, CorelDraw and Inkscape.

## 8.3 Drawing in vector applications

This section provides a discussion of the purpose of a drawing program and what features and interactions are required. In this context the term “drawing program” is used to refer to programs whose primary purpose is to create vector graphics.

### 8.3.1 What are vector graphics?

Two-dimensional vector graphics are graphics that are composed mostly from geometrical data, where the data usually forms a kind of “recipe” for generating a final image. A simple example of what could be a vector graphics recipe is “draw a red circle on top of a blue square”. This tells the computer how to arrive at the final image. Real vector graphics are much more specific and describe the exact shape, positioning and styling of the different components. Vector graphics are used widely in different domains and some common file formats are: DXF, SVG, PDF, and PS.

Almost all vector graphics are composed from a few basic “ingredients”; shapes and the instructions about how to draw them. While there are few

basic component types, vector graphic pictures can contain thousands of components and can be very complex.

Most graphics on computers are not vector but instead are raster graphics which are constructed from an area of lots of coloured pixels. Compared to vector graphics, uncompressed raster graphics are relatively simple for a program to draw. Adobe Photoshop and Microsoft Paint are examples of programs that edit raster graphics.

Raster graphics often originate from input devices such as digital cameras, which take pictures that are raster graphics. The majority of graphical input and output devices including scanners, cameras, screens and printers are all raster based. Graph plotters are one of the few examples of an output device that uses vector data.

In contrast, most vector graphics are created from scratch by hand. The differences in how raster and vector graphics are created dictates the focus of the respective applications for manipulating them. Raster graphic applications are often focused on editing and adjusting input like photos, vector graphic drawing programs are usually focused on the creation of vector graphics.

### 8.3.2 Why use vector graphics?

Creating vector graphics on a computer provides many advantages over both raster graphics and the traditional methods of drawing using pens and paper. Vector graphics are far more flexible and are very versatile in how they are edited and manipulated. Every aspect and component of a vector image is adjustable. Raster graphics and to a greater extent paper lack this versatility and changes made to the graphic are rarely later adjustable. Although the description of editing vector graphics in future sections makes creating vector graphics seem complex, this is in part because of their flexibility.

A pen and paper interface provides two basic interactions, *drawing* on the paper and *erasing* part of the drawing on the paper. Raster editing programs provide a few more interactions, but they are still not as flexible as vector drawing applications. A drawing program additionally provides the ability to rearrange, move, reorder, change, delete, stylise and adjust each individual component of the drawing in hundreds of different ways.

It is this extensive amount of possible interactions that make vector graphics editors a powerful tool for a designer and at the same time means that the respective user interface is often complex.

### 8.3.3 Technical reasons

Raster graphics and vector graphics have different benefits. Generally raster graphics are best for photographs and vector graphics are better for text or diagrams.

One of the main advantages of vector graphics is that because they are a recipe for creating an image, the recipe can be adjusted to create the best image for the device they are to be recreated on. For example a printer has a much higher resolution than a screen, and thus by using a recipe it can generate an image that will look as sharp as possible. If instead a raster graphic was being printed, the image cannot be altered and it is printed at the fixed resolution of the image. Vector graphics are therefore often used in the print industry.

Another advantage of vector graphics is that it is possible to alter the different steps of the recipe very easily. This means that for an artist creating a graphic, vectors provide a huge benefit because every single part of the graphic can be altered without any problems.

### 8.3.4 Where do they come from?

The main source of vector graphics is from people using drawing and illustration applications to create images in graphical user interfaces. Almost all vector graphics originate from the user in such a fashion. They are also often generated automatically as a visualisation, for example creating graphs in Microsoft Excel generates vector graphics.

### 8.3.5 Semantic requirements

There are many different types of recipes for vector graphics, and each can have different abstractions. One recipe might simply specify drawing a star, another recipe could specify the same star as a polygon of individual points.

Despite there being a large number of different types of vector graphics, the basic components of vector graphics are fairly simple and consistent, these are: shapes, styles and rules of composition. Shapes are described using polygons, Bézier curves, circles, or text. Styles provide the method with which to draw the shapes utilising information like fill, stroke, colour, gradients and transparency. The individual shapes are then composed together for the final result using composition rules, including clipping and Z-order rules (what is on top of what).

Almost all vector graphics are built from these basic components, however very few vector graphic representations stop there. Many representations provide more complex ways of describing drawings, for example using

Gouraud shading, vector effects and raster filters are more complex aspects of different vector graphic formats.

## 8.4 User interface requirements

It is possible to create vector graphics by hand in a text editor, for example using PostScript, but the nature of the data that is being edited makes this very difficult. Very few people create vector graphics this way — most users make use of a graphical drawing program such as Lineform.

Using a vector graphics user interface, the user edits and manipulates the underlying recipe through a graphical representation. This involves user interaction to create and edit shapes, style and compositing information. This interaction should happen in a way that is as logical and as direct as possible. Building on the principles discussed in Chapter 4, the interface should also incorporate the ideas of *projection*, *continuity*, *WYSIWYE* and *declarativeness*.

This section summarises the main requirement for a drawing program. Some aspects of this section are based on the absolute requirements of drawing applications and others are more based on the cultural expectations and desires of users.

### 8.4.1 Graphics

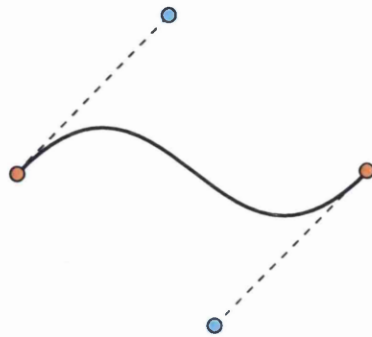
Allowing the direct manipulation of graphics is very important because the raw numeric data that the shapes are created from is generally very hard to understand and edit. Direct manipulation means the user can change and alter the drawing directly using an input device such as a mouse. There are no widely used modern graphical drawing programs that do not provide a direct manipulation of the drawing.

A WYSIWYG view of the drawing is essential in both providing a view of the drawing as intended and allowing the user to interact with the graphics to achieve their desired result. Without WYSIWYG, editing even simple drawings is hard to achieve. Non-WYSIWYG graphical editing is still useful for diagram specification, eg. dot files, and graph generation.

### 8.4.2 Bézier splines

Cubic Bézier splines [Bezier, 1972] are the most general shape description which many vector editing applications provide. These are composed of multiple cubic spline segments which have start and end nodes and two intermediate control points. A larger shape or path is composed of many segments combined together to make up the final shape.





**Figure 8.4: A cubic Bézier segment**

$$\mathbf{B}(t) = (1 - t)^3 \mathbf{P}_0 + 3(1 - t)^2 t \mathbf{P}_1 + 3(1 - t) t^2 \mathbf{P}_2 + t^3 \mathbf{P}_3, t \in [0, 1] \quad (8.1)$$

Figure 8.4 shows a single cubic Bézier segment, the start and end points are highlighted in orange, the two control points in blue. The control points describe the curve's tangent at the start and end but do not lie on the actual curve segment itself. The thicker spline in between the start and end points is described by Equation 8.1 which is a parametric cubic equation in  $t$ .

Several other spline types are also used in drawing applications, most notably B-splines and quadratic Bézier splines. B-splines have better smoothness properties and are thus used more frequently in CAD software, but they are often harder to use. Quadratic Bézier splines are simpler to use but not as flexible as the cubic Bézier spline. Cubic Bézier splines have become popular because they occupy a good middle ground; they are relatively easy to use and provide good flexibility.

### 8.4.3 Tools

Due to complexity and the many different tasks the user can be trying to accomplish in a drawing application, there are too many ways of interacting without using different modes of interaction. This is generally accomplished by a set of tools that can be selected one at a time, these provide a modal interaction experience focused on one task, when the user needs to perform another task they have to switch tools.

Examples of tools are: rectangle, oval, pen, selection, node editing, zoom and text. These tools are used to provide interaction modes for creating shapes and editing the drawing.

#### 8.4.4 Selection

Many operations and much of the editing is performed on a small portion of the whole drawing. To do this there needs to be a means of specifying a portion of the drawing that will then be affected by the user's actions. Specifying a selection is usually performed through direct clicking or dragging a selection marquee over graphics when using a selection tool. The selection should be clearly visible so that the user has the visible feedback about which objects are being affected.

Selection is also important within different modes for more detailed editing. Both text editing and Bézier path editing involve their own concept of selection, allowing the selection of characters or Bézier nodes respectively.

#### 8.4.5 Editing

Different shapes require different editing interactions; paths, text, rectangles and groups all have various properties and need to provide differing methods of editing. To edit Cubic Bézier spline paths, the user needs to be able to create, delete, move and position the nodes and control points of each spline segment. Editing paths is often the single most complex part of creating vector graphics.

Editing text needs all the standard text interaction providing: fonts, sizes, styling, colouring, and paragraphs. Other standard text tools such as finding and spelling improve the interaction. From the user's point of view the closer editing text is to the word-processor they are familiar with the better.

Special shapes such as circles, rectangles, spirals and polygons, all have different specific attributes such as corner radius or the number of points. The user has to be able to edit these attributes, the more direct the editing is, the better.

The ability to group graphics together provides coherence in a drawing and allows certain effects. A group links graphics together, for example a face composed of eyes and a nose on a background, allowing the user to interact with the group as a whole rather than as separate components. Editing groups requires a distinction between editing the group and editing its contents, the ability to do both is important.

All shapes should be manipulatable on the canvas. Positioning, rotation and scaling are generally provided by affine transformations. In addition to the many ways of editing and manipulating shapes directly on the canvas, there are also many other actions or operations, for example alignment and Boolean composition, that provide useful capabilities to modify shapes.

### 8.4.6 Navigating

The ability to navigate the document is necessary for the user to be able to view larger documents than fit on the screen or edit small details too small or awkward to edit at the document's natural size. The common navigation operations are panning the document to view different sections of it and zooming in to work on detailed parts of the drawing or zooming out to work on larger overviews.

### 8.4.7 Style

Once the component shapes of a drawing have been created, what they look like needs to be specified. Most vector graphics provide two main components to style: the stroke and the fill. The stroke of a shape is how the line around the outside of the shape is drawn and the fill specifies how the interior of a shape is drawn.

A stroke style can have attributes such as: width, colour, opacity, dashes, and end cap shape. Some of these, like width and dashes, specify the shape of the stroke, others like colour, specify how to draw it.

A fill style generally has fewer attributes, the fill of a shape can usually only be drawn as solid colours, potentially with an opacity. More complex fills are often provided through a clipping shape that contains other drawing components that are only drawn inside the clipping shape, for example a clipping shape could contain more shapes, an image, or a gradient.

### 8.4.8 Compositing

The ordering in which objects are drawn, the Z-order, affects the final output. There needs to be a user interface to rearrange objects in this dimension. Standard ordering actions are: bring forward, bring to front, send to back, send backward.

Also to help manage the Z-order of the drawing, programs often provide layers that contain many objects. Layers provide a coarse grouping of graphics and can be rearranged themselves and often allow the toggling of visibility and locked status. These enable better management of complex documents.

Most modern drawing programs also support colour blend modes, which describe how the colours of graphics are composited together.

### 8.4.9 Filters

Many modern vector drawing programs also now support raster effects. These convert the vector shapes into a raster image and then apply a raster

effect, this allows effects to be created that are nearly impossible with only pure vectors. Examples of raster effects are Gaussian blur, solar flare and crystallise. Raster effects can also be layered so that, for example a blurred flare is possible.

#### 8.4.10 Documents

Documents contain whole drawings, the user needs to be provided with some document management, to allow different sizes of documents, multiple pages and possibly features like markers and grids.

### 8.5 Poor user interface design

This section describes the bad user interface design of products similar to Lineform, which played a large part in inspiring Lineform's design. The aspects of these interfaces that have seemed to provide bad user interaction design are highlighted and individual examples are used to demonstrate different aspects of poor user interface design. The interaction design that is highlighted in these examples is not exclusive to the individual drawing applications used as examples. The same user interface design flaw is found in many other drawing applications.

In many cases there are legitimate reasons for the designer's choice. For example, Illustrator comes with the legacy of 20 years of features and was originally designed to work on hardware much slower than today's. However while these reasons might provide good excuses, they are now unimportant to the design issues discussed; as such these decisions will not be more than cursorily defended.

There are many aspects of the design of the various drawing programs that are basic and obvious bad user interface design. Examples are inconsistent and cluttered user interfaces, obscure user interaction, non-standard user interface controls and slow and even buggy interaction. Describing these issues does not provide any insight into the development of new user interfaces and are ignored for the purposes of this section. The examples of interaction design described in this section have been chosen to highlight aspects of design that are common to drawing programs. These include the lack of direct manipulation, unclear use of modes, over-rigid design and the lack of immediacy of interaction.

#### 8.5.1 Direct manipulation

Drawing programs provide an essentially visual interaction; the primary user interaction is through manipulating the visual drawing on the canvas. Direct

manipulation [Shneiderman, 1983], which captures the concept of directly interacting with the object of interest, is especially apt when applied to these user interfaces. Some of the advantages of this kind of interaction are the ease of learning, exploration, avoidance of errors and user satisfaction [Shneiderman, 1997].

A real pen or paintbrush provide a very direct interaction when drawing, in comparison a mouse and keyboard provide much more indirect control. Thus they are harder and less intuitive to use. The further interaction is removed from manipulating the drawing directly, the less obvious and easy it is to interact with the drawing.

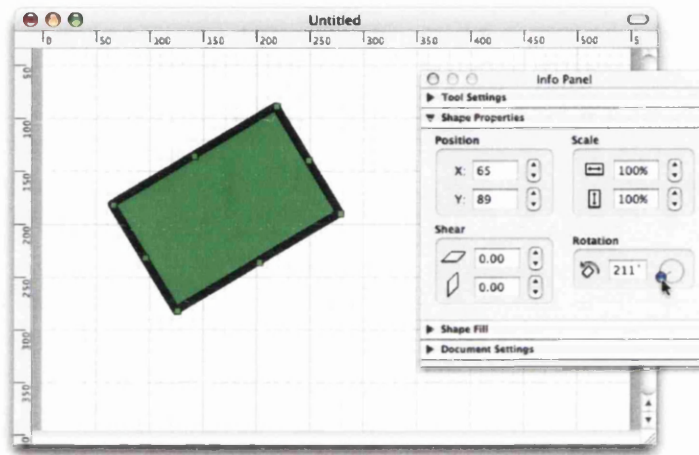
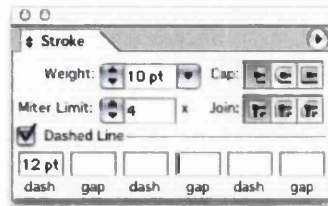


Figure 8.5: Indirect rotation controls in iDraw

Several programs have an indirect approach to some manipulations of the drawing. This is often because it is easier to create indirect user interface controls, that can be abstracted from the ‘physical’ graphics they are manipulating, than to provide a method of directly manipulating the graphics. One example of an indirect user interface controls in drawing programs is rotation controls that are located in palettes. When using these the user has to manipulate a small control that is not directly linked to the graphics they are trying to rotate. Interacting with a small control rather than directly with the graphics is awkward because the user has to link their interaction to the results. The user also cannot both look at the control and the graphics at the same time and thus has to switch their attention back and forth, making it awkward to get the right value. Figure 8.5 shows the rotation control in iDraw, the circular-slider rotation control in the palette is small and fiddly and the graphic being manipulated is visually separate from the user interface control.

A different situation where user interfaces lack direct manipulation is when they force the user to edit an object through an interface that has a different form to that of the object. In this case the object can be edited directly but



**Figure 8.6:** Indirect stroke weight in Adobe Illustrator

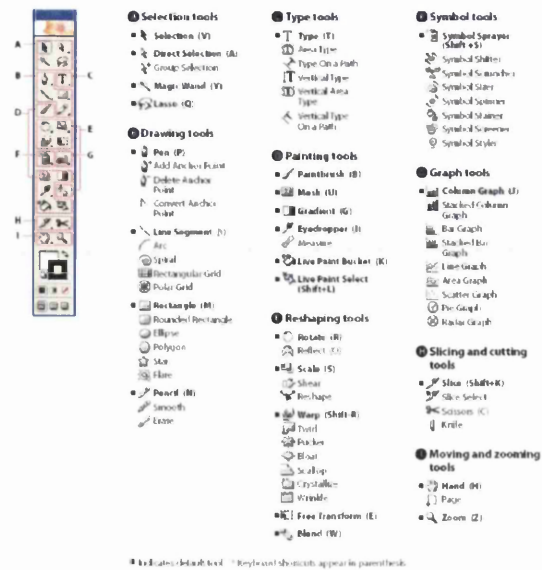
the interaction is not natural. Figure 8.6 shows Adobe Illustrator's stroke palette, to change the stroke weight in Illustrator, the user either has to type a weight in, select one from the pop-up menu, or use the stepper controls to the left of the text field. All of these interactions with the user interface to change the stroke weight provide a discrete interaction, that is, they do not allow the user to continuously change the value. (A slider is an example of a user interface that allows continuous interaction.) The problem is that these controls provide a discrete way of interacting with a continuous value, forcing the user to interact with the data in a way that is not natural for the underlying data. Discrete interaction like this can be very useful, but it is less natural and can be a very awkward way of exploring a range of values.

### 8.5.2 Modes

User interface modes are the various states that a user interface can be in which determine how it reacts to the user's input. User interface modes make it possible for user interaction to make use of the same actions to do many things, instead of requiring new user interactions for each task. A simple example of a mode is the current tool selection in a drawing program. A brush tool mode draws a smooth curve when the user drags with the mouse in the document, compared to a rectangle tool mode which will draw a rectangle or a selection tool which selects graphics. Modes are often necessary for complex user interfaces but they can easily be abused.

Many of the drawing applications have many more modes than needed. Some modes are needed to enable the interface to provide enough functionality with the limited user input devices of the keyboard and mouse, but unnecessary modes are often added because creating modes is simpler than utilising different interactions. Modes can cause a lot of confusion and errors. The more modes a user interface has, the more interactions a single action can have and the more effort the user has to make to remember the individual functions of all the modes.

Modes fail to work when the user forgets which mode the system is in and expects their interaction to have a different effect than the one the current mode provides. This could be a completely different reaction than expected. Invisible user interface modes are bad design [Raskin, 2000].



**Figure 8.7: Adobe Illustrator CS2's list of tools**

Many programs like Adobe Illustrator, shown in Figure 8.7, have lots of tools and thus lots of modes. EazyDraw has 170 tools for different shapes and functions. In Adobe Illustrator's case many of these tools are probably included because of backward support causing the inevitable feature bloat that comes from being widely used by lots of people for different tasks for over 20 years.

Another approach to the task of rotating graphics is to have a specific rotation tool. Applications like Intaglio, EazyDraw and Adobe Illustrator all have rotation tools. While this method is better than a distinct user interface control, a separate tool means that the user interface has more modes and is more complex. In Adobe Illustrator it is possible to rotate graphics with several different interactions by:

1. Dragging near a corner of the shape with the selection tool.
2. Dragging near a corner of the shape with the free-transform tool.
3. Dragging anywhere with the rotation tool (this tool has several other modes as well).
4. Use the Object > Transform > Rotate menu to get a dialog box to type an exact rotation in.
5. Enter an exact value in the Transform palette.

In Illustrator's case the three individual tools (selection, free-transform, and rotation) duplicate a lot of the rotation functionality and can confuse the user simply by the number of options. Another problem of using a specific

rotation tool is that to rotate a graphic the user first has to select the rotation tool, then rotate the graphic, then select the tool they were originally using to continue. The unnecessary mode change slows the interaction down and can cause further errors.

### 8.5.3 Rigid design

When designing or drawing, artists often repurpose different parts of the drawing, using them for something different to what they originally started out as. During the creation of a drawing, solid coloured boxes can become text boxes, or text shapes turn into gradient filled shapes. In conflict with this flexible repurposing, most drawing programs are quite rigid in how they let the user interact with different graphics. Often there are several specialised types of graphics that each perform different functions. This specialisation means that graphics are prematurely specialised from creation and it is awkward to repurpose them for other uses.

In almost every other drawing program, individual concepts such as text boxes, images and text-on-a-line are separate concepts and rigidly distinct. These unnecessary distinctions between objects provide no usability benefit. They make the programmer's (not the user's) life easier because the distinctions between types of graphics usually mirror the underlying structure of the classes and objects in the implementation, which means that the user interface can be less complicated to create because it simply exposes the data structures of the program.

The increase in specialised types of graphics also affects the rest of the interface. Each kind of shape can be interacted with in a different way and thus more kinds of shape create more user interface interactions and controls. For example, Adobe Illustrator has a specialised graphic type for symbols, to interact with symbols Illustrator has extra palettes and eight additional tools. Instead of adding functionality through additional specialised tools and shapes, the same effect can be achieved by providing more generalised tools and shapes. A user interface with fewer specialisations has fewer components and is simpler.

In fact, the symbols graphic type in Illustrator provides very little extra functionality over using groups for the same purpose. The specialised symbol tools allow the user to do things like pushing, rotating, scaling symbols, these tools are only useful for using on symbols. If these tools were instead generalised to work on all graphical shapes then the tools would be more flexible, more consistent in their use, and the entire concept of symbols could be removed from Illustrator. This simple change would make Illustrator's user interface more simple and more powerful.



#### 8.5.4 Over complication

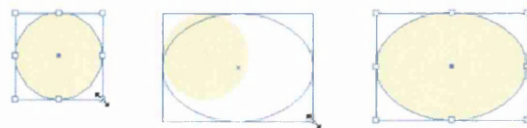
Adobe Illustrator has nine different graph tools, including a column graph and a stacked-column graph. It also has 65 other tools that do everything from ‘symbol scrunching’ to drawing polar grids. Illustrator’s whole interface has grown with feature bloat, it includes lots of different tools that all offer very similar functionality. As a consequence it seems that Illustrator’s design now lacks a lot of coherence. There are lots of different types of objects and styles, and most of them have a different method of interaction and a corresponding different tool.

#### 8.5.5 Lack of immediacy

When drawing, the ability to immediately see what changes are happening is a large part of being able to create drawings quickly. If changes to graphics only become visible after the user has finished interacting with them then the user can be forced into a slow iterative, almost trial and error style of interaction.

A simple example of the lack of immediacy is the proliferation of modal dialog boxes in Adobe Illustrator that do not provide live updates. These force the user to interact solely with the dialog box, restricting the user’s interaction, and also to repeatedly perform the same action using the dialog box, undo and try again when they are not sure what the result will be.

Setting the position of a gradient fill for a graphic is another example of a lack of immediacy. In Intaglio and Adobe Illustrator (prior to CS4, October 2008) setting the gradient position does not provide any real-time feedback. To set the position the user drags a line using a gradient tool across the graphics, the gradient is updated once the user stops dragging. As the user drags the gradient tool on-top of the graphics there is no visual update of the gradient other than a single line showing the dragging. This makes it very hard to tweak or adjust gradients, as well as making it hard to get the gradients right in the first place.



**Figure 8.8: Ghost resize outlines in Adobe Illustrator**

Figure 8.8 shows another example of an interface that does not provide immediate feedback. Here Illustrator uses ghost outlines rather than redrawing the graphics as the graphics are scaled. This works in a similar fashion to how dragging windows used to work in most operating systems since computers were not fast enough to provide immediate feedback for this style of

interaction 20 years ago. This lack of immediacy also makes it hard for the user to know what the result of their interaction will be.

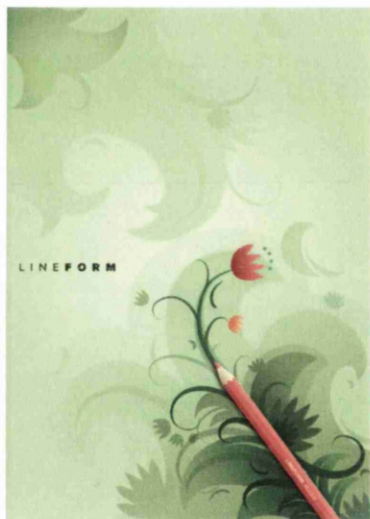
## 8.6 Summary

This chapter has outlined the context of drawing applications. The research within the area of vector drawing user interfaces is fairly limited and this context is primarily shaped by commercial drawing applications, specifically Adobe Illustrator.

What vector drawing applications should provide has been outlined, these factors provide the context for what any drawing application needs to support. Conversely the user interface design flaws of commercial drawing applications provide the context for what a drawing application should avoid.

## Chapter 9

# Design & development



Lineform is a vector drawing application, similar in functionality to applications such as CorelDraw and Adobe Illustrator. The purpose of Lineform is to enable users to draw, design and edit two dimensional vector drawings.

Lineform was initially designed and programmed as a hobby project intended as a drawing program for the author's personal use. After a few months of development and design it was decided to develop and polish Lineform further with the aim of releasing it commercially to the public. The remainder of the development of Lineform happened over a period of approximately a year and a half before it was released to the public for sale online, costing \$80. After the initial release, it continued to be developed as a product, and it has been through seven minor updates providing small bug fixes and four major releases providing extra features. Lineform is now published by Freeverse Ltd both in retail and online; and separately the rights have been bought by Apple. The following list is a summary of the major releases of Lineform.

- 1.0 — 4th April 2006, Initial release
- 1.1 — 16th June 2006, SVG support and raster filters
- 1.2 — 11th September 2006, Published with Freeverse
- 1.3 — 23rd February 2007, PDF editing support
- 1.5 — 23rd February 2008, Pressure support and new Layer and Transform inspectors

The calculator, described in Part 1, provides a good example of novel user interface design and of the principles that were critical to its design. This chapter describes the design and development of Lineform, which is a different example of a novel design and application. Lineform was designed and built after the calculator and its design builds on the principles and ideas that emerged from the calculator's development.

## 9.1 Motivation

The motivation for designing Lineform was slightly different to the motivation that conceived the calculator. Lineform was originally created out of frustration with existing drawing applications with regards to features, usability and price. Lineform's design is therefore partially motivated by solving the user interface problems and copying the good ideas from existing applications. The calculator was designed to be a completely new and better approach to mathematics. Lineform was designed to be a familiar but streamlined, focused and much easier to use approach to vector drawing, using some of the principles from the calculator.

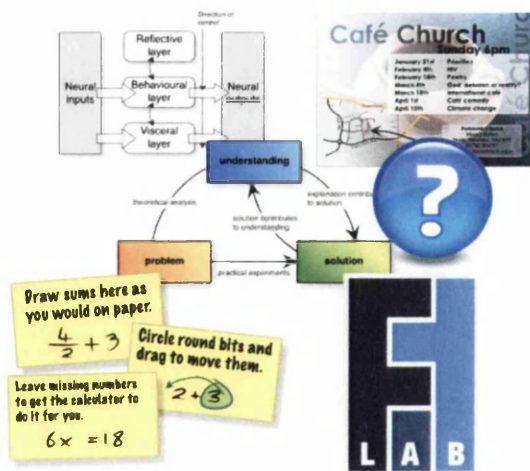


Figure 9.1: Example drawings and diagrams created in Lineform

Drawings and diagrams, such as those in Figure 9.1 are examples of the

type of graphic that Lineform was created to help produce. There are numerous vector drawing and illustration applications including EazyDraw, Corel Draw, Adobe Illustrator, OmniGraffle, Intaglio, iDraw and Inkscape, but none of these programs satisfied either in terms of their user interface or value-for-money. Some of the programs were too basic for general needs (iDraw), others seemed to have appalling user interfaces (most of them) and others were too expensive (Adobe Illustrator). Many drawing applications also lacked important features. Most problematic were the lack of functionality like Boolean operations, flexible and capable Bézier path editing, and the inability to interact and transform more than one shape at a time.

Lineform's design was both born out of the user interface design flaws visible in existing products and also in the novel user interface design of the calculator. Lineform is designed for today's computers with the knowledge of Illustrator's success and failures. By starting from scratch with Lineform, rather than 20 years of legacy, its design was streamlined and focused on the user interface.

Like the calculator, Lineform developed in a fluid way from these starting inspirations. The main principles and lessons from the design of the calculator were part of the original design of Lineform, which provided a very different type of user interface for their expression. And just like the calculator some original principles also evolved out of the Lineform's design and development process.

## 9.2 Initial design

Lineform was designed to be a simple, straight-forward and streamlined vector drawing application. While this design was not fancy or hugely different to other drawing applications it built on the four flow-principles from the calculator: *projection*, *continuity*, *WYSIWYE* and *declaration*. These principles were used in different ways to how they were used in the calculator.



**Figure 9.2:** Some of the first drawings done with Lineform (Spring 2004)

The design and creation of Lineform lasted about 2 years. The initial application was functional from the start and was used for various projects early on. Some of the first drawings done in Lineform are shown in Figure 9.2, these were created for a racing game and Lineform, although only a couple of months old, was used for all of the stylised drawings in the game.

Lineform was created over the course of the next 2 years, finally being released after a short beta period on the 4th April 2006. The initial feature list included:

- Easy simple powerful interface
- Powerful Bézier editing tools
- Boolean operations
- Great typesetting
- Artistic strokes
- Bitmap and vector import/export

Initially Lineform was called Inform, however this was changed to with the release of version 1.1 so as to not conflict with the existing Inform programming language.

## 9.3 Continued development

After the initial release, Lineform was developed further, incorporating the feedback from many users. Initially the design and development was spurred onwards to meet user needs and in order to be submitted to the Apple Design Awards which celebrate “technical excellence, innovation, and outstanding achievement” and are the Mac software’s equivalent to the Oscars.

### 9.3.1 Apple Design Awards

Applications submitted to the design awards are judged in several different categories including: technology adoption, user interface design and innovation. In order to have a strong chance of winning, for the first three months after release Lineform’s development focused on incremental features or bug fixes in response to user feedback and a major version 1.1 which incorporated many Mac OS X technologies.

The main features that were specifically implemented and were aimed at winning these awards were: Core Image raster based filters, AppleScript support and Spotlight support. All these were specific Apple technologies that would improve Lineform’s success with the judges, and in fact these were all mentioned at the awards ceremony when Lineform won.



**Figure 9.3: iPod Nano (Karen Hughes, May 2006)**

Drawings were also solicited from all the early users to submit to the design awards. One of the drawings of an iPod Nano by Karen Hughes, Figure 9.3, was used to demo Lineform at the awards ceremony.

### 9.3.2 Initial user feedback

The initial feedback included many bug reports but was overwhelmingly positive. Here are a couple of examples:

But don't let all those bug reports fool you. I'm still deliriously happy with the program.

— Uli Kusterer (pril 2006)

I just downloaded Inform today and must say I'm impressed with your app. It's exactly what I've been looking for to help with some of my work.

— Jeff Hester (April 2006)

The incremental changes involved several bug fixes and performance improvements, but also several important features such as an outline view which enables non-WYSIWYG but easier editing of complex drawings and many more keyboard shortcuts.

Single-pixel horizontal or vertical lines drawn on a per-pixel grid suffer from being anti-aliased into two-pixel wide "blurry line" when exported (this is a big blocker for me) this looks like a rendering bug

— Dave Balmer (April 2006)

One of the mistakes made at this point, in order to please initial customers was implementing pixel aligned grids. Dave Balmer's feedback about a specific issue with anti-aliasing prompted the creation of the specific solution of a toggle that adjusted the grid position by half a pixel. This feature is



rarely useful and its use is obscure. Lineform's user interface would be simpler without it, using a different solution to anti-aliasing would be a better approach.

A status bar that provided contextual help and information was one of the main features that were implemented in version 1.1 in response to user feedback. Lineform makes use of several modifier keys to change the interaction mode, these *physical modes*, described further in Chapter 10, are used for many different interactions but they are invisible to the user. Users often needed the functionality but were confused about how to achieve what they wanted. The status bar provided feedback to the user about what modifier keys and modes are available, this allowed users to discover the functionality without reading the manual. An additional *physical mode* was also added to allow the user to control the scaling of the stroke size when scaling a drawing.

The initial method of rotating a shape was copied from CorelDraw. To toggle the rotation or transform modes the user clicked on a selected shape. To rotate a shape a user would select it, then click on it to toggle the mode then use the visible rotation handles to rotate the shape in place. This interaction was found to confuse users and was replaced with an alternate *physical mode* using the control key to toggle the rotation or transform mode. This made it quicker and easier to rotate shapes.

Other improvements such as group editing, clipping, SVG import and export were also added. SVG support was in response to the frequent requests for easier ways to get vector data in and out of the application.

Tools in version 1.0 had two modes: locked and unlocked, a locked tool would stay selected after use, an unlocked tool would switch back to the selection tool after use. The idea behind this was to enable a user to quickly draw a shape with one tool and to not be stuck in it, but also to allow a user to double click a tool to lock it so that they could also draw many shapes. After user feedback it was decided that this was additional complexity that did not lend the user interface any benefit, and it was removed and tools always locked for simplicity.

### 9.3.3 Commercialisation

Freeverse Inc who had expressed an interest in publishing Lineform acquired the publishing rights after version 1.1 was released. Freeverse did not change the development or design process of Lineform, they handled support and raised its profile by creating a drawing competition and selling boxed copies in retail stores. To provide a good release for them, several fixes and small improvements were fitted into the time before their release. A new transform inspector was added which allowed users to specify the exact dimensions of shapes. This was a feature that was requested often from users creating



exact drawings. This inspector was also designed to be *projected*, such that it provided live feedback as the user transformed shapes on the canvas.

Some user interface aspects which were not entirely *projected* were updated, for example the stroke width in an inspector now changed when transforming a shape on the canvas.

A new *physical mode* was also used for toggling the scaling of shapes when transformed on the canvas. This might seem slightly contrary but this allows some handy, if infrequently used interaction. For example dragging a shape with a image fill and toggling off the shape transformation means that the image fill is transformed and not the shape, this allows the fill to be positioned inside the containing shape.

#### 9.3.4 User feedback

The remainder of Lineform's development focused on adding features, often in response to user feedback. This section provides examples of user feedback and the features that came from it.

##### Implemented features

"I often have to manipulate elements of existing PDFs, such as graphs created by scientific programs like Matlab. Is it possible to do this in Inform?

— Arjun Raj (April 2006)

Some user feedback was directly helpful in prioritising features for Lineform. Several users requested PDF editing, both as a way to import graphics into Lineform and to edit existing graphics. PDF import as images was a feature from the start, PDF editing was added later after several requests.

The type tool doesn't seem to offer kerning and spacing adjustment.

— Roger Harris (April 2006)

Some users had problems kerning, or wrongly assumed that Lineform did not support kerning. The standard OS location for kerning was buried in a menu and it was decided that it would be easier to use and provide a better experience to add a specialised text inspector. This inspector provides continuous control using sliders of character spacing and other text attributes. The continuous interaction allows for much easier exploration and use.

##### Unimplemented features

Some features that users requested were just too complex and did not fit the goals for Lineform, these were features like animation or bitmap editing.

Is Lineform a potential replacement for the vector and bitmap creation/editing functions of Canvas?

— Bob (October 2006)

## Other applications

Many users wanted Lineform to work like their favourite drawing application. These desires were often in contention with each other and with the goal of making Lineform a streamlined and good vector drawing app. Here are a couple of examples:

I really hope that Lineform can become the new 'Canvas' and better.

— Vikingz (February 2007)

[I want Lineform to:] 1. look like Adobe Illustrator CS3.

2. feel like Adobe Illustrator CS3.

3. read Adobe Illustrator CS3 files natively.

— Lucius Kwok (August 2006)

I urge the developers to adopt the best conventions of CorelDraw

— Stokestack (October 2006)

The main response to these requests was to politely say that Lineform had its own design goals.

## 9.4 Flow principles

Lineform was designed within a context shaped by the calculator's design. From the very beginning, the calculator's flow-principles outlined in Chapter 4 affected the design of Lineform and were a continued source of guidance and inspiration. This section describes how these principles affected Lineform's design.

### 9.4.1 Projection

Chapter 8 mentioned the lack of immediacy that some drawing programs have, this is in direct opposition to the *projection* principle. A projected user interface is one which is always up-to-date and immediate. Situations like that of Figure 8.8 are caused by user interfaces that are not projected and provide many problems for users.

The lack of *projection* is especially a problem for drawing applications, because it severely hampers exploration and experimentation, which are critical components of most artistic creation. A non-projected user interface

forces a trial and error approach to exploration which is slow and error prone.

The flow principle of *projection* asserts that the user interface controls and the rendered drawing are different views of the same data. A lack of immediacy in updating the drawing, for whatever reason, creates an inconsistency between the user interface controls and the drawing in the canvas. Lineform keeps the user interface and the drawing in sync at all times. There is no inconsistency between the drawing and the user interface controls in the inspectors whether the user is modifying a control in an inspector or modifying the drawing on the canvas with direct manipulation.

Lineform was developed with the immediacy of *projection* in mind from the very beginning. The first release of Lineform (then called Inform) listed this immediacy as a selling point when describing the gradient feature: “Unlike some awkward interfaces for gradients, Inform provides excellent interactive tools. You can interactively set gradient stops from the inspectors including transparency. And from the document you can drag and alter the gradients in interactive real time.” This feature means that when the user changes a gradient, the user sees the result of their change immediately. This makes Lineform much faster to use than other applications where the user has to finish interacting before updates occur.

All visual updates happen immediately when interacting with the drawing in Lineform. Every change to a user interface control is immediately reflected in the drawing, there are no modal or delayed operations. Everything in Lineform is always up-to-date, everything the user does provides immediate feedback. There is not a single modal dialog box, which hinders the experience of *projection*, in the entire application. Modal dialog boxes often break *projection* and force the user to wait until the dialog box is closed before any action is taken and the user can see what they have actually done.

Not only does *projection* reduce the number of possibilities for confusing the user but it allows faster interaction and decisions about the creation of a drawing. *Projection* means that the user interface provides very useful feedback whilst manipulating the graphics. For example, when changing the line width of a stroke in an inspector it is possible to see the drawing updated immediately in the canvas. This allows the user to quickly get the exact value they want in one continuous interaction, without resorting to repeatedly trying different values.

Exploration is a critical component of drawing. The combination of projected editing and providing continuous interaction with user interface controls, are large factors in making it easy for the user to easily explore and enjoy drawing. These principles as part of Lineform’s design allow the user to very quickly see and explore the effect of a entire range of values and their effect on the drawing.

### 9.4.2 Continuity

Lineform provides a very different user interface to calculator. While the calculator often made state changes independent of the user, causing components of the mathematical expression to move around; Lineform does not make any state changes without the user initiating and controlling them.

As the user changes aspects of their drawing in Lineform they are directly manipulating the drawing and the feedback from that manipulation is *projected* and immediate. There are no real state changes made by computer, and thus no unexpected continuity errors. The immediate feedback which is smooth and fast provides the *continuity* or ‘morphing’ as the user interacts.

### 9.4.3 What You See Is What You Edit

With respect to how the user edits drawings, Lineform again provides a different sort of interaction to the calculator. As a drawing program the majority of user interaction is a direct interaction with the visible objects. Thus almost all drawing programs are WYSIWYE by default. They have no hidden state; the drawing is what the user edits.

There are cases of drawing programs ignoring this, for example Bézier nodes can be selected while they are not visible in Adobe Illustrator. The direct selection tool for editing Bézier nodes can be used to select invisible nodes on an unselected shape, the nodes appear as the user hovers over them. This is a case of the designers choosing a potential speed benefit over the visibility and clarity of what is editable. A user interaction where you interact on something that is unseen is by definition not WYSIWYE, and can be very confusing for users.

Vector graphic based drawings can also get very complicated. In fact there are innumerable ways in which the same drawing could be composed or created. Thus editing what you see can be complex, because what you see is not a one-to-one match with the underlying vector model. The user can not know what the underlying structure of shapes are from looking at the resulting image. A potential approach to this problem, one which Lineform does not use, is planar maps which provide a completely different but more WYSIWYE experience Baudelaire and Gangnet [1989].

For a drawing program both seeing what you get when you export or print the drawing (WYSIWYG) and editing what you see (WYSIWYE) are both very important. In order to aid both of these purposes Lineform provides a soft proof and outline view of the drawing. Soft proof supports WYSIWYG by rendering the drawing in the colour space of the printer, usually CMYK, this shows the colours and contrast of the drawing as they will be when printed. The outline view supports WYSIWYE by drawing every shape as a thin outline, this allows the user to better see the individual shapes and

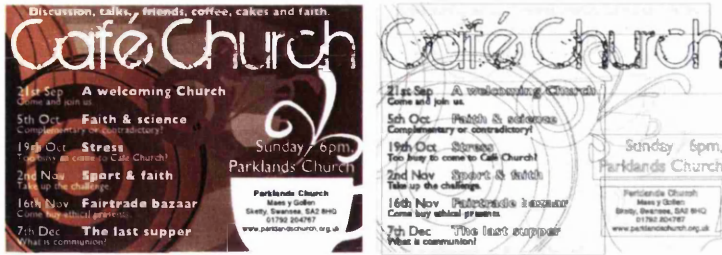


Figure 9.4: The outline view of a flyer

how they overlap. Figure 9.4 shows the drawing and outline view of the same flyer.

Features such as the outline view, the object tree in the Layers inspector and focus/isolation mode are designed to allow the user to more easily see and understand the vector data. These are designed to support a more WYSIWYE user interaction.

#### 9.4.4 Declarative interaction

The calculator provides a declarative interface to mathematical calculations. It is possible to write on both sides of the equality. The calculator then enforces the equality by correcting the side or sides of the expression that are not completely specified.

Lineform is declarative because there no distinction between ‘input’ and ‘output’ in part because of the nature of drawing programs. Any visible data is editable whether it is by direct manipulation on the canvas or by interacting with user interface controls in the inspectors. None of these representations are considered ‘output’ and cannot be edited.

Lineform allows the graphics to be edited from many different views. While there are no semantic “declarations” that Lineform enforces, it has very little distinction between input and output, which is what makes declarative interfaces easy to use.

The Transform inspector in Lineform is an example of a user interface control that could be considered primarily for input but supports both input and output equally. The Transform inspector can be used to enter exact dimensions for the selected graphics, and conversely when changing the size or position of graphics in the canvas it is possible to see the exact numerical values in the inspector updated continuously. This allows the user to use the Transform inspector as guidance when they are manipulating graphics.

## 9.5 Lineform principles

The calculator's flow principles provided an initial starting point and further refinements or new principles were developed alongside these as Lineform was created.

### 9.5.1 Physical modes

As the design got more complex, additional modes were needed and *physical modes* were used repeatedly. Physical modes provide quick and simple mode switching that reduces the user's cognitive burden and allows for a very fast exploration of what each mode achieves. As the design of Lineform evolved, physical modes were used in many different situations especially when short term modes that benefited from quick toggling were needed.

The rotation interaction started off as a non-physical mode that was toggled by clicking on the selection. After user feedback and continued use, this additional mode seemed to cause confusion and slowed users down. Here is one example of user feedback on the initial rotation interaction:

Make it easier to toggle between the square "grippies" and the oddly-shaped ones for rotating. I find myself clicking several times until I'm finally rid of the rotation grippies so I can resize an object. What about having a dedicated "resize" tool or menu item instead? Or you could also just have the regular grippies, and in some spots a special "rotate" grippie next to it or so.

— Uli Kusterer (April 2006)

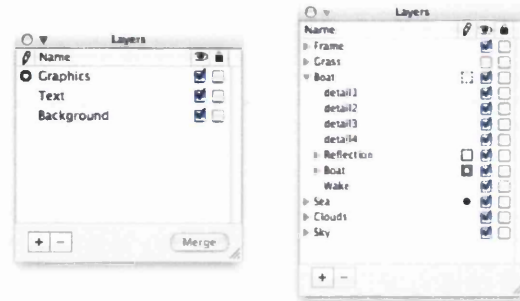
While none of the user's suggestions were taken, the underlying problem was solved in a way that streamlined the user interface and removed a confusing non-physical mode.

### 9.5.2 Flexible design

Lineform's flexible design was directly inspired by the rigid design of other drawing applications. Instead of providing rigid specialisations, such as a specific text or image object, Lineform was designed from the start to allow objects to be reused for different purposes. In Lineform there is no over specification, the original marketing blurb explained this as: "Fit text inside any odd shape. Inform does not constrain you to special text objects, but lets you put text inside any shape. Text is then flowed through the shape to create the designs you want."

An example of this in action is how groups and layers which perform similar roles evolved to provide the same capabilities and in fact are now exactly the same. The original 1.0 release of Inform had a completely distinct concepts

of groups and layers. Layers could be named, locked and made invisible from the Layers inspector; groups on the other hand could be resized and directly manipulated on the canvas.



**Figure 9.5:** The evolution of the Layers inspector

Figure 9.5 shows the evolution of the Layers inspector from left to right. The latest layers inspector allows every graphic to be manipulated just as the layers were in version 1.0, they can all be named, locked and made invisible. Groups and layers became identical and removed the arbitrary restrictions the distinction placed on the user. The new version, without the distinction, allows users to: apply an opacity or filters to layers, to select and resize layers, to name, move and lock groups or other graphics, to drag rearrange groups, layers and graphics (including dragging ‘layers’ into groups or groups out of ‘layers’).

### 9.5.3 Appropriate controls

Appropriate controls sums up the idea that continuous values should be controlled through continuous interaction. For example the width of a line is controlled using a slider. Combined with *projected editing* this allows the user to quickly and easily explore a range of values.

This was a core principle from the very start of Lineform’s design. Almost every continuous value is controlled through either direct manipulation or through a continuous control such as a slider. Both user feedback and continued use refined this initial principle into ensuring that continuous values have both a continuous control for exploration and quickly setting rough values easily, and a discrete control for setting exact values and adjusting rough values.

Later in version 1.5, Lineform added a Transform inspector that provided the discrete, exact controls to complement the continuous direct manipulation of transforming graphics on the canvas. The Transform inspector provides discrete numerical value entry for the size, shape and rotation of graphics which are extremely useful for setting the size of graphics to exact values.

Version 1.5 also added a Text inspector that provided continuous controls for aspects of text layout that were originally controlled through discrete menu items. The Text inspector provides continuous setting of character and line spacing and baseline position, these are *projected* and allow much easier exploration and setting of values than the discrete menus did.

## 9.6 Summary

Lineform was dually inspired by the success of the calculator and the failings of the existing drawing applications. The problems seen with other applications outlined in the previous chapter include a lack of direct manipulation, lots of modes, rigid design, over complication, lack of immediacy and a lack of features. Lineform was designed specifically avoiding these flaws seen in other programs and building on the principles that developed along with the calculator.

The design of Lineform once publicly available stretched to meet the needs of the many users while also attempting to retain the core goal of being a streamlined, focused and much easier to use approach to vector drawing. Since its initial release Lineform has continued to be developed and incorporates user feedback when it meets the core goal.

The calculator flow principles: projection, continuity, WYSIWY and declarative interaction; are also an important part of Lineform's design. The immediacy of projected editing means that exploration, a large part of drawing, is much faster. Continuity is maintained through no state changes that the user is not directly controlling and the speed and immediacy of the interaction. WYSIWY is partially supported by providing different views of the drawing, like focusing and the outline view, that provide a clearer view of the underlying data. Declarative two-way editing on the canvas and in inspectors allows users to switch repeatedly and iterate between the different views, so they can edit in the view that best suits their purpose.



# Chapter 10

## Principles

This chapter mirrors in part, Chapter 8's criticism of bad vector graphics drawing user interfaces. The areas of bad user interface design that were identified are used to show how Lineform avoids the same failings. How the flow principles from the calculator affected Lineform design was discussed in Chapter 9. This chapter describes the new concepts and principles that were important and solidified during Lineform's design.

### 10.1 Physical modes

Mode errors [Norman, 1981], such as drawing with the wrong tool, originally defined as what happen when the user misclassifies a situation resulting in actions which are appropriate for the user's interpretation but not the true situation.

Studies [Sellen et al., 1992] have shown that both visual and kinaesthetic (by physically pressing a key or a foot pedal) feedback can significantly reduce mode errors. Kinaesthetic feedback has been shown to be more effective than other forms of feedback (e.g. visual or audible) in reducing errors and reducing the cognitive load of mode changes. Kinaesthetic user-maintained modes, that is modes which are maintained by a continuous physical user action such as holding a key down, provide feedback to the user that is hard to ignore or forget. Raskin [2000] uses the phrase *quasimode* to label the same concept.

There is a limit on the number of different interactions that a user interface can provide without overloading the interface and the user with different methods of interaction, so using modes allows a user interface to provide more functionality by allowing single interactions to have multiple functions in different modes. This often leads to a profusion of modes in complex user interfaces like drawing applications and the numerous modes and tools of drawing applications was one of the design flaws outlined in Chapter 8.

The primary problem with modes in a user interface is their virtuality. That is, the current mode is often forgotten by the user, even if it is visually represented on the screen. Modes have a large impact on what the user's interaction does, but they rarely have a high visibility or awareness. This leads to mode errors, where the user expects a different result from their interaction than the current mode provides.

Instead of "kinaesthetically user-maintained", *physical modes* is the term used here to describe the concept of the user controlling interaction modes by physical action. Most human sensory inputs operate such that the awareness of constant stimulus decreases over time, which continues until there is no awareness of the stimulus at all. In comparison the awareness of muscle action to produce a force does not fade with time [Raskin, 2000]. This helps explain the greater awareness of modes caused by muscle action and the corresponding reduction of mode errors. It is much harder to forget what the current mode is if body is physically part of the interaction that causes the mode. This sense of the body's location and positioning is also called proprioception.

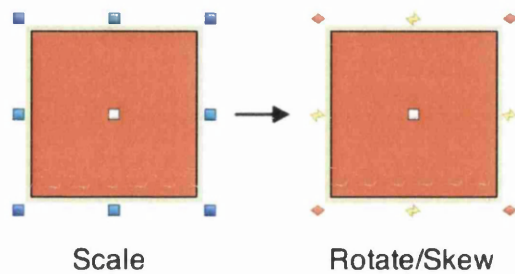
Use of *physical modes* is not a new concept, for example many user interface use the Shift key to toggle a mode where typed letters are capitalised or objects are added to a selection and the Alt key on a Mac is often used to toggle the copying of dragged items. Consider how natural these modes become, even though there is no other representation of the mode except for the physical act of holding down a key, there is very little confusion over which mode the user interface is in. Capitalising letters using the Shift key comes very naturally to computer users, however the Caps-lock key which provides the same functionality, the same mode, but in a different form causes confusion. Because the Caps-lock key is not user-maintained, or *physical*, the awareness of the mode is reduced, and even though the Caps-lock key has its own toggle light on the keyboard Caps-lock sometimes cause even experienced users confusion. Most experienced computer users rarely use the Caps-lock key because of the increased cognitive and physical effort of using it.

Of course for long term modes, interaction modes that a user interface is used in for extended periods of time, a *physical mode* does not always make sense. Holding down the Shift key to type a paragraph or even a heading in capital letters could get physically tiring.

Some user interfaces such as SMARTboards and Wacom tablets provide real physical modes, where different physical tools perform different functions. For example a SMARTboard has three different coloured pens and an eraser. The SMARTboard recognises when these are removed from their holding trays and triggers interaction modes that draw the correct colour on the screen or erase parts of the drawing. Which tool the user is using is physically dependant on which tool they are holding.

### 10.1.1 Lineform

The primary interaction with Lineform is through the mouse, but Lineform makes widespread use of physical modes to alter the effect of this interaction. Lineform uses several modifier keys: Shift, Control, Alt, Command, Z, and ~. Each of these keys when held down affects how the user interface interacts at the moment.



**Figure 10.1:** Rotation modes in Lineform, toggled by the control key

Figure 10.1 shows two modes for modifying shapes in Lineform: when the Control key is held down the current control handles that surround a selection change from resizing to rotation handles. Here the mode provides both visual feedback, the handles change appearance, and physical (or kinesthetic) feedback of the user physically holding down the Control key. Because of the physicality of the mode it is almost impossible to forget that the system is in the rotation mode.

The modifier keys augment the current tool mode in Lineform, affecting the action the user is currently performing. Different modifier keys are valid in different situations but their function is consistent, for example the Shift key toggles the restriction of transforms, when rotating a restriction of  $15^\circ$  angles, when scaling a restriction of a constant aspect ratio and when dragging a restriction to horizontal or vertical movement. Table 10.1 shows a summary of the physical modes used in Lineform and a description of their function.

Figure 11.5 in Chapter 11 shows how toggling the fill and the stroke mode affects the scaling of an image in a box. These two modes, scaling the fill and the stroke, are physical modes, and they are toggled by holding down the Command and Z key respectively. These modes can be toggled independently of each other and the user's actions. As the user transforms a graphic they can toggle these modes off and on, instantly seeing the changes each mode affects.

Physical key	Mode function
Shift	Constrains modifications to a restricted set of values, like 15 deg angles or rational fraction scales.
Control	Toggles between scaling and rotation
Alt	Transforms the centre of the modification to the centre of the selection.
Command	Toggles whether the fill style of the selection will be modified.
Z	Toggles whether the stroke style of the selection will be modified.
~	Toggles whether the shape of the selection will be modified.

Table 10.1: The physical modes used in Lineform

### 10.1.2 Recall

Physical modes such as using the Shift key to capitalise letters becomes very natural and are remembered because they are used consistently across most user interfaces. In this way they also utilise “muscle memory” and become almost automatic.

However, in a new user interface, such as Lineform; where the modes are different, knowing which keys do what is not obvious. Therefore to provide the user with a clear knowledge of the currently applicable modes and keys Lineform provides a status bar just below the toolbar in each document window. The status bar for various different tools is shown in Figure 10.2, the highlighted status bar is shown when using the selection tool. This status bar shows the current scale of the selection and the current modifier keys that affect the transform, such as Shift to constrain the transform. The changes to how the shapes are modified are immediately shown when any modifier keys are pressed, this *projection* allows very quick correction of mistakes and a comprehension of what each mode does.

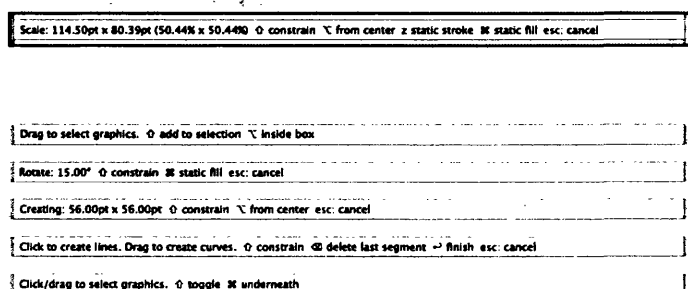


Figure 10.2: The status bar in Lineform

### 10.1.3 Disadvantages

A disadvantage of using modifier keys for *physical modes* is that the user usually needs two hands, this is especially true when using the mouse as the main tool of interaction, which Lineform does. Users that are unable to use both hands are unable to access these modes. This is a common problem and Mac OS X provides a universal access feature called *sticky keys* that can treat the standard modifier keys as toggles. One press of a modifier key and it is temporarily held down until a different key is pressed, if the modifier key is pressed twice then the modifier key is kept held down, a third press and the modifier key is toggled off. These different modes are accompanied by both a typewriter sound and an on screen depiction of which keys are held down.

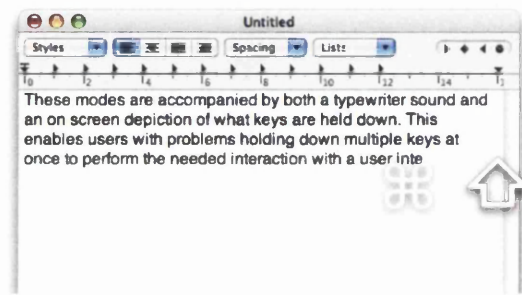


Figure 10.3: Sticky keys in Mac OS X

Figure 10.3 shows the on-screen visual reminder of the current modifier key mode provided by sticky keys, in the figure the Shift key is permanently held down and the Command key is temporarily held down. The next non-modifier key press will be typed with the Shift and Command modifier keys, after that non-modifier keys will be typed with just the Shift modifier key. Sticky keys enables users that have difficulty holding down multiple keys at once to perform interaction with a user interface as if they were holding down all the keys at the same time. However despite both the audible and visual reminder of the current mode it can be very confusing.

### 10.1.4 Key concepts

*Physical modes* offer a very simple and effective way of providing short term modes in a user interface. They allow very quick toggling and are very hard to forget.

- Quick — Modes that are short term or need to be toggled quickly should be physical.
- Complementary — Physical modes complement another interaction, they should be secondary in effect and simple to toggle, not distracting from the main interaction.

- Projected — The changes a mode affects should be immediately visible.

## 10.2 Flexible design

Green's [1989] cognitive dimensions describe several heuristics which can be used for design or evaluation. Two of these related to flexibility: *viscosity* and *premature commitment*, are useful for describing flexible design. Viscosity describes how much effort is required to affect change in a program and premature commitment is when the user is forced to make unreversible decisions before they want to.

A design goal for Lineform was to provide a user interface that was both flexible and that did not constrain the user by over-specialising certain types of graphics. In Lineform there are very few different types of graphic and it is very easy to convert between them. This was directly inspired by the rigid and restricting design of some drawing applications, such as Adobe Illustrator.

Lineform does not have specialised graphics types like: text-boxes, text-on-a-line, images, spirals or graphs. The more complex shapes such as graphs or spirals can be constructed from the simpler shapes, and the other graphic types like text or images are provided as styles which can be applied to any shape.

In Lineform it is possible to draw any shape and fill it with text or an image. In contrast, in most drawing programs, images and text are provided by using specialised types of graphic. Both approaches allow the same drawings to be created. However, Lineform's approach does not pre-specify a graphic's function at its creation, they can easily be used for any kind of purpose. Applications such as Illustrator, that provide lots of specialised graphic types, make changing the function of a graphic awkward, and often have different user interfaces for each different specialisation. Additionally the extra controls and inspectors that these require can make the applications both harder to learn and use.

Groups are also used for both layers and on canvas groups, there is no specialised layer class. This means that layers provide all the same powerful operations as groups, for example, just like a group, layers can have transparency, raster effects, or be edited in isolation mode (this hides the rest of the document so the user can focus on a group).

Thus there are fewer special concepts in Lineform's user interface and it is therefore simpler and more coherent. This design also means that the user is more free to work with their preferred process, fewer aspects of the drawing are pre-specified unnecessarily restricting the user.

One aspect of WYSIWYE is that what things look like should determine

what they are and how they can be interacted with. In many drawing programs there are specialised object types that look identical but have a specific role that is fixed and unchangeable. The over-specification of the role of an object not only forces the user to make a decision before they want to, but also hinders WYSIWYE by treating objects that look similar in vastly different ways.

### 10.2.1 Key concepts

Viscous or rigid user interfaces increase the amount of barriers a user has to overcome in order to affect change. In contrast flexible user interfaces allow users to delay decisions until they are ready and to easily change their mind, providing a much more enjoyable user experience.

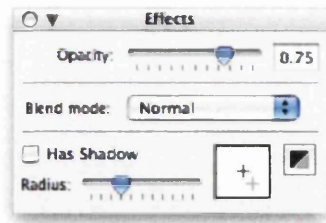
- Flexible — Allow the user to change their mind whenever they want. Enabling change facilitates exploration.
- Deferred — Do not force the user to decide anything before they need to. Unnecessary, early or premature specification restricts the user needlessly.

## 10.3 Appropriate controls

The underlying values in a user interface, the user interface control types and the user's interaction can all be discrete or continuous. Matching the correct control, value and interaction is important for providing the right experience.

Discrete values are those that are individually distinct, they do not have an ordering or have a finite number of states between two values. Examples of discrete values are integers or types of fruit. Continuous values are those where there is a smooth and unbroken progression between any two values. Examples of continuous values are real numbers or temperature.

Correspondingly, user interfaces have controls for discrete and continuous values. Check-boxes, pop-up menus and selection in tables are all discrete, whereas sliders and directly manipulatable two dimensional position controls are continuous. Some controls such as text boxes can specify both discrete or continuous values. However both types of user interface control can still offer discrete or continuous interaction. Discrete interaction is a stop-start affair where the value is only set once the user has finished editing. Continuous interaction is a smooth *projected* experience where the value is continually adjusted whilst the user interacts. Without *projection* or continuous interaction even a continuous control exhibits discrete behaviour and hinders the user exploration and ease of use.



**Figure 10.4:** User interface controls in Lineform's Effects inspector.

The user interface in Lineform is primarily composed of the canvas and the inspectors. The inspectors are mostly made up from standard user interface controls. Although interaction with the graphics is indirect, the interaction with the controls is direct. Figure 10.4 shows Lineform's Effects inspector which controls the opacity, blend mode and the drop shadow of the selected graphics. The continuous variables (opacity and drop shadow radius) are controlled through sliders which are continuous user interface controls. The position of the drop shadow is similarly controlled through the custom user interface control that looks like a white box, where dragging on the box changes the offset of the drop shadow as a continuous interaction. The discrete attributes, such as blend mode and shadow, are controlled through the check-box and pop-up button which are discrete controls.

This is a result of a principle informing the design in an unanticipated way. Only in retrospect, in clarifying the principle, does the potential improvement to Lineform's design become apparent.

A continuous discrete user interface control is possible when there is a live preview of the result during the interaction. An example would be if the blend mode pop-up menu showed the results of the selection whilst the user scrolled through the menu. Although Lineform does not do this now for any discrete controls, it would be a big improvement especially for the blend modes. Lineform has 13 different blend modes, some of which can have unusual results, therefore it is often the case in the current implementation for the user to repeatedly click on the pop-up menu and select the modes in turn in order find the right blend mode. In a continuous implementation the user could click the pop-up menu then just scroll down through the items seeing the results as each item is hovered over in turn, this would result in a vastly faster and easier interaction.

Providing continuous interaction and control for a continuous attribute makes it easier and more natural to change the attributes. A continuous control that is *projected* and provides live updates allows the user to quickly explore a large range of possible values. Of course, being able to specify the exact values for these attributes is still valuable, so Lineform often provides both types of control and interaction. In Figure 10.4 both a slider and a text-box



are provided for the opacity attribute of the graphics.

### 10.3.1 Key concepts

The correct user interface controls for the right values makes a huge difference. The right control can enable the user to manipulate it's value quickly and easily.

- Discrete — Discrete values should be controlled through discrete controls.
- Continuous — Continuous values should be controlled through continuous controls.
- Exact — Continuous values should also have an exact way of setting values.
- Continuous interaction — Every control should, where possible, provide continuous *projected* interaction.

## 10.4 Other principles

These are some of the other principles that were important during Lineform's development.

### 10.4.1 Direct manipulation

Direct manipulation [Shneiderman, 1983] is a standard user interface design principle, but one that is also critical to Lineform's user interface. Lineform provides manipulation controls on the actual shapes as often as possible, instead of in other distinct user interface components. For example it is possible to directly grab the corner of a selection and rotate the selected graphics. Manipulating the graphics is done by 'interacting' with the graphics, this provides a much easier and natural control of the graphics than a separate user interface control.

Unfortunately there are many complex aspects to vector drawings that do not easily allow for an obvious direct manipulation. For example, it is not obvious how to directly specify the stroke dashes of a graphic by interacting with the graphic on the canvas. In fact most of the attributes of the visual style of graphics have the same problem. In general the geometric shape and position of graphics lend themselves to direct manipulation, but their visual style and appearance do not. Lineform manipulates the shape and appearance of objects in different user interfaces that best control those aspects, this is further described in Section 10.4.3.

### 10.4.2 Simplicity

Lineform seems simple, but the more you look, the more features there are. That's a nice feat that Adobe and others don't seem to be able to pull off.

— Dylan (April 2007)

Part of the motivation for having few specialised graphical types in Lineform is the motivation of keeping it simple. This desire was driven by the over complication of user interfaces. The fewer concepts and the more coherent those concepts are; the easier and simpler the user interface should be to learn and use.

The design of Lineform tries to minimise the number of specialised tools, modes and concepts. The aim was not to overload individual modes of operation but to reduce the complexity of Lineform's user interface.

Lineform lacks many of the numerous "complex" features of programs such as Adobe Illustrator. However the feature set of Lineform probably provides almost all of what novice users require and a large proportion of what professional and amateur users want.

Many of the complex features of Illustrator can be replicated with a couple of extra steps in Lineform. The benefit of a simpler and streamlined user interface outweighs the cost of needing extra steps to achieve complex effects.

### 10.4.3 Well defined roles

Lineform provides a clear distinction between the function of different aspects of its user interface. The main two components of this are the document canvas and the inspectors. The inspectors always and instantly show the current state of the selection, the document always and instantly shows the current drawing. They do not do anything else. The well defined roles of the different parts of the user interface contribute to Lineform's ease of use.

There are no dialog boxes or other modal user interface components in Lineform. All the different aspects of a drawing are presented to the user through the inspectors which can be visible all the time and are always up-to-date, providing *projected* editing. The inspectors have a single function, which is showing and modifying aspects of the current selection or document. The inspectors do not provide any non-reversible actions, undo is never needed to change a value in an inspector back to its original value.

Manipulating the shape of graphics is performed on the canvas and additional actions such as Boolean operations and aligning graphics are available through menus, short-cut keys and toolbar buttons.

## 10.5 Summary

Lineform avoids many of the issues described in Chapter 8 by providing a straightforward, flexible user interface to drawing. The consistency and overall simplicity of the user interface design are one of the key qualities of Lineform.

The primary principles that are important to Lineform's user interface design include: physical modes, having a flexible design and using appropriate controls. The use of physical modes simplifies and makes mode switching within tools simpler and quicker. By using physical modes the user can very quickly switch modes within a tool whilst they are performing any interaction. Lineform's *flexible design* allows users to not worry about restrictions from premature commitments, this allows users to not worry about future changes of mind while they want to be creative. Using the *appropriate controls* for the right underlying values means that the values are both easy to set and to explore using continuous *projected* interaction.

Other principles such as direct manipulation and the simplicity of the user interface also were important during Lineform's development.

# Chapter 11

## User interface overview

### 11.1 The interface

Lineform's interface revolves around a document window that displays the canvas and objects being edited. Objects on the canvas are manipulated by the mouse or keyboard and can be altered from the inspectors that float in front of the document window.

#### 11.1.1 The toolbar

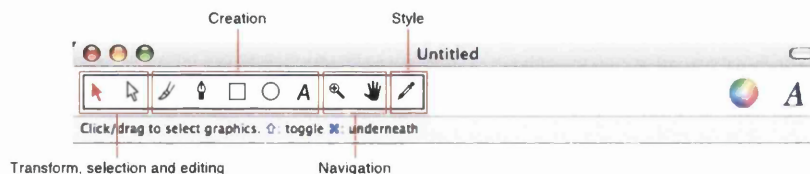


Figure 11.1: The toolbar

Each document in Lineform has its own toolbar at the top of the window. This toolbar shows which tool is currently being used and allows quick access to other tools. These are:

- Selection — Selects and moves objects.
- Editing — Alters curves and lines.
- Brush — Draws smooth lines.
- Pen — Draws lines and Bézier curves.
- Rectangle — Draws rectangles and squares.
- Oval — Draws ovals and circles.
- Text — Draws text boxes.

- Zoom — Zooms in and out of the canvas.
- Drag — Moves around in the canvas.
- Dropper — Picks up styles from objects in the canvas.

The tools can also be selected by pressing the digits 0-9 or their corresponding shortcut keys. The toolbar can be customised to contain several handy functions, such as combining and Boolean operations. To customise the toolbar, choose *Customise Toolbar...* from the View menu.

### 11.1.2 The status bar

**Click/drag to select graphics. ⌘: toggle ⌘: underneath**

**Figure 11.2: The status bar**

Below the toolbar is the status bar. This shows hints and information about what action the user is currently performing. It shows the possible modifier keys for any action in blue and it shows the current size or state of any transformation of objects being modified.

### 11.1.3 Inspectors

Lineform uses inspectors that float above all the other windows. These provide access to object and canvas properties. The inspectors can be minimised or hidden and snap to each other. Inspectors can be toggled on and off in the Inspectors menu.

The Fill, Stroke, and Effects inspectors affect the appearance of objects in the document. The Grid, Layout, and Layers inspectors affect the canvas itself. Finally, the Filters inspector provides advanced options to apply effects to objects.

### 11.1.4 The media browser

The media browser is accessed from the Inspectors menu. It provides access to images and pictures stored in iPhoto or elsewhere on the user's computer. The user can also drag images into any Lineform document.

### 11.1.5 Keyboard

Several keyboard shortcuts and modifiers are used in Lineform, many of the shortcut keys were copied from Adobe Illustrator for consistency. These are

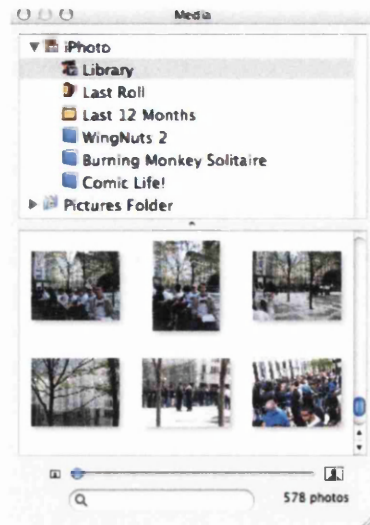


Figure 11.3: The media browser

mentioned in this chapter where they are relevant. The Shift, Alt, and Command keys affect most operations and the Escape key cancels the current action.

## 11.2 Manipulating the canvas

### 11.2.1 Zoom

Clicking with the zoom tool magnifies the canvas by 200%, and Alt-clicking zooms back out by 50%. The user can zoom in on a particular area of the canvas by clicking and dragging the zoom tool around the area of interest.

The canvas can also be magnified to a specific zoom level by selecting a level from the pop-up menu in the bottom right of the document window, or zoomed to fit the current graphics from the View menu.

Pressing Control-Space once selects the zoom tool temporarily, allowing the user to zoom in quickly on part of the canvas. After zooming, the tool reverts to the previously selected tool.

### 11.2.2 Drag

The drag tool is used to move around the canvas. To move the visible part of the canvas, click and drag with this tool.

The drag tool can also be temporarily selected by holding down the Space bar.

## 11.3 Creating graphics

The brush, pen, rectangle, oval and text tools are used to create line and shape graphics. When one of these tools is selected, clicking and dragging on the canvas creates new shapes. Newly created graphics use the current style from the inspectors.

### 11.3.1 The brush tool

The brush tool is used to draw smooth curves and paths and is ideal for drawing smooth arbitrary shapes and tracing pictures. The paths the brush tool generates are automatically smoothed and if a more accurate path is required the Alt key can be held to limit the smoothing.

### 11.3.2 The pen tool

The pen (Bézier) tool creates a sharp node for each click, building an arbitrary shape out of a series of nodes. Clicking and dragging with the pen tool create smooth nodes and curves. Clicking creates a node at the initial click point then dragging specifies the control point positions. The curve is updated live as the user draws ensuring it is easy to see what the result of the action will be.

Double-clicking on the canvas or on a node completes the current path, and Escape cancels it. Pressing the Delete key cancels the last segment of an unfinished line allowing the user to go back and redo parts of the path being drawn. The Shift key limits the line and curve drawing to 15° angles.

Clicking on the initial node closes the path being drawn, creating a closed shape.

Both the pen tool and the brush tool allow new paths to be drawn starting from either the beginning or end of existing paths. When either tool is selected, nodes appear at the ends of selected paths that are available to append to. Drawing from one of these nodes automatically appends the new path to the existing path. This allows both tools to be used to draw different parts of the same shape.

### 11.3.3 The rectangle and oval tool

The rectangle and oval tools draw their respective shapes. Holding the Shift key restricts the new shape to a 1:1 aspect ratio, thus creating either squares or circles. The Alt key allows the creation of shapes from the centre. These modifier keys are exactly the same as the modifier keys for resizing shapes.

### 11.3.4 The text tool

The text tool can be used in two modes: it can be dragged to create a text region, or used by simply clicking and then typing, the region created will be the same size as the text typed. The text tool does not create a special object: it is just a handy shortcut, the same effect can be achieved by creating a rectangle with a text fill style.

## 11.4 Manipulating graphics

### 11.4.1 Selecting graphics

Graphics are selected using the selection tool to click on them. Multiple graphics can be selected by Shift-clicking or by dragging a selection around several graphics. The selection tool starts a drag-selection when clicking and holding down the mouse button in an empty part of the canvas. Graphics are added to the current selection by holding the Shift key while selecting more graphics. Graphics can be deselected by Shift-clicking on them after they have been selected.

Graphics with no fill can be selected by clicking on the stroke line or by drag-selecting them.

Selected graphics are drawn with a highlighted thin line around them. The primary object is drawn with a green highlight and all other selected graphics are drawn with a blue highlight. Pressing the Tab key changes the primary graphic cycling through all the currently selected graphics.

The primary graphic is the graphic displayed in the inspectors when more than one graphic is selected. Operations such as Boolean subtract or align, are performed relative to the primary graphic. For example, aligning left edges will always align the edges of the selected graphics to the left edge of the primary graphic.

By holding down the Alt key, only graphics that are entirely within the selection region whilst dragging are selected. This is useful for selecting small graphics or graphics that are on top of each other.

All the graphics on the canvas can be selected with *Select All* from the Edit menu, or deselected with *Select None*.

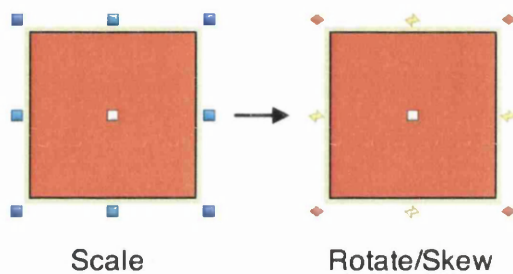
Often one graphic will obscure other graphics hidden below, these are normally tricky to select. Clicking with the Command key held down selects the next graphic beneath the click and can be used to select an obscured graphic by repeatedly clicking on top until the desired graphic is selected. The first Command-click selects the top object like a regular click, then each subsequent Command-click will select the object below the current selection.



Several Command-clicks will cycle through the objects below the mouse in order, eventually returning to the topmost object.

### 11.4.2 Moving, scaling and rotating

Once some graphics have been selected, nine handles appear around the selection. These handles manipulate the selection allowing resizing and scaling. The Control key toggles the mode from scale to rotate/skew, the handles change appearance to reflect their use, this is shown in Figure 11.4.



**Figure 11.4:** Resize and rotate modes in Lineform, toggled by the control key

Using the resize handles it is possible to:

- Move graphics by dragging on them or by dragging the white centre drag handle. Constrain the movement vertically or horizontally by holding the Shift key.
- Create a copy of the graphics by holding the Alt key while dragging.
- Scale the graphics by dragging any of the blue and green edge handles. The blue corner handles scale the graphics both horizontally and vertically. The green edge handles scale only in one axis. The Shift key constrains the corner scaling to preserve the original height/width aspect ratio. The Alt key transforms the graphics from the centre.

The Control key changes the handles of the selected graphics so that it is now possible to:

- Rotate the graphics by dragging the orange corner handles. Holding the Shift key down constrains the rotation to 15 angles.
- Skew the graphics by dragging the yellow edge handles. Holding the Shift key down constrains the skewing to 15 angles.

While manipulating objects, holding down the Command key stops their fill from changing. This allows the user to clip pictures easily because resizing scales the shape but not the picture. Command-dragging also allows the size of a text box to be increased without changing the size of the text.

In a similar manner, holding the Z key down while manipulating objects stops the stroke style of the objects from changing size. By default when resizing graphics the stroke width changes correspondingly, the Z key toggles this so that graphics can be resized whilst retaining their original stroke width.

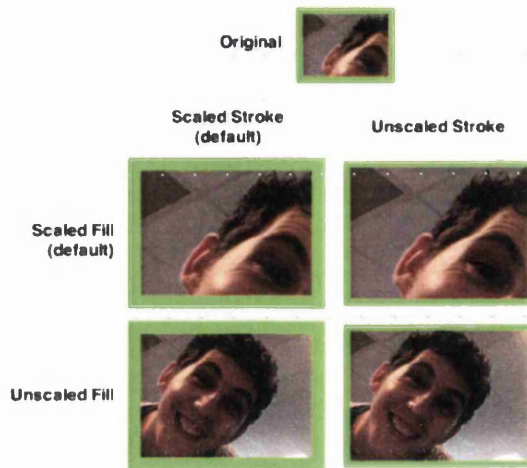


Figure 11.5: Scale modes in Lineform

Figure 11.5 shows the difference between scaling the fill or stroke, these can be toggled live whilst the user is resizing the graphics.

### 11.4.3 Transform inspector

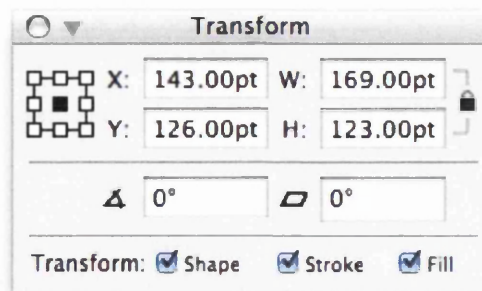


Figure 11.6: The transform inspector

The Transform inspector, shown in Figure 11.6, makes it easy to manipulate the width, height, layout, rotation and skew of graphics. The grid control in the upper left of the inspector sets the origin of the transformation. The Shape, Stroke and Fill checkboxes enable or disable transformations for the

graphic's respective components. The Transform inspector can also be used to rotate or skew graphics.

#### 11.4.4 Transforming with the keyboard

The arrow keys can be used to nudge objects. Graphics are moved 1 pixel by default or 10 pixels if the Shift key is held down. Graphics can also be nudged at the current zoom level by holding down the Alt key.

#### 11.4.5 Align and distribute

A graphic can be aligned both horizontally and vertically with the edges and centres of other graphics. Commands for alignment can be found in the Objects menu. The selected graphics are aligned relative to the primary selected object.

Graphics can also be distributed horizontally and vertically from the Objects menu. The distribute commands space the selected graphics out so that the size of the gaps between the graphics are the same.

#### 11.4.6 Flip

Graphics can be flipped around their centre horizontally or vertically by using *Flip* from the Objects menu.

### 11.5 Layers and Z-order

Z-order refers to the order in which objects are drawn, i.e., which objects are drawn on top of others. The ordering of objects can be changed with the *Bring to Front* and *Send to Back* commands in the Objects menu. These move the selected objects all the way to the bottom or top of the stack.

Objects can also be raised and lowered one step at a time with the *Bring Forward* and *Send Backward* commands. These will move objects up or down, respectively, as long as the next layer is visible and unlocked.

Each layer contains its own graphics. The layers also have their own z-order, and can be rearranged in the Layer inspector by dragging. Each layer can be toggled visible/invisible and locked/unlocked using the checkboxes in the last two columns. Hiding individual layers sometimes makes it easier to concentrate on the remaining visible layers. Locked layers are visible but not editable, which is useful when editing objects stacked on top of each other.

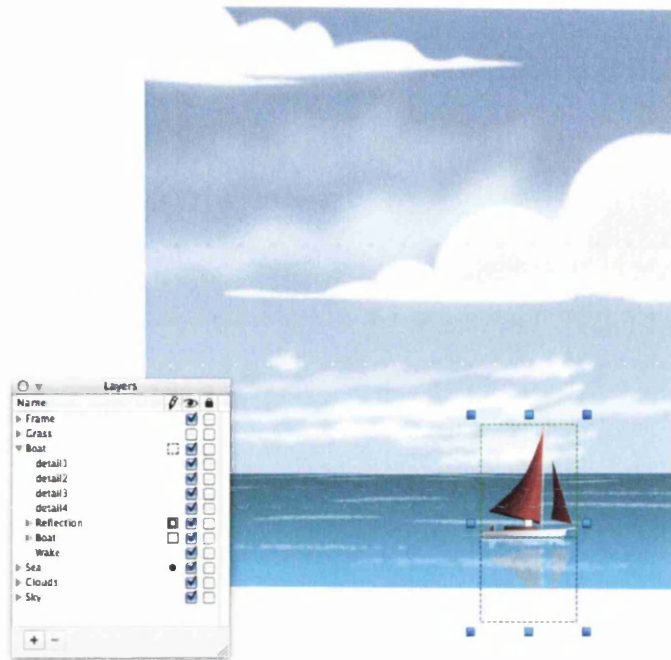


Figure 11.7: The layer inspector

The third column from the right, represented with a pencil, shows the editing status. A black circle highlights the currently active layer; all new objects will be created in this layer. To make a layer the active layer, double-click in editing status column of that layer. Selected shapes are shown by a square, the primary selection is shown by a double square and groups or layers with selected shapes inside them are shown as dashed squares.

Graphics are selected in the layer inspector by clicking in the editing status column of the graphic. The same modifier keys used on the canvas affect selection in the inspector: Shift adds graphics to the selection and Command toggles the selection of specific graphics.

Layers are created and deleted by the + and – buttons, and can be merged together with the Merge button in the Layers inspector. Layers are named by double clicking their names and editing.

### 11.5.1 Isolation mode

Isolation mode is a mode which dims out all graphics that are outside of the current group when editing a group. This allows the user to concentrate on the graphics inside the group without any other graphics getting in the way.

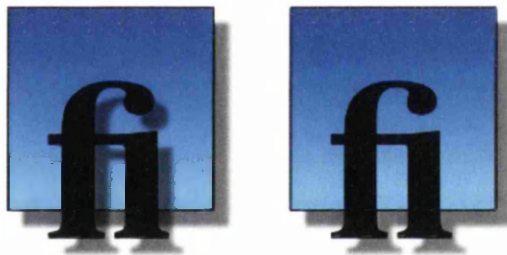
When editing groups, isolation mode is the default, but it can be toggled

on and off from the View menu. In isolation mode only graphics inside the current group can be edited and all other graphics are drawn dimmed out behind the current group. If isolation mode is off, then what the user sees is the same as if they were not editing a group and the user can select any graphic. Editing layers works by default with isolation mode off, but it is possible to switch it on to focus only on one layer.

## 11.6 Groups and combining

Several graphics can be grouped into a single group which behaves like a single graphic when it is manipulated. Grouping graphics is a useful way to keep graphics together, for example keeping text positioned over a background image.

Graphics can be grouped together with the *Group* command in the Object menu. This creates a new group from the selected objects. The *Ungroup* command reverses the grouping of selected groups, separating out their components.



**Figure 11.8:** A drop shadow applied to individual shapes and a group

A group can be edited easily by double-clicking the group or clicking the *Edit Group...* button in the Fill inspector. When a group is being edited, double-click outside the group or select a layer to finish editing the group. When a group is being edited it is displayed without any effects applied to it and everything else is dimmed out.

Groups do not have stroke or fill styles as with other objects but they can have effects. Thus it is possible to use a group to combine several objects together before a drop shadow or applying a transparency. The left of Figure 11.8 shows two objects that have a drop shadow effect, and the result looks ugly because the drop shadows are drawn separately on top of each other. On the right of Figure 11.8 the objects are grouped together and the group has a drop shadow effect, this results in one single drop shadow for the combined objects.

### 11.6.1 Clipping

Groups are also useful for clipping graphics, clipped graphics are only drawn inside the clipping region. Graphics are clipped by enabling *Clip to Top Path* in the Fill inspector. With this enabled all the graphics inside a group are clipped to the topmost path in the group. If the topmost object has a text fill then the group is clipped to the bounds of the text.



Figure 11.9: Using text to clip a group

Clipping to text in this way enables complex fills on text that is still editable. For example Figure 11.9 shows how grouping some text with a gradient and transparent squiggle beneath it, then clipping the group results in a complex logo where the text remains editable within the group.

### 11.6.2 Combining

Graphics within groups retain their separate styles and shapes. A different way of combining graphics together is to use the *Combine* command. Graphics combined together are merged into a single graphic, losing their separate style.

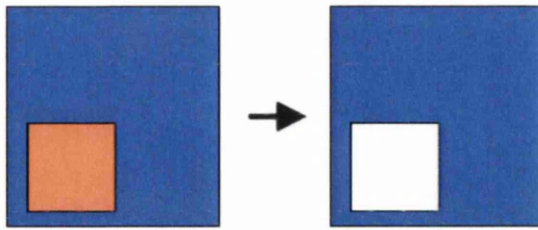
In Figure 11.10 two rectangles are combined together. The resulting shape takes on the style of the primary selected graphic in this case the larger blue rectangle. This is useful for creating shapes with holes or making bigger shapes where the different parts of the shape have the same style.

*Separate Paths* splits the parts of composite shapes out again. Split out graphics will always be Bézier paths, so rectangles that are combined then separated lose the ability to change their corner radius.

## 11.7 Style

The style of a graphic determines how it is drawn. Each graphic can have its own style specifying fill, stroke and effects. Fill specifies how the centre





**Figure 11.10:** Combining two shapes to create a single shape with a hole

of the graphic is drawn, the stroke specifies how the outline of the graphic is drawn, and effects specify shadowing and how the object is composited onto the page.

When there are no graphics selected, the stroke, fill, and effect inspectors show the current tool style. This is the style used when creating new graphics, this can be used to create many graphic with the same style.

The dropper tool transfers styles between graphics. Clicking on a graphic with the dropper tool picks up that particular graphic's style. The style is applied to the currently selected graphics or to the tool style. The individual components of the style can be picked up with the dropper tool by holding the Shift, Alt, and Command keys, which pick up the fill, stroke, and effects respectively.

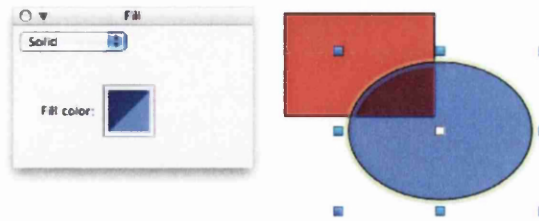
### 11.7.1 Fill

In Lineform there are five fill styles that can be applied to graphics from the Fill inspector: None, Solid, Image, Gradient, and Text. These are selected from the pop-up menu in the top-left of the inspector.

The fill of a graphic, image, gradient or text is usually transformed with the object when it is resized or changed. Sometimes this is not the behaviour wanted, holding the Command key during resizing or altering the shape allows the fill to remain unaltered. With this technique, pictures can be cropped or size of a text box expanded without altering the text.

To reset the fill, use the *Reset Fill Transform command* in the Objects menu. This will reset any alteration that has been applied to the fill, such as the rotation of an image.

The content of the inspector is dependent on the type of fill that is selected. Selecting None simply does not fill the graphic, leaving the centre of the graphic completely transparent.

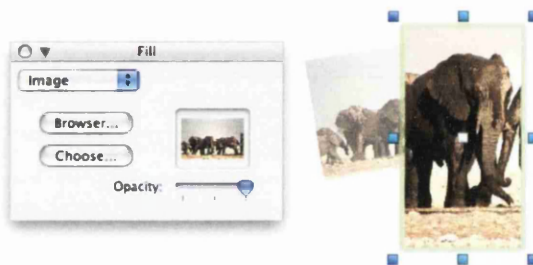


**Figure 11.11: Solid fills**

### 11.7.2 Solid

Solid fills the graphic with a solid colour. This colour can have an opacity setting, allowing both opaque and transparent graphics as extremes.

### 11.7.3 Image



**Figure 11.12: Image fills**

An Image fill draws an image inside the graphic. The image is clipped to the graphic's outline and can be clipped to arbitrary shapes. The slider in the inspector controls the opacity of the image. The *Choose...* button provides an open file dialog for picking an image and images can also be dragged onto the image preview in the inspector.

Images can be easily clipped by holding the Command key whilst dragging a scale handle.

### 11.7.4 Gradient

A Gradient fill creates a linear or radial smooth transition between two or more colours. The inspector provides a horizontal preview and a sequence of colour swatches. Colour swatches are edited by clicking on them and can be dragged around by clicking above the swatches on the gradient preview. Additional colours are added by clicking underneath the gradient preview and removed by dragging the swatches from the gradient preview.



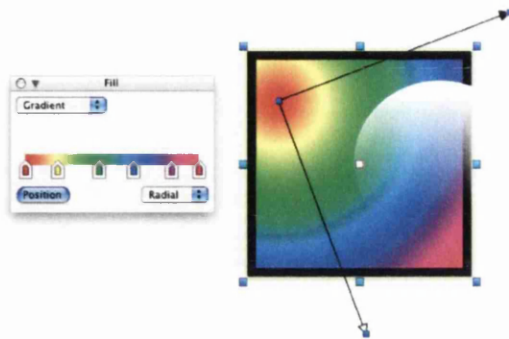


Figure 11.13: Gradient fills

The *Position* button provides an interactive way to change the location and direction of a gradient. Clicking the *Position* button begins editing the gradient, this enables direct manipulation of the gradient inside the graphic. While positioning clicking elsewhere on the canvas or pressing any key exits the positioning mode. The gradient handles can also be dragged around to alter the gradient position.

Holding the Shift key whilst dragging limits the angle of the gradient to  $15^\circ$ , the Alt key allows the setting of the individual position of radial gradient handles without affecting the other handles.

### 11.7.5 Text

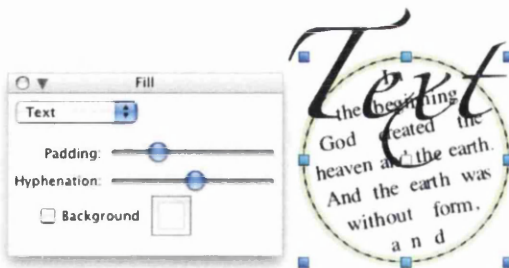


Figure 11.14: Text fills

A Text fill allows any shape to contain text. The padding and hyphenation of each text fill is set from the inspector. Text is edited by double-clicking the object and simply typing. Resizing the text graphic without resizing the text is achieved by holding down the Command key whilst resizing.

The background fill check box fills the graphic with a plain background colour. Padding insets the text inside a graphic and hyphenation controls how text is wrapped. In Figure 11.15 the '8's shows text accurately filling



Figure 11.15: Unhyphenated and hyphenated text fills

a complex shape, also styled with a blue background fill, an artistic stroke and a drop shadow. The text is fully justified: the left-hand 8 shows the unhyphenated text, while the right hand 8 shows the text fill with maximum hyphenation.

### 11.7.6 The text inspector

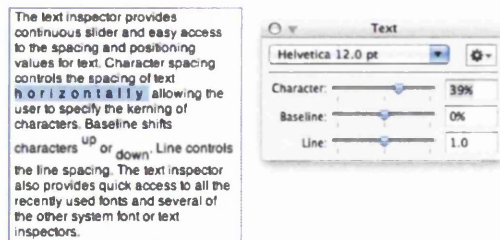


Figure 11.16: The text inspector

The text inspector, shown in Figure 11.16 provides a continuous slider and quick access to the spacing and positioning values for text. Character spacing controls the spacing of text horizontally allowing the user to specify the kerning of characters. Baseline shifts characters up or down. Line controls the line spacing. The text inspector also provides quick access to all the recently used fonts and several of the system font or text inspectors.

## 11.8 Stroke

Strokes are applied to graphics in the Stroke inspector, it is similar to the Fill inspector but applies to the outline of the shapes. There are four options for strokes: None, Solid, Artistic, and Text.

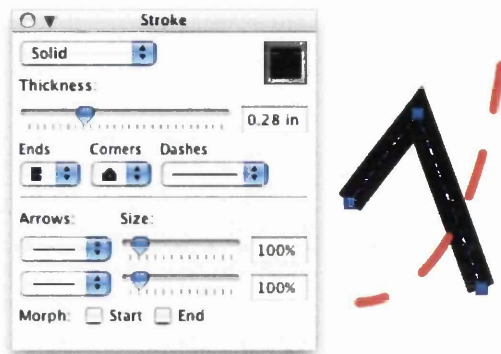


Figure 11.17: Strokes

Solid applies a solid colour to the line drawn and the inspector allows the colour, thickness, and style of the line can be set.



Figure 11.18: Stroke ends and corners

Figure 11.18 shows the ends of strokes which can be (left to right) Butt, Round or Square and the corners which can be set to Mitre, Round or Bevel.

Lines can also be dashed and this can be set from the Dashes pop-up menu.

### 11.8.1 Arrows

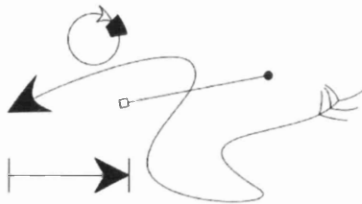


Figure 11.19: Arrows

Arrows can be placed at the beginning or end of any line. This is useful for drawing diagrams and charts. Each arrowhead can be sized relative to the line size and can be morphed to the line shape.

Notice how some of the arrows are morphed in Figure 11.19. Morphing can

be applied to the start or end arrow and can make the arrows look much more natural and professional.

### 11.8.2 Artistic strokes

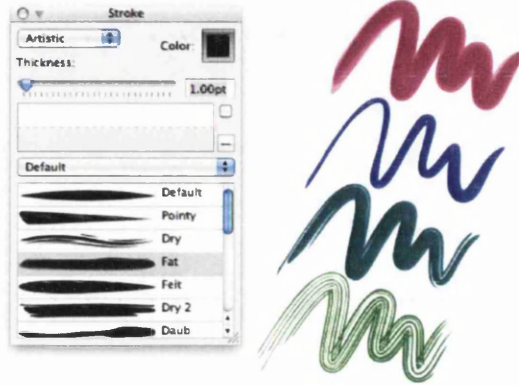


Figure 11.20: Brush strokes

Artistic strokes are an expressive way to draw vector objects. The strokes remain fully editable, and provide a richness in appearance that vector objects often lack. Figure 11.21 shows a single brush stroke being edited, the actual path of the shape is visible as a dashed blue, the fat red brush stroke is the result of an artistic brush applied to this path.

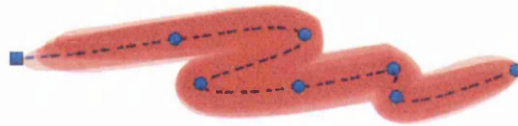


Figure 11.21: An artistic brush stroke



Figure 11.22: An artistic snail and its “skeleton”

Artistic strokes also provide a great way to create entire stylised drawings. This quick sketch of a snail in Figure 11.22 was drawn with a graphics tablet, making use of the brush tool. Layers were used to build up the snail using different artistic strokes with transparency to achieve the final image. On the right hand side of Figure 11.22 the “skeleton” of the snail is shown, this is what it looks like without artistic strokes.

### 11.8.3 Pressure

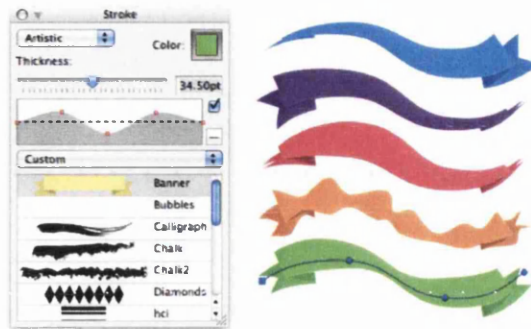


Figure 11.23: Identical strokes with different pressure profiles

Artistic strokes support varying pressure along the stroke, Figure 11.23 shows a custom banner stroke applied to the same path but with differing pressure profiles. The pressure of a stroke can be set in the stroke inspector by clicking to add nodes, dragging nodes or by deleting nodes by dragging them out of the profile. The pressure can also be reset using the ‘—’ button and can also be toggled on and off. Pressure is also recorded when a path is drawn using a graphics tablet, this provides a easy way to draw thick, thin or varying strokes.

### 11.8.4 Custom artistic strokes

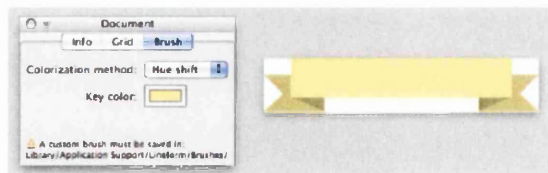


Figure 11.24: A custom banner stroke

Custom strokes such as the banner stroke in Figure 11.24 can be used to extend the default set of strokes. Custom strokes are created by saving Lineform documents in “Library/Application Support/Lineform/Brushes”, these documents are used to create the custom brushes.

Documents that are used as brushes distort the across the entire document size such that the vertical mid-point of the document will follow the path exactly. Figure 11.24 shows the banner document and the document inspector. The ‘Colorization method’ controls how the document is coloured when drawn as an artistic stroke and the ‘Key color’ specifies the colour from which a hue shift will occur, this should usually be the primary colour of a custom brush.

The different options for the brush colourisation are:

- None — No colourisation occurs.
- Tints — The brush stroke is colourised so that black becomes the stroke colour, white remains white and everything in between is a shade of the stroke colour going to white.
- Tints and shades — The brush stroke is colourised so that black and white remain the same, and everything in between is a blend from black to white through the stroke colour. The midpoint grey becomes the stroke colour.
- Hue shift — The brush stroke is colourised so the key colour becomes the stroke colour, and everything else is hue shifted relative to the key colour. Greys, black and white remain the same. This is a good choice for strokes that have multiple colours in them.

### 11.8.5 Text

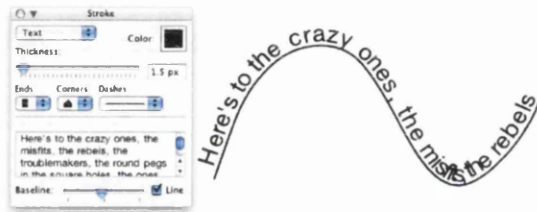


Figure 11.25: A text stroke

Text strokes allow placing text on a path which can be any shape or size. The text is edited inside the Stroke inspector and typeset as the user types out along the path from beginning to end, and paths that contain multiple segments have text typeset along each segment in turn. Text is scaled with the line thickness and a baseline offset can be specified to move the text perpendicular to the line.



Figure 11.26: Bad kerning at a tight corner

By using kerning, or character spacing in the text inspector, it is possible



to correct the compression and extension of the text that happens at sharp corners. Changing the text baseline alters the text layout. A baseline that centres the text over the line can eliminate some of the compression and extension.

## 11.9 Effects

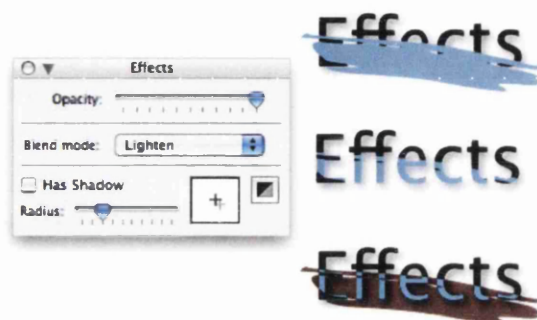


Figure 11.27: Blend modes

Effects change how an object is drawn or composited onto the canvas. It is possible to control the object's opacity, blend mode and shadow through the Effects inspector.

Opacity controls how transparent an object is. The ability to control the opacity of groups is useful as it controls the opacity of the group as a whole.

Blend modes specify how the colours of objects should be mixed. Blend modes allow for powerful compositing modes but are also fairly complex. To understand how blend modes work, it is probably best to experiment with them. Figure 11.27 shows a brush blended with different modes. From top to bottom the blend modes are: Normal, Lighten and Difference. Blend modes are very effective when combined with artistic strokes to add extra depth to an object or image.

Drop shadows are a simple effect that add a lot of depth to graphics, these can be applied to any object in Lineform. Shadows are added to graphics in the Effects inspector. The radius (how fuzzy the shadow is) of the shadow and its colour and position are also set from the inspector.

## 11.10 Filters

Filters allow more complex effects, ranging from blurring and colour adjustment to halftones, which can be easily applied to graphics whilst the graphics remain editable.

Lineform provides a powerful set of filters based on Apple's Core Image technology. Using Core Image allows many effects to be achieved which would not be possible with pure vector objects. For example, filters make it simple to create a Gaussian blur of a graphic, as shown below in Figure 11.28. The same graphics are shown on the left without filters applied and on the right with filters.



Figure 11.28: Filters on some text

Filters are controlled through the Filter inspector, shown in Figure 11.29. At the top of the Filters inspector are three controls, from left to right: whether filters are enabled, the resolution at which the filters are applied and an action button to add new filters. Below these controls is the stack of currently applied filters which can be individually minimised, disabled, deleted and re-ordered. Each filter also has its own parameters that can be set.

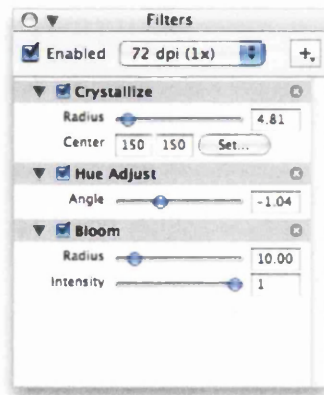


Figure 11.29: The filter inspector

Many filters are available, and it is worth spending some time experimenting and exploring the possibilities. The filters are categorised by type as: Geometry Adjustment, Distortion Effect, Blur, Sharpen, Color Adjustment, Color Effect, Stylize, Halftone Effect, Tile Effect, Generator, Gradient, Composite and Transition. To add new filters, click the top-right button in the inspector. To delete filters once they have been added, click the X button in the right of the filter title-bar. The filters are applied in order from top to bottom and can be re-ordered by dragging the grey title bar up or down to move the filter above or below other filters. The order of the filters can often have a large impact on the final result.



### 11.10.1 Filter resolution

Because filters are, by necessity, bitmap operations that work at the pixel level, the resolution at which the filters are applied makes a difference to how they look. Lineform endeavours to keep the filters as independent of resolution as possible, however it does not provide complete resolution independence.

Each object can be set to have a resolution of 72, 144 or 300dpi, the lowest being screen resolution and the highest being good for printing. For print it is usually best to make use of higher resolution filters, but it is worth experimenting, as some filters with very gradual transitions, such as the Gaussian blur, often work well at low resolutions. Low-resolution filters are faster to draw and take up much less memory.

Bitmap images included in Lineform are optimised to have filters applied at their native resolution. This is only the case when the filters themselves are resolution-independent, such as colour adjustment, and the image graphic does not have a stroke. This allows filters such as colour adjustment to be applied to images at their native resolution and at faster speeds.

## 11.11 Editing graphics in depth

Graphics can be edited and altered in different ways. The most flexible and useful way to edit graphics is to use the Edit tool. This allows shapes to be changed and altered in any fashion. Once the edit tool is selected, nodes appear on the graphics being edited. The type of nodes depends on the type of graphic — rectangle, oval shapes and paths have different nodes.

Lineform provides several different ways to begin editing graphics. The simplest is to select the edit tool from the toolbar, any graphics that are selected will become editable. The Return or Enter keys also toggle between the selection tool and the edit tool, it is possible to hit Return repeatedly to switch modes.

Lastly, it is also possible to double-click to start editing. A double-click acts like a normal selection click, except that it starts editing, thus a double-click can select objects to add and remove them from editing, just as clicking does with the selection tool. A double-click-drag selects multiple graphics for editing and the same modifiers affect the double-click selection as normal selection. The Shift key allows graphics to be added and removed from editing, the Alt key selects only graphics that are fully within the selection, and the Command key allows the selection of graphics below the top object. All these shortcuts are effective both in and out of edit mode.

The double-click selection provides a simple mirror of the standard selection tool and is a very effective way of quickly editing graphics.

Text objects have a slightly different default behaviour. Double-clicking a text object or pressing Return when one is selected will begin editing its text. The object itself can be edited by selecting the edit tool or by editing it at the same time as other objects.

### 11.11.1 Rectangles and ovals

Rectangles and ovals are editable in additional ways, they provide easy methods to alter the specific attributes of the shape. If further editing is needed, these shapes can be converted into Bézier shapes. This is done with the *To Bézier* menu command. Once a rectangle or oval is converted to a Bézier shape it is editable in any way.

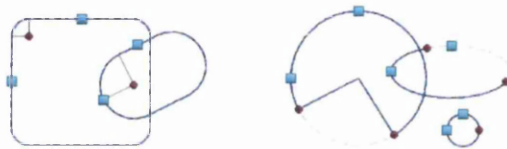


Figure 11.30: Rectangle and oval special shapes

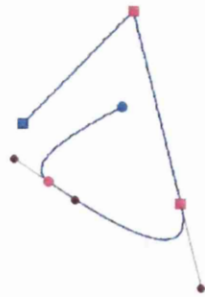
Editing a rectangle allows a rounded corner to be set and the rectangle to be scaled on its original axes. The cyan nodes in Figure 11.30 are used to scale the rectangle along its width and height. The small purple node sets the radius of the rounded corner.

An oval is edited in a similar fashion; two small control nodes allow the oval to be changed into an arc, the nodes control the start and end angles of the arc. If the mouse is dragged inside the oval when dragging a control node, the oval becomes a pie, if the mouse is dragged outside the oval, it becomes an arc. Holding the Shift key forces the angles to be constrained to  $15^\circ$  intervals. Ovals are drawn when the two control nodes are close, this behaviour can be prevented by holding the Alt key.

### 11.11.2 Bézier paths

The shape in Figure 11.31 is being edited, and the squares and circles represent the nodes of the shape, these control the shape of the graphic. In Figure 11.31 there are two blue unselected nodes and three pink selected nodes.

Square nodes represent a line segment and the circle nodes represent curved segments. The smaller purple circles connected to some of the nodes are control handles that affect the shape of the curve. These control handles only appear on selected nodes that are adjacent to a curved segment of the shape.

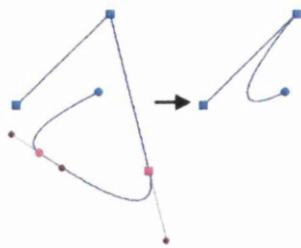


**Figure 11.31: A Bézier shape being edited**

Selecting nodes work in a similar fashion to objects. Clicking on nodes or dragging around them selects the nodes. Holding the Shift key adds or removes nodes from the selection and the Command key selects nodes from underneath, allowing access to nodes on top of one another.

Once nodes are selected they also act in a similar way to graphics. Dragging moves them and holding the Shift key limits the movement to horizontal or vertical. Nodes can also be nudged using the arrow keys in the same fashion as objects.

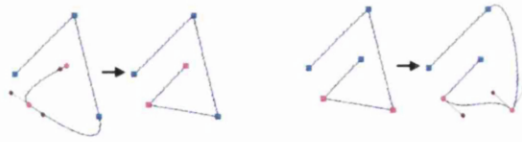
Control handles can be dragged but not selected. Each curve node has two control handles, which are automatically aligned. The Alt key allows the handles to move independently, the Shift key constrain the handles to 15° angles and the Command key keeps the control handles at a constant angle while dragging.



**Figure 11.32: Deleting nodes in a Bézier shape**

The selected nodes can be deleted by pressing the Delete key. Deleting nodes removes them from the shape; the shape is joined up between where the nodes were removed.

Changing nodes to lines, shown in Figure 11.33 on the left, alters the shape so that the nodes selected are now lines and not curves. This and the other editing functions can be accessed from the Objects menu or the contextual pop-up menu. The reverse (Figure 11.33 right), changing nodes to curves,



**Figure 11.33: Changing node types in a Bézier shape**

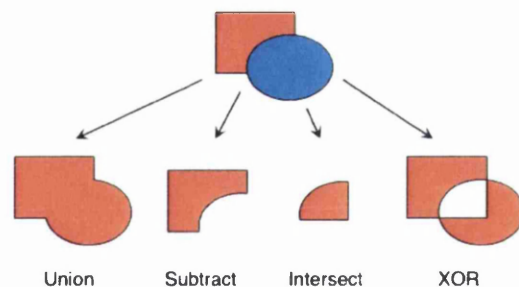
converts any selected nodes to curve nodes. Initially this will look the same until the control handles are moved.

To add a node at any point in a Bézier path without altering its shape, Alt-Command-click at the point in the path to place the new node. New nodes are added without changing the shape of the path.

There are three extra editing commands: duplicate, connect, and cut. Duplicate creates two nodes on top of one another, this is useful for adding in extra nodes and detail into shapes. Connect joins nodes together within the same object. Finally, cut breaks the shape at the nodes selected, creating multiple segments.

### 11.11.3 Boolean operations

Boolean operations are a way of combining two or more graphics together. The resulting graphic is created from a Boolean (yes/no) combination of the selected graphics and takes the style of the primary graphic. Figure 11.34 shows the results of the four boolean operations: union, subtract, intersect and XOR.



**Figure 11.34: Boolean operations**

The Union Boolean operation combines two or more overlapping shapes into one larger shape, consisting of the outer contour of those shapes.

The Subtract Boolean operation removes or subtracts the shape of the selected graphics from the primary graphic. In Figure 11.34 the orange rectangle is the primary graphic. If the circle were the primary graphic, the

resulting graphic would be a blue pie shape missing the top-left quadrant.

Using the Intersection Boolean operation results in a shape that consists of the overlapping area(s) of the selected graphics. If the graphics do not overlap, an Intersection is the same as deletion.

The XOR Boolean operation creates a shape where an odd (one but not two) number of graphics overlap. The same effect can be achieved by simply combining the graphics together. However, XOR creates a new shape with new control points that allow further editing.

#### 11.11.4 Outline

Outline converts the outline of the text or graphics into a path. This creates an outline that can then be edited, filled, and manipulated.



Figure 11.35: Outlining a shape

Figure 11.35 shows a rounded rectangle with a gradient fill (left) and the result of outlining the rectangle with the same fill applied (right). Without outlining the path, a gradient stroke is impossible.

There are many other possible uses of outlining, such as drawing walls in a floor plan, creating logos out of text, and using Boolean operations on the outlines of artistic strokes. Outlining text is an important step in lots of workflows because it allows the text characters to be manipulated, joined, and edited as Bézier paths, creating logos and designs which are visually interesting.

## 11.12 The canvas in depth and exporting

### 11.12.1 Rulers, guides and grid

Lineform provides several methods of aiding accurate positioning of graphics. Rulers allow accurate measurement of distance, horizontally and vertically. Guides can be added to both rulers, these draw pink lines across the drawing area that graphics automatically snap to.

Guides can be easily used to align objects, size objects and create objects with accurate sizes. Guides can be hidden or cleared from the View menu.

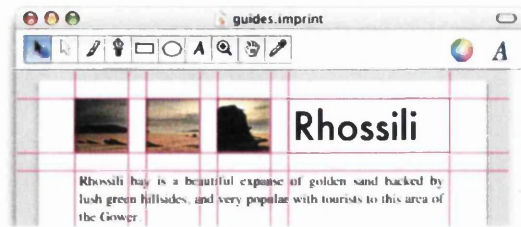


Figure 11.36: Guides

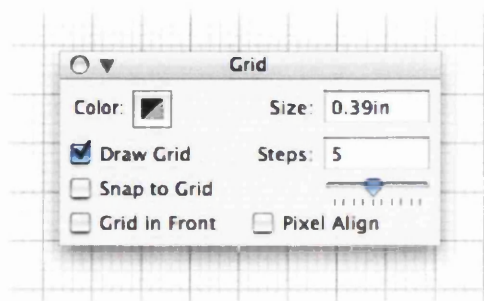


Figure 11.37: Grid inspector

A grid provides a uniform set of guides that graphics can snap to throughout the document. The grid size, steps and colour are all set from the Grid inspector. A grid is useful for drawing many different things including technical drawings, floor plans and typesetting layout. The grid can be set to any major spacing size with minor spacing or steps. The colour and drawing properties of the grid can also be altered from the Grid inspector. Pixel Align aligns the grid lines in the centre of pixels so that bitmap drawings along grid lines are not anti-aliased.

### 11.12.2 Page layout

The Layout inspector sets the size of the canvas in number of pages. Changing the actual page size, for example to A3, is done in the page setup dialog box. This is accessed from either *Page Setup...* in File menu or by clicking the button in the Layout inspector.

The layout panel also allows the canvas margins to be changed, which is useful if the drawing should reach to the exact size of the paper, however most printers will not print to the edge of paper. A specific document size can be specified by unchecking the “Size in pages” option, the document size can then be entered as an exact size.



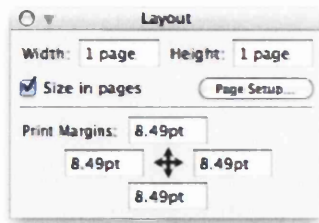


Figure 11.38: Layout inspector

### 11.12.3 CMYK preview

Printers and screens use different methods of creating colour. Screens use an additive combination of red, green, and blue (RGB); printers mostly use a subtractive combination of cyan, magenta, yellow, and black (CMYK). Because of the difference in how colours are created, it is possible to show colours on screen that cannot usually be printed. This often creates unexpected and disappointing results for the user, who has often spent a long time getting the colours exactly right.



Figure 11.39: RGB and CMYK preview (note: these images will appear the same in a printed version of this thesis)

To avoid this problem Lineform provides a *soft proof* mode which is toggled on and off from the View menu. This draws the current document using a different colour space. This is by default a generic CMYK colour space but can be set to a number of different options, such as grayscale, in the preferences. The soft proof provides a clearer idea of how the drawing will appear when drawn in that colour space. Figure 11.39 shows two screen shots of a composition, the left hand screenshot is the normal view of the document, the right hand shows the CMYK soft proof of the same document. Notice how the bright blues and reds are more muted in the CMYK soft proof. This is closer to what will be printed on a CMYK printer.

### 11.12.4 Outline view

An outline view renders all shapes as outlines, allowing easier selection and editing of overlapping and complex drawings. Like the soft proof it is toggled

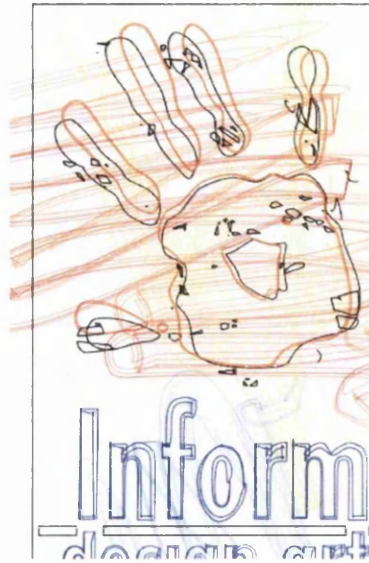


Figure 11.40: Outline mode

on and off from the View menu. In outline mode, shapes are rendered as outlines, without any effects or fill. A composition of a logo can be seen in Figure 11.40. The artistic strokes composited using blend modes on top of the hands, which cannot be seen normally, can be seen as outlines.

### 11.12.5 Export

It is possible to export graphics from a document to both vector and bitmap file formats. To export graphics, select the objects then use Export... command in the File menu. Graphics can be exported to BMP, EPS, JPEG, PDF, PNG and TIFF file formats. When exporting the exported transparency and resolution can be specified in the export dialog.

EPS, PDF and SVG are vector formats and are resolution-independent. These formats should be used if the exported file will be printed.

PNG and TIFF are the only bitmap formats that support transparency.

### 11.12.6 SVG

SVG is a modern XML-based standard for vector-based artwork that is independent of resolution. Lineform supports SVG as both an import and export format. SVG support is growing in many places such as web browsers, and also provides a good intermediary format between Lineform and other programs. Lineform provides comprehensive support for the basic SVG standard.



### **11.12.7 AppleScript**

AppleScript support is built into Lineform. If the user needs to extend or compliment Lineform's abilities with additional features or automation, then it is simple to do so using AppleScript.

## **11.13 Summary**

This chapter has provided a comprehensive overview of Lineform's many features. Vector graphics have many different aspects and are complex to create and edit. A graphics editor mirrors this complexity and itself requires many different features and abilities to provide full editing capabilities. However with good design this does not mean that the user interface needs to be complex. Lineform manages to provide a simple and elegant user interface for the complex task of vector graphics editing.

## Chapter 12

# Implementation

This chapter provides an overview of the implementation of Lineform. To provide a sense of the scale of the implementation, Lineform now comprises of 67 classes and over 12000 lines of code written in both C++ and Objective C. The majority of the code is written in Objective C and makes extensive use of the core Mac OS X libraries including: Cocoa, Quartz and Core Image.

This chapter describes the three main aspects of Lineform's design and implementation. Firstly, the document model structure that contains the vector drawing and drawing model is described; secondly, the user interface and interaction structure is examined and finally the vector file format and representations supported are described.

### 12.1 Document model structure

A Lineform document contains all the information about a single vector drawing. The drawing is composed from a small number of classes, which provide the methods to draw the document using Quartz, serialise it in both Lineform's own proprietary format and in the SVG format and support the modification and editing of the document structure.

Figure 12.1 shows a simplified version of Lineform's class structure for representing documents and the contained vector graphic drawings. Each document can contain many graphics, such as groups, Bézier graphics, rectangles or ovals. Each graphic has a fill and a stroke style that is used to draw the graphic. These few classes provide the basis for the construction of every drawing.

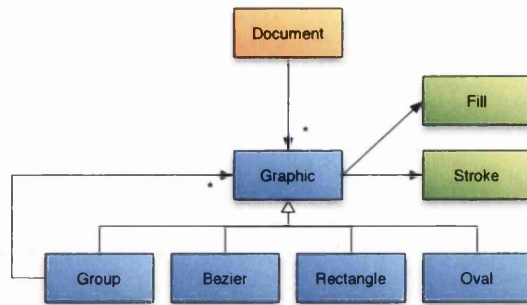


Figure 12.1: Lineform's document structure

### 12.1.1 Drawing model

All of the vector drawing in Lineform utilises the Quartz 2D library in Mac OS X. Quartz's drawing model is based on PDF's model, which provides the simple concept of compositing paint on-top of selected areas of a drawing to produce the final output. Summarised, the model provides three main capabilities:

- Shapes can be in the form of character shapes (glyphs), geometric Bézier shapes or polygon lines.
- These shapes may be outlined or filled with a paint or used for clipping other shapes. The paint may be any colour and have transparency, or may also take the form of a repeating pattern or a smooth gradient between colours.
- Sampled images such as digital representations of photographs can be drawn.

### 12.1.2 The document

The document's primary purpose is to manage the collection of graphic objects that make up the drawing. The document class encapsulates various properties such as the paper size, printer information, grid resolution and rulers; it also handles high level resource management for resources such as images and external files.

Finally the document class provides general file management for saving and loading. The document can be saved as vector graphics in EPS, PDF, SVG and Lineform's own format. The document can also be saved as various raster formats, at different resolutions, the formats supported include the PNG, JPG, BMP and TIFF formats. The EPS, PDF and raster format support is primarily provided by the Quartz drawing library.

The basic structure of a drawing contained within a document object is

an array of groups that provide the main layers of the document and which contain all the graphics that make up the drawing. Creating the final output of a document, whether raster or vector, is done by drawing each of these layers or groups in sequence from back to front.

### 12.1.3 Graphics

The graphic class provides the encapsulation of all the properties of the graphic's shape, including how to composite and draw the graphic. It also provides support for the user interaction of creation, editing and transformation.

The individual subclasses of the graphic class: rectangles, ovals and Bézier graphics provide specific editing capabilities for each subtype. For example, the rectangle class encapsulates the roundedness of its corners and implements the user interaction for modifying them.

The Bézier graphic has the most complex interaction, it provides extensive node based editing such as node transformations, connect, join, node selection, hit testing and path creation using different interactions.

### 12.1.4 Groups

Groups provide a different function to the other graphic types, the distinction is similar to the difference between a folder and a file in a disk hierarchy. Instead of providing a shape that is drawn, groups contain other graphics and draw all of the objects it contains from back to front. Groups provide the Z-ordering and organisation of individual graphics and also subgroups. The group class provides the functionality to rearrange, insert and delete its contained graphics. All object organisation in Lineform is performed by groups. They also allow effects, such as transparency and clipping to be applied to a collection of graphics. Layers used to be a separate concept but were merged with groups providing simpler code and a more powerful user interface.

Groups also provide the the ability for the user to manipulate multiple graphics as if they were a single graphic and the ability to apply composition effects, such as opacity, to several graphics at once.

### 12.1.5 Fill

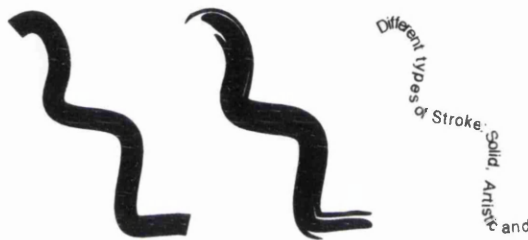
The Fill class implements the functionality that draws the insides of a graphic. Lineform provides five different types of fill shown in Figure 12.2: None, Solid, Image, Gradient and Text.



**Figure 12.2:** Lineform's different fill styles

The Fill class provides the drawing functionality for all these different types of fill. The Solid and Image fills provide simple colour and image contents for a graphic. The Gradient fill provides smooth transitions of colours that can be adjusted both in the inspector and on the canvas. The Text fill provides a rich-text container that flows text inside the shape, the text can be edited using the standard text editing components by double clicking the graphic.

### 12.1.6 Stroke



**Figure 12.3:** Some of Lineform's different stroke styles

The Stroke class provides similar functionality to the Fill class, but for the outline of a shape. There are four different types illustrated in Figure 12.3: None, Solid, Artistic and Text.

A Solid stroke is drawn with a simple fixed width brush of solid colour along the outline of the shape, a Solid stroke also has properties for end-cap shape and dash pattern. An Artistic stroke warps a vector drawing or brush so that the X-axis of the brush is distorted to follow the outline of the shape. The resulting effect is a vector distortion that can create powerful effects, like those of natural media, just by using vector graphics.

The idea is similar to that of skeletal strokes [Hsu et al., 1993], but the implementation is different. Skeletal strokes are generated by cutting the brush into lots of small sections with more frequent cuts at locations of high curvature on the shape the brush is drawn along. Lineform's implementation recursively subdivides the distorted Bézier segments until the distorted segment is accurate enough. Figure 12.4 shows the result of a 'S' shape drawn using a skeletal stroke which uses a high density polygon and Lineform's recursive subdivision brush stroke which uses a smaller number cubic Bézier segments to achieve the same effect, the blue nodes show the polygon or Bézier control points.



**Figure 12.4:** An artistic brush using skeletal strokes and recursive subdivision

The Text stroke warps text along the outside of the shape outline by placing each character such that its baseline lies along the path. The text is edited in the Stroke inspector and can be made up of multiple different styles and fonts and the result on the canvas is updated live.

## 12.2 User interface

Figure 12.5 shows the basic class structure of Lineform's user interface model. The structuring of the implementation of the user interface is based on the Model-View-Controller (MVC) architectural pattern [Reenskaug, 1979]. The MVC pattern de-couples the user interface and the data structures or model of a system by introducing an intermediary controller. The benefit of this architectural pattern is that both the user interface and the model can be changed and reorganised without changing the other. In this case, the Model is the graphics contained in the document and document state, the View is the visible user interface (the graphics view, inspectors and toolbar) and the Controller that provides the interfacing between these is primarily the Document class (shown centrally in Figure 12.5). Although there are more classes that are not shown in Figure 12.5, this is an accurate but simplified structural view. For example, in Lineform there is more than one controller class, as each of the different aspects of the user interface have different individual controller classes.

The document as the primary controller, takes central place in the structure of the user interface message passing. When an attribute is changed in an inspector, the selected graphics are altered by passing the message through the document. The altered graphics inform the document what area of the drawing has changed, the document then tells the graphic view which area needs updating. The document is an intermediary for most user interface

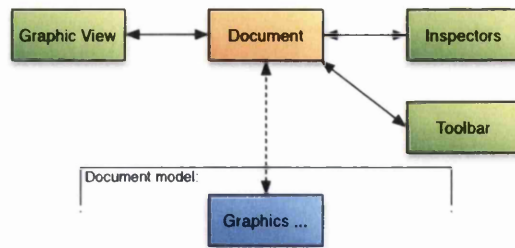


Figure 12.5: Lineform's user interface structure

changes. A similar process takes place when graphics are edited in the main graphic view. The changes that are made to the document, which then informs all the different views of the graphics data (the graphic view, inspectors and toolbar) ensuring that they are always up-to-date (projected editing).

## 12.3 Representations

Vector graphics can be represented in many different formats. All of the different representations support different sets of semantic data. The main vector representations Lineform supports are: its own specialised '.lineform' format, SVG and PDF. These formats are summarised below:

- Scalable Vector Graphics (SVG) [W3C, 2003] is a XML based open standard developed by the World Wide Web Consortium for both static and animated two dimensional vector graphics.
- Portable Document Format (PDF) [Adobe, 2008] is a file format created by Adobe Systems, for representing two dimensional documents in a device and resolution independent format. It is also the ISO 32000 standard. Adobe's development of PDF was motivated by the need to extend PostScript, a page description language, to encapsulate all the necessary data needed for page rendering, for example images and fonts.
- Lineform's own format provides much of the same functionality as PDF and SVG. The vector data format is primarily a binary serialisation of the graphics object graph that a document consists of. The file format is a bundle that contains the binary vector data and any imported images used in the document saved as PNG files. (A bundle is a folder or directory in Mac OS X that looks and acts like a file but has arbitrary content.)

Lineform's format was originally stored in the YAML format [Ben-Kiki et al., 2004], because of YAML's readability and ease of editing while the format



specification was being developed. However version 1.1 of Lineform switched to using a binary format, this was primarily because the speed of loading the binary documents was many times faster (in some cases 10 times faster). Parsing large text based “human-readable” formats like XML and YAML was found to often be very time consuming compared to binary storage, storing documents as binary data also meant that they took up much less disk space. Switching to a binary storage had some drawbacks in the code complexity needed to handle both formats but provided a huge improvement in user experience, especially when handling large documents.

An advantage of using a bundle and separating the vector data from the image data, is that file access can be optimised. Saving document changes can be optimised because writing changes to the vector data does not mean that the images, which are potentially very large, also need to be written to the disk. Similarly adding or removing images does not mean that the vector data needs to be rewritten. Bundles also provide a simple process to access the images used in a Lineform document, the individual image files can be accessed by selecting the “Show Package Contents” item in the Finder.

### 12.3.1 Semantics

It is possible to represent nearly all vector graphics drawings in the three different file formats. The final output will look identical, however semantic information is often lost in translation between two of the formats. An example is text on a path like the example in Figure 12.3, this is represented in SVG as a `textPath` element that retains the semantic data but can only be represented in a PDF file as lots of individual characters with different transformation matrices. They look identical, but the PDF version lacks the extra semantic information, which makes it much harder to edit.

Often this semantic data is unnecessary, for instance when printing, only the appearance of the vector graphics is of interest. However for editing, importing and changing the vector graphics the semantic data is important. Editing a string that was attached to a path and is now composed of single unattached characters is almost impossible, because the editor does not recognise the characters as a string but as many separate objects.

Adobe Illustrator CS2 uses a PDF file format that is augmented to contain extra proprietary semantic data. This provides semantic data, like the path data in the previous example, to allow easier editing. Inkscape, an open source vector graphics editor, uses an augmented SVG format as its native file format. Both Illustrator and Inkscape modify the respective standard file formats but in a way that remains compatible with the standards. This allows each to extend their chosen format, whilst maintaining a native format that is a useful export.



Illustrator also exports vector graphics as SVG and Inkscape (with some modifications) also exports PDF.

Lineform uses a different approach: it uses its own file format that can then be exported as both PDF or SVG. The advantage is that the file format and data structures are designed specifically for the semantic constructs Lineform supports. This means the files can be smaller and the user interface and graphical objects the user interacts with are separated and abstracted from the specifics of a target format, whether PDF or SVG.

Feature	Lineform	SVG	PDF
Solid strokes	yes	yes	yes
Text on a line	yes	yes	no
Arrows	yes	partially	no
Artistic strokes	yes	no	no

The above table shows some of the features of line drawing styles the different formats support. Lineform provides the most options, although all formats are capable of the same visual output. Lineform’s features such as artistic strokes which are unsupported in PDF and SVG have to be converted to simpler vector graphics when exported to these formats.

12.3.2 Comparisons

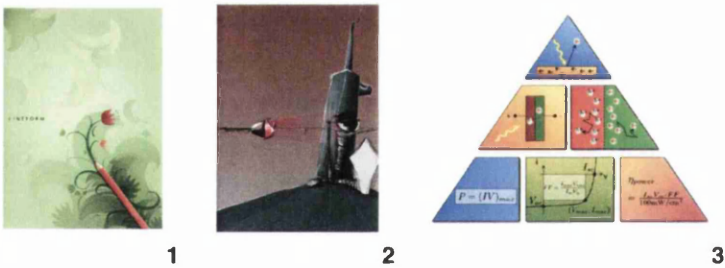


Figure 12.6: Three Lineform documents

The following Table shows the different files sizes for the documents in Figure 12.6. Document 1 is a simple vector image with gradients, Document 2 is a complex vector image with hundreds of paths and artistic strokes, and Document 3 is a simple diagram.

Document	Lineform	Lineform (XML)	SVG	PDF
1	49KB	245KB	49KB	127KB
2	393KB	2,252KB	4,300KB	2,355KB
3	33KB	224KB	56KB	304KB

Lineform contains some semantic data that was ignored for these comparisons.

- Although further compression (for example zip) is possible for all the document formats; in all three examples Lineform's file format size is the smallest.
- Lineform's raw format is always a lot smaller than Lineform (XML) which is the XML equivalent to the binary Lineform file.
- Document 2 is very complex, and contains over 1600 artistic strokes. The use of artistic strokes provides Lineform a large advantage over the other formats, due to the semantic compression of storing a single path and a reference to a brush instead of the complex stroked outline of the brush along the path. This semantic compression is sufficient that the Lineform (XML) version is still smaller than PDF, which is a binary format.
- PDF has an overhead that increases the file size for small documents but for drawings with lots of stroke data, such as document 2, it is half the size of the respective SVG document.

## 12.4 Other features

### 12.4.1 Linkback

Document 3 in Figure 12.6 contains LinkBack<sup>1</sup> data so that the equations in the document can be double clicked in order to edit them in the original editor outside of Lineform. LinkBack is a library for Mac OS X that provides a method of embedding semantic data and providing a link to the originating application. Lineform itself does not use this data but stores it so that objects can remain externally editable in the original linked application. The  $\text{\LaTeX}$  equations in document 3 remain editable in the original equation editor even though Lineform does not handle  $\text{\LaTeX}$ . When these objects have been edited externally and saved, the new objects are passed back to Lineform which updates its document with their new appearance.

### 12.4.2 Scripting

Lineform provides scripting support through AppleScript. Scripting allows Lineform to be extended beyond the tools it provides by allowing an external program to manipulate the graphics in a document. Scripting provides a powerful addition to what Lineform can do, because it allows users to write scripts that change Lineform to meet their needs, in ways it could never have originally been designed for. Scripting also means that Lineform can be controlled from other applications, which allows Lineform to be part of extended workflows. For example, Lineform could be used as part of

---

<sup>1</sup><http://www.linkbackproject.org/>

a workflow to generate a PDF contact sheet from a collection of images downloaded from the web.

Listing 12.1 shows an AppleScript script that prompts the user for a number of sides and generates a regular polygon or star in Lineform. Lineform does not provide the tools to create stars and regular polygons and scripting can provide this ability as a useful extension to Lineform.

Listing 12.1: Polygon generation script

```

— A simple script to provide Lineform with star and
  polygon shapes

tell application "Lineform"
  tell first document
    activate

    — Get parameters from the user
    display dialog "Please enter the order of the polygon."
      default answer "5" buttons {"Next"} default button
        "Next"
    set order to text returned of result as integer
    display dialog "If you want a star, please enter a
      ratio for the point radius." default answer "0.5"
      buttons {"Star", "Polygon"} default button "Polygon"
    if button returned of result = "Star" then
      set radiusRatio to text returned of result as real
      set order to order * 2
    else
      set radiusRatio to 1
    end if

    — Default settings
    set radius to 100
    set xcenter to 200
    set ycenter to 200

    — Generate path data
    set pathInfo to "M" & xcenter & "L" & ycenter - radius
    repeat with i from 1 to order
      set angle to i / order * 360 + 90
      set myRadius to radius
      if i mod 2 is not 0 then
        set myRadius to radius * radiusRatio
      end if
      set pathInfo to pathInfo & "L" & xcenter + myRadius
        * (my cosine of (angle)) & "L" & ycenter +
          myRadius * -(my sine of (angle))
    end repeat

    — Create path
    make new path at end with properties {data:pathInfo}
  end tell

```

```
end tell
```

### 12.4.3 Core Image

Core Image is Apple's framework for image manipulation, which provides raster operations such as Gaussian blur, crystallise, and sepia tone. These operations are calculated with 32-bit floating point math, so there is little loss of image quality or precision throughout the image processing pipeline. The filters are compiled down and lazily run across multiple CPUs and GPUs. Core Image also provides a plug-in style architecture for accessing filters, transitions and effects packages called Image Units. This means that filters and image manipulations can provide real-time, interactive responsiveness as you apply and adjust them.

Core Image fits in well with Lineform's approach to interactivity. There are no progress bars or delays between selecting or changing filters and seeing the results because it is fast. Shapes and graphics can have Core Image filters applied to them with the filter palette. The underlying graphics remain vector based, and are still completely editable.

## 12.5 Summary

Lineform is implemented with a straight forward object oriented model-view-controller model using a small number of core classes.

Although the implementation is not a restriction on user interface principles, by using a good flexible design the implementation supports the same flexibility that Lineform's user interface provides. For example, in Lineform text is available as both a fill and a stroke type for any graphic, these provide text-on-a-path and arbitrary text fills, this is a different user experience to using specialised graphic types which needlessly restrict a user in how they are used. Lineform's implementation and class design, which provide text as a fill or stroke, enable this flexible user interface directly through the class model design.

## Chapter 13

# Evaluation

This chapter provides an evaluation of Lineform, which is primarily provided through expert reviews. A cognitive evaluation using Green's cognitive dimensions supplies a different approach using a heuristic evaluation to compare Lineform and Adobe Illustrator.

### 13.1 Why expert reviews?

The evaluation of Lineform in this thesis is primarily provided through professional expert reviews. The purpose of this method of evaluation of Lineform is to examine and highlight the differences the principles make.

Simply creating two different versions of Lineform, one using the principles of *continuity*, *projection* etc. and one without them would, at its best, be a highly biased evaluation. A particular problem is that Lineform's user interface was designed with its principles as core design concepts. To remove the principles from Lineform would be to neuter the design and weight any comparison heavily in favour of the success of the principles. To create a drawing application without the same principles would require starting from scratch and building an application with a different philosophy.

Thus perhaps the best comparisons for evaluation are drawing applications such as Adobe Illustrator and Corel Draw, which while they have had many thousands of design hours put into them were not designed explicitly using *flow* or other Lineform principles.

User and comparative evaluations are difficult for a large program such as Lineform. Lineform is used by professionals and its main competition and comparison is Adobe Illustrator, a major application with over 20 years of development. Both programs take weeks (and possibly months for Illustrator) to learn and few users have the time to be familiar with both. Also the very fact that they are creative programs where the output is often subjectively good makes empirical comparison very hard.

Expert reviews are a good tool because unlike users, they are unbiased through voluntary choice and unlike usability study participants, experts have an expertise in drawing and a wider experience of comparative products. Their unsolicited reviews and comments are made with the knowledge and skills of drawing.

Users provide useful feedback, but often their comparative feedback is statistically flawed, they are self selecting and have already decided that Lineform is a suitable solution for them. Several of the examples of user feedback in Chapter 9 also highlight the desire of users that Lineform would work exactly like other applications. Thus this chapter is focused on expert reviews by unbiased professionals. While the reviewers are not necessarily user interface experts (in the sense of being HCI professionals) they do have considerable expertise in illustration, vector drawing and knowledge of similar programs. Their views are arguably better than the usual “*n* students in my department” approach to evaluation!

## 13.2 User reaction

Lots of users evidently really like Lineform. Here is a short selection of some of their quotes. Chapter 9 also provides more user feedback that helped shape Lineform’s design.

Ostensibly a competitor to Adobe Illustrator, Lineform is a vector drawing program that’s almost completely different: it’s small, efficient and reasonably priced. Revolutionary, right? Lineform provides ninety percent of Illustrator’s crucial functionality in just one-tenth of the disk space; it claims just 7.1MB on your hard drive and US\$79.95 from your wallet to use and own this program, and it’s a thing of beauty.

— Khol Vinh, Director of NYTimes.com. (Oct 2006)

This is probably one of the best software products I’ve ever seen for the creative artist or business person. I had been trying the 30 day free trial and after a week, I had created so many logos and animated characters for my business that I couldn’t wait to get the whole program. I’ve been able to use it in conjunction with other programs like Comic Life and even word processing programs. This program is one of the reasons I am always trying to sell my friends on Mac and why I will never use anything else. There is nothing that even comes close to this program for ease of use, adaptability and creative potential. And I can say this now that I have the program, but at twice the price, it would still be cheap for what you get. I suggest you go to their website and download the trial version first. I think you’ll see what I’m saying is true.

— Peter Marino (Amazon user review, May 2007)



Figure 13.1: Clockwise © Martin Howard, Rory Prior, Bill Rogers, Jonathan Leavitt, Aarni Heiskanen, Matt Gibson

I'm a pretty heavy Illustrator user and I love it. I bought a copy of Lineform to see if there's an alternative for use at home. The learning curve was really low and features like the bitmaps as fill are great (why doesn't Illustrator have this?).

— Alan Brown (Amazon UK user review, Nov 2007)

Lineform users also initiated a group on Flickr (<http://www.flickr.com>) for sharing images created with Lineform. Many users have shared their Lineform designs, pictures and creations. A small selection of these and other user images are shown in Figure 13.1.

### 13.3 Expert reviews

Lineform has received much praise for providing a simple, effective and enjoyable user interface for drawing vector graphics. It has been well received by both users and the press.

To provide the flavour of both user and press opinions, some quotes taken from reviews of Lineform are included below. These show a selection of different users' reactions to Lineform. These quotes, from both professionals and amateur users, support Lineform's successful design.

It is perhaps worth pointing out that professional reviewers rarely publish negative reviews: they simply don't get published often. One would therefore expect a bias towards positive reviews in any selection of reviews. However, in citing the reviews as contributing towards Lineform's evaluation, we are not using the scores the reviewers used, but their words and understanding. This use of the reviewers is very similar to expert usability evaluation, where trained usability professionals (rather than participants selected as representative of the target user population) are used to evaluate a system.

After years of messing around with Illustrator, then moving on to Freehand, then bouncing back to Illustrator, I was never able to do more than the most basic of tasks, and even then only with the help of a manual. Lineform has addressed this complexity issue with a simple interface that actually behaves the way you would expect it to, for the most part. If the Illustrator developers were concerned about how to make a certain function work, Thimbleby seemed more concerned with how the user would want it to work.

— Applelinks. (July 2007)

Lineform from Freeverse Software claims to be the solution for modern drawing and illustration. It is. Winner of a 2006 Apple Design Award, Lineform is not only easy to use, but the interface design makes the application so intuitive, Mac users need no explanation to start illustrating. Not only is it easy to use, it produces professional illustrations for less than \$80.

— CreativeMac (February 2007)

It's not often that you find a product you literally have to gush over... but Lineform, for me at least, is that product. I'm a graphic designer, t-shirt designs mostly, and I use Adobe Illustrator daily. I've never loved Illustrator, and I've REALLY never loved the \$499 price tag for it... but it has been necessary to do my job.

— AppleGazette (January 2007)

David Karlins reviewed Lineform for MacWorld, he is an author of half a dozen vector graphics books [Karlins and Hopkins, 2005a,b] and teaches Illustrating for San Francisco State University.

If you're looking for an easy-to-use, affordable vector drawing package that can create EPS and PDF files, it's hard to imagine a better deal than Lineform 1.3.2.

— David Karlins, MacWorld (May 2007)

Lineform has two other selling points. First, its speed: the program launches in a couple of seconds and shames Illustrator throughout in its responsiveness. Second, its ease of use. The simple interface alone makes it easier to find things.

— MacUser (October 2006, volume 22, issue 22)



When I say that Lineform offers a simplistic interface, or takes the easy route to giving me the tools to do the job in hand, this is really a positive. For many years now I have used Freehand instead of Illustrator, because the interface is a lot simpler to find your way around. Lineform is very similar to this, the interface is very clean and allows you to get on with what you are trying to achieve, but when you need more powerful tools, they are on tap too, but without being over-complicated (unlike some very expensive apps).

— Geekanoids (October 2006)

Darren Rolfe reviewed Lineform for MacReviewCast, he works as freelance artworker, designer and illustrator and in his own words is a “fully paid up member of the Adobe Illustrator Fan Club”. His review written from the view point of a heavy duty Illustrator user is full of comments such as “compared to Illustrator its an absolute joy to use”.

And the goodies just kept coming. Little things that have so obviously had some serious thinking time spent on them. I was pleasantly surprised!

[...]

This is a stunning piece of software. And if you are in the market for an viable alternative to the heavyweight option this is it!

Darren Rolfe, MacReviewCast (Dec 2007)

I found completing the necessary tasks to be surprisingly easy with Lineform. It took me approximately 45 minutes to learn the program (without any previous knowledge), find the tools I was looking for, and use them how I intended.

[...]

The program has a small learning curve. It's clear that Lineform programmers put in a great deal of effort to make it as simple and user-friendly as possible.

Epoch Times (July 2009)

## 13.4 Apple Design Award

Lineform won a 2006 Apple Design Award, the most prestigious awards for Mac software, recognising the best, most innovative Mac products, technical excellence and outstanding achievement on Mac OS X.

Thousands of applications are submitted each year and they are rigorously examined by a large number of judges. To win an award is a testament to Lineform's design, user interface and its underlying design principles.

## 13.5 Commercial success

Lineform has also been a commercial success, it is now available as a retail product, published by Freeverse Inc. It has sold over 10,000 copies since it was released in 2006.

Apple bought the rights to Lineform in 2008 and elements of Lineform are now a part of iWork which is Apple's suite of office applications. iWork includes presentation, word processing and spreadsheet applications: Keynote, Pages and Numbers. Lineform's users (from amateurs to professionals) and Apple (a big multinational corporation) bought Lineform for completely different reasons, but were motivated by the same high quality of design and implementation that Lineform represents.

## 13.6 Cognitive evaluation

Green's [1989, 2000] cognitive activities and cognitive dimensions allow a heuristic evaluation, similar to the one that was performed on the calculator. The table below exhibits Green's original cognitive dimensions, using them to compare Lineform and Adobe Illustrator.

Cognitive Dimension	Illustrator	Lineform
Viscosity	medium	low
Visibility	medium	high
Premature commitment	high	none
Hidden dependencies	none	none
Role-expressiveness	medium	medium
Error-proneness	low	low
Abstraction	medium	none

**Table 13.1: A comparison of drawing applications using Green's cognitive dimensions framework**

Change is how drawings are created, a *viscous* drawing application that makes change hard would be a complete failure. Thus both Adobe Illustrator and Lineform provide many different ways and tools to change and interact with the drawing. Neither user interface is *viscous*, they are both fluid interfaces that allow and encourage change. Although Illustrator does not have the same principle of *appropriate controls* and sometimes provides awkward discrete controls for continuous values. This can be seen in Illustrator's stroke palette, shown in Figure 13.2, which only provides discrete stepper controls or pop-up menu for the stroke width which is a continuous value. These inappropriate controls detract from the fluidity of the user interface, making it harder for the user to change the stroke width value and explore the results. Illustrator's *premature commitment* also reduces

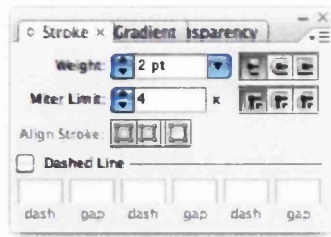


Figure 13.2: Adobe Illustrator CS3's stroke palette

the fluidity of its user interface.

*Visibility* describes how well a user interface makes the information a user needs available. Lineform and Illustrator are both graphical applications and provide a WYSIWYG interactive view of the document as well as further outline modes and layer palettes. However Illustrator does not provide *projected* feedback for several interactions and this severely reduces visibility whilst the user is interacting and thus makes it much harder for the user to get the desired result. Examples of this are the outline resizing, shown in Chapter 8 Figure 8.8, and the gradient tool which draws a single line as feedback instead of drawing the result of the gradient.

*Premature commitment* was one aspect of drawing applications that Lineform specifically avoids and is codified in the *flexible design* principle. Illustrator often determines the role of an object at the point of creation, for example whether an object is a text-box, image, stamp, shape or mesh is determined by the tool that created it. Later if the user changes their mind it is nearly impossible to change the role of these objects. In contrast Lineform lets the user change the role of any object freely without restriction.

*Hidden dependencies* occur when important links between entities are not visible. Neither application has any dependencies, each shape or object on the canvas is independent of any other shape.

The direct manipulation graphical WYSIWYG user interfaces mean that both applications provide a *role expressive* interface on the canvas. However the tools and concepts of a drawing application are fairly specialised and for a novice the role of many tools like the Bézier pen or eye-dropper might not be immediately discernible. These concepts that the user has to learn reduces the *role expressiveness* of both user interfaces,

*Error proneness* is not a big problem for either application. The visual and direct manipulation user interfaces means that mistakes are rarely made. However Illustrator's lack of *projection* when using certain tools means that temporary mistakes are made as the user attempts to achieve a desired effect by a trial and error exploration. Lineform instead provides a *projected* exploration that presents immediate and continuous feedback which reduces any intermediate errors.

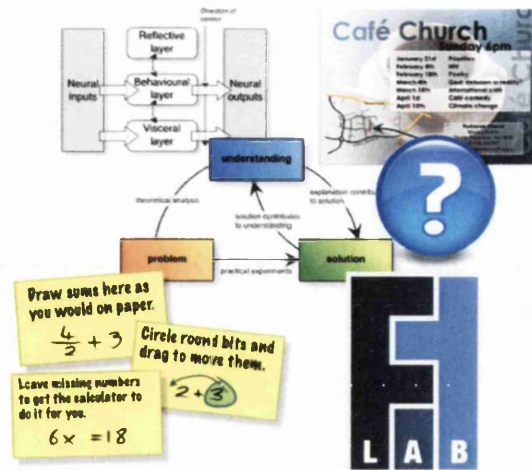


Figure 13.3: Several examples of my use of Lineform

Illustrator's multiple shapes and tools that operate only on a specific class are complex layers of *abstraction* which can confuse users. Lineform's lack of specialised classes of shapes or tools removes the potential for confusion.

### 13.7 Summary

Lineform is now used by thousands of people, from professionals to home users, in order to fulfil a huge variety of different needs.

Figure 13.3 shows some more drawings that Lineform has been used to create. It is the creation of these sort of drawings that motivated the creation and design of Lineform. Figure 13.3 includes examples of diagrams, flyers, buttons, logos and the help graphics for the calculator. Almost all diagrams and drawings in this thesis were created using Lineform.

Vector graphics are fairly complex, and vector drawing applications require many different abilities or features so that users can create and edit graphics. Well designed drawing applications that meet these requirements can allow powerful and complex drawings to be created easily. Lineform succeeds in doing so, its user interface incorporates all the required features that were previously identified, and does so in a way that is evidently clear and easy to use. The quotes of users and reviewers, and indeed its commercial success with thousands of users, testify to Lineform's user interface design, and in turn to the design principles that guided its development.



## Chapter 14

# Conclusions

The user interfaces, designs, and design principles of two new systems has been described. Both systems are effective, as evidenced by a variety of empirical evaluation including expert reviews, as well as argument and appeal to the literature. They are now both successful commercial products. Both were designed using a similar principle-driven processes. Both have produced novel user interface ideas. And both now open up interesting opportunities for future research.

This chapter concludes the thesis with a summary of the primary principles that are the key components of the designs of these user interfaces, followed by an overview of further research, which could extend and build on the novel contributions of the thesis.

If we intend a science of human-computer interaction, it is essential that we have principles from which to derive the manner of the interaction between person and computer. It is easy to devise experiments to test this idea or that, to compare and contrast alternatives, or to evaluate the quality of the latest technological offering. But we must aspire to more than responsiveness to the current need. The technology upon which the human-computer interface is built changes rapidly relative to the time with which psychological experimentation yields answers. If we do not take care, today's answers apply only to yesterday's concerns.

— Donald Norman [1983]

### 14.1 Contributions

While the programs, description of the design process, and novel user interfaces are themselves contributions and are valuable for future design, it is the principles, their relationship to the development process, and their potential future role in interactive system development, that form the lasting

contributions of the thesis.

Norman [1983] states that principles are essential, in order to apply today's answers to tomorrow's problems. The principles which resulted in the novel and effective interfaces of the calculator and of Lineform, also allow the same constructive user interface design ideas to be applied to future designs.

Although principles in general are essential, that does not mean the principles described in this thesis are. Nor that they are uniquely important or special. They were however a major factor in the design of both systems and the four *flow* principles were key components in both designs. Crucially, the two different applications, the calculator and Lineform, allow us to talk about the principles and their consequences on design in a concrete way.

### 14.1.1 Utility

While the principles were not followed unthinkingly, they were utilised almost without exception, Chapters 3 and 9 describe how they were used and when they were ignored. The designs of the systems are consistent and logical in their approach to problems; without implementation the principles consistently the resulting designs would not be as coherent nor work as well. For users, this consistency might not be immediately apparent but (we have argued) it leads to an ease of use that would otherwise have been missing.

Although the conception of these principles has been illustrated in specific designs, their application and utility is general and extensive. The principles described in this thesis are not only important for these two user interfaces, but can provide relevant guidelines and ideas for future user interface design. The underlying design process is also an important factor of consideration. To fully utilise these principles, it is key to have a design process that ensures their consistent and pervasive application.

The use of the *flow* principles when designing Lineform highlights their utility in a contrasting type of user interface to the calculator. The principles need not be used in the same way as how they were used in the designs in this thesis. Each principle may vary in relevance to a particular design but still be useful to consider — just as *continuity* is less relevant to Lineform's design but it still provides useful insight. Also in the same way as Lineform's design generated new principles, more applicable to its purposes, so to would future principle-driven designs. The discussion and insights that principles focus, may be as useful as the principles themselves.

### 14.1.2 Validity

User studies, qualitative feedback, heuristic, and analytic evidence of the effectiveness of both user interfaces is found in Chapters 7 and 13. The principles themselves have not directly been subject to any empirical testing

outside of their impact on the design of those specific applications. However, the principles are an important aspect of the design of the novel user interfaces described, and this thesis has argued that they are the primary reason that both user interfaces have been effective.

The design and development of each application built on the synergy that formed between the principles and the application designs. The applications and principles developed together, to such an extent that they are inseparable. This makes evaluation complex, because without the principles the applications are incomplete husks. The principles provide the core structure of how the user interfaces of both applications work.

## 14.2 Principles

Principles were developed during the design and development process of both the calculator and Lineform, and each main principle is summarised in this section.

The calculator was designed using the *flow* principles of *projection*, *continuity*, *WYSIWYE* and *declarative interaction*. Each of these principles relates to an aspect of the user's interaction with an interactive computer system. Each of the four *flow* principles is focused on a different component of interaction and together they combine to describe a system that as a whole engenders flow.

Lineform's development incorporated these principles in a manner that was more appropriate to its domain, and built up some more narrowly-focused principles of its own. These principles of *physical modes*, *flexible design* and *appropriate controls* developed during the implementation of Lineform.

All these principles are described fully in Chapters 4 and 10.

### 14.2.1 Projection

*Projection* could be described as “consistent and immediate changes everywhere.” Inconsistency in a user interface requires a cognitive effort to keep track of, and can cause users problems when it is not clear which values are valid.

A *projected view* is a view that is projected outwards from the data and has no state of its own; as the data changes so does the view. Projected views are updated immediately and continuously as the underlying data being projected changes, thus multiple projected views of the same data will always be consistent. Similarly, editing a projected view immediately changes the underlying data, and provides an immediate response to the user's edits in other views of the data. This can make a user interface



easier and faster to use because the user is not burdened with remembering inconsistencies and is provided with immediate feedback of any edits.

The calculator provides projected editing of mathematical expressions. It always updates and accommodates new input as the user writes new symbols. The handwriting recognition is immediately reflected in the canvas, there is no need to press a button to get the answer, and the answer is updated immediately to reflect the new input. What the user sees is *always* mathematically consistent. Lineform also *projects* all its data ensuring that no part of the user interface is ever inconsistent.

Another example of *projection* is “search as you type.” In these user interfaces, the search box and search result are multiple views of the same data. As the user types, these different views are always kept consistent, and the results are immediately updated as the user edits search terms. A normal search box provides old and inconsistent search results for different search terms, until the user presses return.

### 14.2.2 Continuity

When a user interface changes state without *continuity* there is a sudden visual change that can confuse or mislead a user. A user interface that provides continuity uses animation and morphing to provide the user with the clues to follow what is happening and to enable them to mentally join up the state changes.

Morphing and *continuity* form a large part of the user’s experience of the calculator. Every time symbol or expression recognition occurs the user interface morphs the current input into a typeset expression and displays an answer. This morphing not only provides *continuity* but also enjoyable physical movement. The continuous feedback and smooth morphing the calculator uses provides the user with a clear idea about what is happening and a consistent linking between input and output. Continuity enhances the user’s appreciation of the calculator’s use of *projection*, because they see it working.

For example, the user’s hand-written input is morphed into a typeset sum, which provides a clear link between mathematics being calculated and how it relates to the user’s hand-written input. This morphing not only makes the calculator easier to use but is also very visual and enjoyable.

### 14.2.3 WYSIWYE — What you see is what you edit

Users should only interact with what they can see. Hidden state and structure means that user input can have unexpected consequences, thus causing a frustrating experience and causing the user trouble. A What You See

Is What You Edit (WYSIWYE) user interface is one in which there is no hidden state or constraints that affect how the user interacts with it.

A WYSIWYE user interface is predictable, the entire underlying model that affects the user is visible to them. The user can see anything that affects the result of an action and therefore can (after learning the user interface) predict what any action will do. A WYSIWYE interface has very few constraints on how the user can edit, and those that do exist, are visually obvious to the user.

The calculator is a good example: what the user sees — digits and symbols — is *exactly* what they can edit. The user can edit any expression by dragging, adding, or deleting the symbols, without being concerned about unexpected results or modes or constraints. In contrast, most other two-dimensional mathematical editors restrict the user's actions to the implicit, but invisible, underlying application structure.

#### 14.2.4 Declarative interaction

Declarative interfaces blur the distinction between input and output. Often input and output in a user interface are entirely and conceptually separate, which means that if a user wants to change the output in a certain way they have to work out how to change the input to affect the desired result.

A declarative user interface aims to allow the user to edit most views of the data, whether 'input' or 'output' views. By not distinguishing between input and output, the user interface can allow for more powerful and intuitive interactions. An example of this is how a user can, by providing the desired output data discover an example of the input needed, a sort of *interaction by example*. The user is able to incrementally explore or construct an answer by cycling between 'input' and 'output,' adding changes in whichever way makes most sense. Declarative interfaces also facilitate exploration of the underlying process and allow users to gain a deeper understanding of the input-output relationship.

A declarative calculator treats input and output equally, such that the user can change either, and the other will be altered to ensure the expression is mathematical correct. This means the user can solve simple mathematical expressions by editing both sides of the equality, including the output. Examples of expressions the user could solve directly are ' $2+3$ ', ' $4 \times ? = 24$ ' and ' $\sqrt{?} = 100$ '. Both sides of the equality are treated equally and the calculator solves the expression so that it is mathematically correct.

In Lineform, the user can drag out a shape to roughly the right size in the canvas, then tweak the exact measurements by entering or editing numbers in the Transform inspector, then the user can reposition the shape in a better location on the canvas. Each view, the inspector and the canvas, lends itself to a different style of use, and neither view need be used solely for output

or input. The combination of both is powerful and gives the user flexibility. In Lineform, the views are also *projected*, thus the different views are never inconsistent or confusing.

### 14.2.5 Physical modes

Modes in user interfaces alter what a single action achieves. Mode confusion, where the user is unaware of what mode they are in, can cause problems when an interaction gets an unexpected result. We define *physical modes* as modes that are maintained by some continuous physical action, for example holding down the Shift key to type capital letters. There is a mode (upper or lower case shifting of letters, in this case), but the user has to be engaged in a particular physical action to be in a particular mode.

*Physical modes* provides a much better reminder and awareness of the current action being performed than a state or action that is indicated outside of the user's body. Using a physical mode instead of 'virtual' visible or audible mode identifiers, such as icons on the screen, reduces the possibility of mode confusion.

Lineform makes extensive use of physical modes throughout to control the action of different tools. These make it easy to switch modes quickly without any confusion. Furthermore, if the user takes their hands off the keyboard, all modes reset without the user having to know which mode was which or what state the user interface was in. This is a significant simplification over non-physical mode approaches.

### 14.2.6 Flexible design

*Flexible* user interfaces allow users to delay decisions until they are ready to make them and then to easily change their mind. Compared to rigid interfaces this provides a much more enjoyable user experience. Rigid user interfaces enforce unnecessary premature specification that often needlessly restrict the user.

Lineform provides a flexible user interface where any shape or object can be repurposed for any use at any time. This is in contrast to (for example) Adobe Illustrator that provides specialised shapes, that once created are very hard to change. Lineform's *flexible design* allows users to be free to draw and design without the fear of future constraints based on the initial choices.

### 14.2.7 Appropriate controls

Using right user interface controls for the right values makes a huge difference. The right control can enable the user to easily and quickly manipulate

the underlying values.

Discrete values should be controlled through discrete controls and continuous values should be controlled through continuous controls. Continuous values should also have an exact discrete way of being set. Every control should also provide continuous *projected* interaction.

If a user interface provides a discrete control for a continuous value, then the user's interaction when changing that value is limited. By always providing continuous controls for continuous values, the user's interaction is not restricted. Discrete controls are important for discrete values and also for continuous values where setting an exact value is important.

Lineform provides sliders, which are a continuous control, for its continuous values. Most of these values can also be set to exact values using a discrete control, like a text input box. Further, in Lineform, these controls are *projected* and they allow the user to explore the range of possible values quickly, because the user can immediately see the effect of the range of possible values on the canvas.

## 14.3 Further work

The *flow* principles of the calculator and the principles drawn out from Lineform's design suggest many new and interesting ideas for further development.

Instead of going into great detail about possible further research, the remainder of this chapter is structured to provide short sections of potential further research, organised by research area. The next sections cover various aspects of the research and how the ideas could be built on and extended in each research domain.

### 14.3.1 Computer science

Using the principles of *projection* and *declarative interaction*, the calculator corrects any input so that it is mathematically correct. It attempts to do this in a manner that causes the least disruption to the user. There are however many possible ways that calculations can be corrected. Questioning the calculator's particular implementation that attempt to show a valid equation at all times raises interesting questions.

- How can ambiguous equations with multiple unknowns be corrected to valid equations?
- What are the least disruptive corrections that can be made to equations?

The calculator's approach could also be applied to other domains, for instance, Boolean logic. In general, extending the mathematics of the calculator poses both interesting problems algorithmically and in the appropriate styles of user interaction.

- How can the fluid correctness of the calculator be extended to more complicated maths, from simple algebra to completely different domains?
- Is it possible to provide the same principle-based correcting mechanism consistently in these areas?

The current calculator can be put in a mode to conceal an answer, as in  $4 + \square = 14$ , with the box not automatically filled in. This idea raises further design tradeoffs, which still offer interesting exploration; for example, if the user wrote  $4 + 1 = 14$ , should the display become  $4 + 1 + \square = 14$  or should it be  $4 + 1\square = 14$ ?

There are also other mathematical problems, such as what should the calculator do with  $\frac{1}{0}$ . It is not clear what the best solution is.

- How should the grammar be extended to other domains, and what applications are there that could exploit the user interface?

One of the potential ways in which the principles could be explored is embedding the calculator in other user interfaces: wherever a number is needed, it could be entered or manipulated with a calculator-style user interface, applying the combination of the flexibility of editing and the automatic corrections to user interfaces.

Another approach is generalising the syntax correction. For example, a possible use is in code editing, where the approach could provide some of the benefits of syntax directed editing without the major disadvantages in how editing code is restricted.

- In what other user interfaces does automatic correction (e.g., *projection*) in a constrained environment work?
- How could different forms of input (for example, diagrams) be corrected?

WYSIWYE is a powerful way of interacting, but in this thesis there were limitations in how it was implemented: Lineform provides a very structured interaction and the calculator editing operates only at the symbol level. The WYSIWYE nature of both applications could be taken further.

- How could WYSIWYE editing work at the sub-symbol level (e.g., drawing over a 3 to change it into an 8, or drawing over a  $-$  so it can be changed to  $+$  which can then be changed to 4)?
- How can the benefits of bitmap drawing (with its easy WYSIWYE) be combined with vector drawing (flexibility and editability)?

- The current calculator is numeric; how can it be extended to symbolic maths. Is there a continuum?

A solution to this last question would help those users who very quickly create calculations like  $10^{10^{10}}$  or  $100000!$  that cause overflow. Such expressions are currently evaluated numerically in Java floating point numbers, and then fail due to overflow. Instead, they could be retained symbolically and hence avoid any interruption in the user interface flow. Potentially, symbolic solutions would also permit  $1/0$  and other ‘errors’ to be handled gracefully.

### 14.3.2 Human computer interaction

The main area in which this thesis could be extended by usability researchers would be further evaluation and user studies. In particular, longitudinal studies in primary schools could provide many useful and interesting insights, and there are a lot of useful questions that could be answered by such studies ...

- Does the novelty of the morphing user interface wear off?
- Would students enjoy using the calculator every day, for the maths they do in class? (Maths in class is ‘work’ whereas all evaluation in this thesis might be accused of evaluating ‘play.’)
- Does a pen-based user interface improve with practice or is it awkward and slow compared to keypad entry?
- Do users begin to make use of the declarative aspects of the calculator rather than rearranging equations before they enter them?

Users certainly enjoy using the calculator. Children like the calculator, and it is certainly more powerful, easier and more reliable to use than any other. The appeal of proprioception, gesture and affordance all potentially play a part. A key part of learning is exploration, by surfacing the rules so that a user can interact with them a declarative interface can potentially support easier exploration. However, longer-term (longitudinal) experiments will be needed to see whether the fun and other benefits persist.

- Is the user’s “fun” a surprise due to the unfamiliar nature of the user interface, or is it durable?
- How can declarative user interfaces, such as the calculator, encourage exploration and learning?
- Subsequently, how can experiments and user studies rigorously examine the process of exploration?

Carefully constructed user studies could better inform us of the utility of the different principles developed both in the calculator’s and in Lineform’s

design.

- Are the principles generally useful, and, if so, where are they effective?
- What are the best ways to provide *continuity* to the user?
- How do the principles work in other user interfaces?

There are still many interesting questions to be answered in the domain of drawing and specifically in vector graphics.

- Are there easier ways to draw curves than the ubiquitous Bézier curve? Can the Bézier curve control points be simplified in some way?
- What are useful interactions and operations for drawing?
- How can we provide an experience that has the preciseness of computer vectors *and* the ease of sketching with a pencil?

### 14.3.3 Interactive editing

This thesis explores in detail just two applications and their relation to the design principles. In fact, a third application was also developed: Recdit is an example of an application that was inspired by the design of the calculator, specifically its ‘undo’ clock (see Appendix H). Recdit is a text editor that records the entire history of a document, and provides a timeline-like interface for scrubbing through the creation process of the document. The majority of this thesis was written using Recdit, and the graphs describing the creation of this thesis, generated by Recdit, are provided in Appendix H, along with a draft paper on the editor.

### 14.3.4 Teaching

Currently the calculator provides the ability to hide and show calculated answers. These let a teacher write an equation on an interactive board for a class to see, whilst hiding the answer. Teachers have mentioned that it helps to have the answer calculated for them: it means that where if math skills are often rusty they are more prepared and more confident to answer questions and explore the answers with the students.

- How is the calculator used by teachers to provide richer teaching of mathematics?
- How can its features lead to richer teaching of mathematics? And how effective would it be?
- How can it be extended to facilitate use by teachers in novel and interactive teaching methods?

The calculator was designed purely as a calculator, to do arithmetic sums. In this sense, the goal was to make it natural and easy to use. Those positive attributes have led teachers to strongly encourage us to make it more effective in the educational setting. In this context, children are learning not just arithmetic but the mathematical notation itself — for example children may be learning Arabic numerals. Typically, an educational device is wanted, not a calculator. For example, teachers have wanted ‘drills’ of various kinds, rather than an open-ended tool.

- How can the calculator, or similar user interfaces, be developed to support disabilities?

A further-developed calculator might provide or support numerical games; it might do tests and provide assessment; it might provide features for children with specific learning difficulties; and so on.

Many fun exercises are of the form “only use fives to make the number 30” (some answers are  $5 \times (5 + 5/5)$ ,  $55 - 5 \times 5$ ,  $5 \times 5 + 5$  and so on). The calculator could show a legitimate calculation in green (say), and any ‘cheating’ (in this case, using other digits) in blue. The user’s goal is now to make 30 with an all-green calculation.

### 14.3.5 Learning

The calculator poses interesting questions about how to enable the ability to explore mathematics whilst the visible calculation remains mathematically correct affects learning. The ability to explore different mathematical expressions while the calculator ensures correctness is potentially one of the most effective tools for learning. Exploring this in use, and the potential of this approach in the calculator and other user interfaces, offers a lot of potential, both for understanding mathematical skills and for developing or extending the calculator.

- Is allowing students to explore mathematics beyond the level that they have been taught a good idea?
- Does the ability of the calculator to ensure correct mathematics all the time help in exploration, for instance in enhancing confidence?
- How can the calculator encourage exploration and learning?

## 14.4 Summary

The development of novel user interfaces, a new calculator and Lineform using a principle-driven process, has generated and refined useful principles as well as raising many interesting questions for further work. These underlying design principles have been key to the design of these systems. The



same principles now provide the opportunity for their use in future user interface design.

There are many examples of worthwhile further work that extends the work done in this thesis. In particular more thorough evaluations of both the systems in longitudinal studies and of the principles' efficacy would be valuable. Although much further work now seems useful, arguably little of it would have been considered but for the critical development of principles and the development of the calculator and the drawing program.

It is hoped that the design of new user interfaces can use and build on the work of the *flow* principles, and in doing so extend and validate them further, as well as raise even more interesting questions.

# Bibliography

Adobe (2008). Document management — portable document format part 1: Pdf 1.7.

Ahlberg, C. and Shneiderman, B. (1994). Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 313–317, New York, NY, USA. ACM Press.

Ahlberg, C., Williamson, C., and Shneiderman, B. (1992). Dynamic queries for information exploration: An implementation and evaluation. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 619–626, New York, NY, USA. ACM.

Anderson, R. H. (1968). Syntax directed recognition of hand-printed two-dimensional mathematics. *Interactive Systems for Experimental Applied Mathematics*, pages 436–459.

Anderson, R. H. (1977). *Two-dimensional mathematical notation*, pages 147–177. Springer-Verlag.

Anthony, L., Yang, J., and Koedinger, K. R. (2005). Evaluation of multi-modal input for entering mathematical equations on the computer. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1184–1187, New York, NY, USA. ACM Press.

Arvo, J. and Novins, K. (2000). Fluid sketches: continuous recognition and morphing of simple hand-drawn shapes. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 73–80, New York, NY, USA. ACM.

Avitzur, R. (1988). *Milo (a Macintosh program)*. Paracomp Inc., San Francisco, CA, USA.

Avitzur, R. (1998). *Direct manipulation in a mathematics user interface*, pages 43–60. Springer.

Baecker, R. M. and Buxton, W. A. S. (1987). *Readings in Human-Computer Interaction: A Multidisciplinary Approach*. Morgan Kaufmann Publishers, Los Altos, CA.

- Baecker, R. M., Grudin, J., Buxton, W. A. S., and Greenberg, S. (1995). *Readings in Computers and Human Interaction: Toward the Year 2000, 2nd edition*. Morgan Kaufmann Publishers, Los Altos, CA.
- Barkhuus, L. and Rode, J. A. (2007). From mice to men – 24 years of evaluation in CHI. In *ACM CHI07 – Alt.CHI*.
- Baudelaire, P. and Gangnet, M. (1989). Planar maps: an interaction paradigm for graphic design. *SIGCHI Bulletin*, 20(SI):313–318.
- Ben-Kiki, O., Evans, C., and Ingerson, B. (2004). Yaml specification 1.1.
- Bezier, P. (1972). *Numerical Control; Mathematics and Applications*. John Wiley & Sons, London, UK.
- Blostein, D. and Grbavec, A. (1996). *Recognition of Mathematical Notation*, chapter 22. World Scientific Publishing Company.
- Blostein, D. and Schüerr, A. (1999). Computing with graphs and graph transformation. *Software Practice and Experience*, 29(3):1–21.
- Boeve, E., Barfield, L., and Pemberton, S. (1993). *WYSIWYG editors: And what now?*, volume 753, pages 68–82. Springer Berlin.
- Brown, C. M. L. (1988). Comparison of typing and handwriting in “two-finger typists”. In *Proceedings of the Human Factors Society*, pages 381–385.
- Buxton, B. (2007). *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann.
- Cairns, P., Wali, S., and Thimbleby, H. (2004). Evaluating a novel calculator interface. In Watts, A. D. . L., (Ed.), *Proceedings BCS HCI Conference*, volume 2, pages 9–12. Research Press International.
- Cajori, F. (1993). *A history of mathematical notations*. Courier Dover Publications.
- Card, S. K., Thomas, T. P., and Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates.
- Chan, K.-F. and Yeung, D.-Y. (1998). A simple yet robust structural approach for recognizing on-line handwritten alphanumerical characters. In *Proceedings of the sixth international workshop on frontiers in handwriting recognition*, pages 229–238.
- Chan, K.-F. and Yeung, D.-Y. (2000). Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15.
- Chang, B.-W. and Ungar, D. (1993). Animation: from cartoons to the user interface. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 45–55, New York, NY, USA. ACM.

- Chang, S. (1970). A method for the structural analysis of two-dimensional mathematical expressions. *Information Sciences*, 2(3):253–272.
- Chou, P. A. (1989). Recognition of equations using a two-dimensional stochastic context-free grammar. In *Proceedings SPIE Conference on Visual Communications and Image Processing*, pages 852–863, Philadelphia, PA.
- Clapp, L. C. and Kain, R. Y. (1963). A computer aid for symbolic mathematics. In *AFIPS '63 (Fall): Proceedings of the November 12-14, 1963, fall joint computer conference*, pages 509–517, New York, NY, USA. ACM.
- Cockburn, A. and Bryant, A. (1996). *Do it This Way: Equal Opportunity Programming for Kids*, pages 246–251. IEEE Computer Society, Washington, DC, USA.
- Csikszentmihályi, M. (1990). *Flow: The Psychology of Optimal Experience*. Harper and Row, New York.
- Cypher, A., (Ed.) (1993). *Watch What I Do: Programming by Demonstration*. The MIT Press, Cambridge, MA, USA.
- Dix, A., Finlay, J., Abowd, G., and Beale, R. (1997). *Human-Computer Interaction*. Prentice Hall.
- Engelman, C. (1965). Mathlab: a program for on-line machine assistance in symbolic computations. In *AFIPS '65 (Fall, part II): Proceedings of the November 30–December 1, 1965, fall joint computer conference, part II: computers: their impact on society*, pages 117–126, New York, NY, USA. ACM.
- Eto, Y. and Suzuki, M. (2001). Mathematical formula recognition using virtual link network. In *ICDAR '01: Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 762–767, Washington, DC, USA. IEEE Computer Society.
- Faure, C. and Wang, Z. X. (1990). *Automatic perception of the structure of handwritten mathematical expressions*, pages 337–361. World Scientific, Singapore.
- Fleetwood, M. D., Byrne, M. D., Centgraf, P., Dudziak, K. Q., Lin, B., and Mogilev, D. (2002). An evaluation of text-entry in palm os - graffiti and the virtual keyboard. In *Human Factors and Ergonomics Society Annual Meeting Proceedings*, pages 617–621.
- Fuegi, J. and Francis, J. (2003). Lovelace & babbage and the creation of the 1843 'notes'. *IEEE Annals of the History of Computing*, 25(4):16–26.
- Fujimoto, M. and Suzuki, M. (2002). A handwriting interface to various computer algebra systems via openxm framework. In *Proceedings of the RIMS Workshop, Applications of Computer Algebra Conference*, volume 1335.

- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. M. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.
- Gould, J. and Lewis, C. (1985). Designing for usability: Key principles and what designers think. *Communications of the ACM*, 28(3):300–311.
- Grbavec, A. and Blostein, D. (1995). Mathematics recognition using graph rewriting. In *ICDAR '95: Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1, page 417, Washington, DC, USA. IEEE Computer Society.
- Green, T. R. G. (1989). Cognitive dimensions of notations. In *Proceedings of the fifth conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and computers V*, pages 443–460, New York, NY, USA. Cambridge University Press.
- Green, T. R. G. (2000). Instructions and descriptions: some cognitive aspects of programming and similar activities. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, pages 21–28, New York, NY, USA. ACM Press.
- Greenberg, S. and Buxton, B. (2008). Usability evaluation considered harmful (some of the time). In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 111–120, New York, NY, USA. ACM Press.
- Grossman, T., Balakrishnan, R., and Singh, K. (2003). An interface for creating and manipulating curves using a high degree-of-freedom curve input device. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 185–192, New York, NY, USA. ACM.
- Hearn, A. C. (1968). *REDUCE: A user-oriented interactive system for algebraic simplification*, pages 79–90. Academic Press, New York.
- Hsu, S. C., Lee, I. H. H., and Wiseman, N. E. (1993). Skeletal strokes. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 197–206, New York, NY, USA. ACM.
- Johnson, J. (1985). Calculator functions on bitmapped computers. *SIGCHI Bulletin*, 17(1):23–28.
- Joyner, D. (2006). OSCAS: maxima. *ACM Communications in Computer Algebra*, 40(3-4):108–111.
- Kajler, N. and Soiffer, N. (1998). A survey of user interfaces for computer algebra systems. *J. Symb. Comput.*, 25(2):127–159.
- Karlins, D. and Hopkins, B. K. (2005a). *Adobe Illustrator CS2 Gone Wild*. Wiley.

- Karlins, D. and Hopkins, B. K. (2005b). *Adobe Illustrator CS2 How-Tos: 100 Essential Techniques*. Adobe Press.
- Kasuya, Y. and Yamana, H. (2007). Mathbox: interactive pen-based interface for inputting mathematical expressions. In *IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces*, pages 274–277, New York, NY, USA. ACM Press.
- Khwaja, A. A. and Urban, J. E. (1993). Syntax-directed editing environments: issues and features. In *SAC '93: Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing*, pages 230–237, New York, NY, USA. ACM Press.
- Knuth, D. (1984). *The TEXbook*. Addison Wesley.
- Landauer, T. K. (1995). *The Trouble with Computers: Usefulness, Usability, and Productivity*. MIT Press, Cambridge, MA, USA.
- Lasseter, J. (1987). Principles of traditional animation applied to 3d computer animation. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 35–44, New York, NY, USA. ACM.
- LaViola, J. J. (2006). An initial evaluation of a pen-based tool for creating dynamic mathematical illustrations. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling*.
- LaViola, J. J. and Zeleznik, R. C. (2004). Mathpad2: a system for the creation and exploration of mathematical sketches. *ACM Transactions on Graphics*, pages 432–440.
- Lavirotte, S. and Pottier, L. (1997). Optical formula recognition. In *Proceedings 4th International conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 357–361.
- Lee, H.-J. and Wan, J.-S. (1995). Design of a mathematical expression recognition system. In *ICDAR '95: Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 2)*, pages 1084–1087, Washington, DC, USA. IEEE Computer Society.
- Leler, W. and Soiffer, N. (1985). An interactive graphical interface for reduce. *ACM SIGSAM Bulletin*, 19(3):17–23.
- Lewis, C. and Rieman, J. (1993). Task-centered user interface design: A practical introduction.
- Lewis, J. R. (1999). Input rates and user preference for three small-screen input methods: Standard keyboard, predictive keyboard, and handwriting. In *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting*, pages 425–428.
- Li, Y., Hinckley, K., Guan, Z., and Landay, J. A. (2005). Experimental analysis of mode switching techniques in pen-based user interfaces. In

- CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 461–470, New York, NY, USA. ACM.
- Lieberman, H. (1993). *Tinker: a programming by demonstration system for beginning programmers*, pages 49–64. MIT Press, Cambridge, MA, USA.
- Lieberman, H. (2003). The tyranny of evaluation. In *ACM CHI Fringe*.
- Littin, R. (1993). The pen input of mathematical expressions. Master's thesis, University of Waikato.
- Long, A. C., Landay, J. A., and Rowe, L. A. (1998). Pda and gesture uses in practice: Insights for designers of pen-based. In *Technical Report: CSD-97-976*, Berkeley, CA, USA. University of California at Berkeley.
- Lunney, T. F. and Perrott, R. H. (1988). Syntax-directed editing. *Software Engineering Journal*, 3(2):37–46.
- Martin, W. A. (1967a). A fast parsing scheme for hand-printed mathematical expressions.
- Martin, W. A. (1967b). *Symbolic Mathematical Laboratory*. Massachusetts Institute of Technology, Cambridge, MA, USA.
- Martin, W. A. (1971). Computer input/output of mathematical expressions. In *SYMSAC '71: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pages 78–89, New York, NY, USA. ACM Press.
- Martin, W. A. and Fateman, R. J. (1971). The macsyma system. In *SYMSAC '71: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pages 59–75, New York, NY, USA. ACM.
- McLeod, D. (1976). The translation and compatibility of sequel and query by example. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 520–526, Los Alamitos, CA, USA. IEEE Computer Society Press.
- Microsoft (1993). *Microsoft Word User's Guide, Version 6.0*. Microsoft Press.
- Minsky, M. (1963). Mathscope part i: A proposal for a mathematical manipulation- display system. *Technical Report MAC-M-118 Artificial Intelligence Project*.
- Monk, A. (1986). Mode errors: a user-centered analysis and some preventative measures using keying-contingent sound. *International Journal of Man-Machine Studies*, 24(4):313–327.
- Nielsen, J. and Mack, R. (1994). *Usability Inspection Methods*. John Wiley & Sons.
- Norman, D. (1981). Categorization of action slips. *Psychology Review*, 88(1):1–15.

- Norman, D. (1983). Design principles for human-computer interfaces. In *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 1–10, New York, NY, USA. ACM.
- Norman, D. (1987). *Some Observations on Mental Models*, pages 241–244. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA.
- Norman, D. (1988). *The Design of Everyday Things*. MIT Press, Cambridge, MA, USA.
- Odata, K., Arakawa, H., and IsaoMasuda (1982). On-line recognition of handwritten characters by approximating each stroke with several points. *IEEE Transactions on Systems, Man, and Cybernetics*, 12(6):898–903.
- Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., and Salesin, D. (2008). Diffusion curves: a vector representation for smooth-shaded images. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–8, New York, NY, USA. ACM.
- Pavlidis, I. T., Singh, R., and Papanikolopoulos, N. P. (1996). Recognition of on-line handwritten patterns through shape metamorphosis. In *ICPR '96: Proceedings of the International Conference on Pattern Recognition*, volume 3, pages 18–22, Washington, DC, USA. IEEE Computer Society.
- Persoon, E. and Fu, K. S. (1977). Shape discrimination using fourier descriptors. *IEEE Transaction on Systems, Man and Cybernetics*, 7(3):170–179.
- Plamondon, R. and Srihari, S. N. (2000). On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84.
- Popper, K. R. (1963). *Conjectures and Refutations: The Growth of Scientific Knowledge*. Routledge.
- Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. (1994). *Human-Computer Interaction: Concepts And Design*. Addison-Wesley.
- Quill, U. (1999). Introduction to lyx: Make working with latex easier using the wysiwyg editor lyx. *Linux Journal*.
- Raskin, J. (2000). *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley.
- Reenskaug, T. (1979). Thing-model-view-editor an example from a planning system.
- Reiss, S. P. (1984). Graphical program development with pecan program development systems. *SIGSOFT Software Engineer Notes*, 9(3):30–41.
- Runciman, C. and Thimbleby, H. (1986). Equal opportunity interactive systems. *International Journal of Man-Machine Studies*, 25(4):439–451.



- Schuler, D. and Namioka, A., (Eds.) (1993). *Participatory design: Principles and practices*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA.
- Sellen, A. J., Kurtenbach, G., and Buxton, W. (1992). Prevention of mode errors through sensory feedback. *Human-Computer Interaction*, 7(2):141–164.
- Shaw, A. C. (1969). The formal picture description scheme as a basis for picture processing systems. *Information and Control*, 14:9–52.
- Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69.
- Shneiderman, B. (1992). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley.
- Shneiderman, B. (1997). Direct manipulation for comprehensible, predictable and controllable user interfaces. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 33–39, New York, NY, USA. ACM Press.
- Simonyi, C., Christerson, M., and Clifford, S. (2006). Intentional software. In *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 451–464, New York, NY, USA. ACM Press.
- Smithies, S. (1999). Freehand formula entry system. Master's thesis, University of Otago, Dunedin, New Zealand.
- Smithies, S., Novins, K., and Arvo, J. (1999). A handwriting-based equation editor. In *Proceedings Graphics Interface*, pages 84–91, Kingston, Ontario, Canada.
- Starner, T., Makhoul, J., Schwartz, R., and Chou, G. (1994). On-line cursive handwriting recognition using speech recognition techniques. In *ICASSP-94. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 125–128.
- Sutherland, I. E. (1964). Sketchpad a man-machine graphical communication system. In *DAC '64: Proceedings of the SHARE design automation workshop*, pages 507–524, New York, NY, USA. ACM Press.
- Tappert, C. (1984). Adaptive on-line handwriting recognition. In *Proceedings 7th International Conference on Pattern Recognition*, pages 1004–1007.
- Teitelbaum, T. and Reps, T. (1981). The Cornell program synthesizer: A syntax-directed programming environment. *Communications of the ACM*, 24(9):563–573.
- Thimbleby, H. (1986). The design of two innovative user interfaces. In *Proceedings of the Second BCS*, pages 336–351, New York, NY, USA. Cambridge University Press.

- Thimbleby, H. (1990). *User Interface Design*. ACM Press, New York, NY, USA.
- Thimbleby, H. (1996). A new calculator and why it is necessary. *Computer Journal*, 38(6):417–433.
- Thimbleby, H. (2000). Calculators are needlessly bad. *International Journal of Human-Computer Studies*, 52(6):1031–1069.
- Thimbleby, H. and Thimbleby, W. (2007). Mathematical mathematical user interfaces. In *Engineering Interactive Systems: EIS 2007 Joint Working Conferences, EHCI 2007, DSV-IS 2007, HCSE 2007, Salamanca, Spain, March 22-24, 2007. Selected Papers*, pages 520–536, Berlin, Heidelberg. Springer-Verlag.
- Thimbleby, W. (2004). A novel pen-based calculator and its evaluation. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 445–448, New York, NY, USA. ACM Press.
- Thimbleby, W. and Thimbleby, H. (2005). A novel gesture-based calculator and its design principles. In L. MacKinnon, O. Bertelsen, N. B.-K., (Ed.), *Proceedings 19th. BCS HCI Conference*, volume 2, pages 27–32.
- Thomas, F. and Johnston, O. (1981). *Disney Animation: The Illusion of Life*. Abbeville Press, New York.
- Tognazzini, B. (1991). *Tog on Interface*. Addison Wesley.
- Triesman, M. (1977). Motion sickness: an evolutionary hypothesis. *Science*, 197:493–495.
- Twaakyondo, H. and Okamoto, M. (1995). Structure analysis and recognition of mathematical expressions. In *ICDAR '95: Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1, pages 430–437, Washington, DC, USA. IEEE Computer Society.
- Vronay, D. and Wang, S. (2004). Designing a compelling user interface for morphing. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 143–149, New York, NY, USA. ACM.
- W3C (2003). Scalable vector graphics (svg) 1.1 specification.
- Wharton, C., Bradford, J., Jeffries, J., and Franzke, M. (1992). Applying cognitive walkthroughs to more complex user interfaces: Experiences. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 381–388, New York, NY, USA. ACM.
- Wolfram, S. (1991). *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley.
- Young, D. A. and Wang, P. S. (1987). Gi/s: A graphical user interface for

- symbolic computation systems. *Journal of Symbolic Computation*, 4:365–380.
- Zanibbi, R., Blostein, D., and Cordy, J. R. (2002). Recognizing mathematical expressions using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1455–1467.
- Zanibbi, R., Novins, K., Arvo, J., and Zanibbi, K. (2001). Aiding manipulation of handwritten mathematical expressions through style-preserving morphs. In *GRIN'01: No description on Graphics interface 2001*, pages 127–134, Toronto, Ont., Canada, Canada. Canadian Information Processing Society.
- Zelevnik, R., Miller, T., and Li, C. (2007a). Designing ui techniques for handwritten mathematics. In *SBIM '07: Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, pages 91–98, New York, NY, USA. ACM.
- Zelevnik, R. C., Herndon, K. P., and Hughes, J. F. (2007b). Sketch: an interface for sketching 3d scenes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 19, New York, NY, USA. ACM.
- Zhao, Y., Sakurai, T., Sugiura, H., and Torii, T. (1996). A methodology of parsing mathematical notation for mathematical computation. In *ISSAC '96: Proceedings of the 1996 international symposium on Symbolic and algebraic computation*, pages 292–300, New York, NY, USA. ACM Press.

## Appendix A

# Anonymous questionnaire

The following pages show the anonymous questionnaire used for usability testing. The results from these questionnaires are shown in Appendix B

Thank you for taking part in my usability testing. During or after your use of the system please answer the following questions by either circling the appropriate answer or writing in the space provided.

Answers you give here are completely confidential, and will not be looked at till all the user-testing is finished and all questionnaires have been grouped together.

## Questions

Have you seen the system before?

How much do you normally use computers?

What is your occupation?

What is your overall impression of the system?

What did you feel to be the **best** parts of the system?

What did you feel to be the **worst** parts of the system?

How accurate was the system at recognising your handwriting and mathematics?

What do you use to normally calculate mathematics?

Which aspects make it **better** than how you normally calculate mathematics?

Which aspects make it **worse** than how you normally calculate mathematics?

What did you think about the user interface? Feedback and editing?

Did you enjoy using the system and what could be improved?

## Comments

Are there any other comments you would like to make?



# Appendix B

## Initial results

How much do you normally use computers?

- Not very often - for essays mainly
- lots
- Everyday for about 7 hours
- Lots
- Lots and Lots
- Only when I need to
- 1-4 hours a day
- 2 or 3 times a week
- Not much

What is your occupation?

- Student
- Placement officer
- Computer science student
- Student
- Doorsafe manager
- International student
- Handyman
- Mathematics student

What is your overall impression of the system?

- Its really good when you get the hang of it



- sleek styly + I want to use it
- very good
- very cool user friendly concept — got a few niggles that need ironing out
- new — interesting
- a lot of potential — needs to recognise my 5's
- very good and highly intuitive
- It works very well — sometimes the recogniser seems to be a bit sluggish. It's easy to be confident of what I'm calculating because it displays it on screen.

What did you feel to be the **best** parts of the system?

- Is fairly easy to use when you get the hang of it — you can write what you know of the sum and it works the rest out.
- sound effects — clean look — idea!
- much more natural than having to type equations in
- when the computer would fill in the blanks even if you hadn't finished the sum
- add and change elements in the calculation
- it's simplicity, use of "question mark" — also the ability of people with poor handwriting to produce clear equations
- the ease with which sums could be written in to the system having been copied direct from 'printed' notation
- Easy to edit equations — adding and removing parts — it works out the answer for me — understood nearly all of my handwriting

What did you feel to be the **worst** parts of the system?

- It didn't always recognise my numbers/signs
- got confused when it didn't understand my writing — actually I have an idea, decrease the size of the numbers when its a
- simple equation so you can sneak nos in
- Had to adjust how I wrote = to get it to work
- hand recognition (not working that well)
- The time used to clean the window
- the tablet takes a bit of getting used to and it didn't like my number 8s which I though were inoffensive

- Interpreted = as two fraction bars creating a big mess to clean up from a small error

How accurate was the system at recognising your handwriting and mathematics?

- Generally very good but didn't recognise some.
- good for all numbers — except  $4/5$  had to make it recognise
- good, had problems with = and 5's
- had trouble with some numbers + signs — but possibility of updating the system on my scribbling was good
- problem with getting it to delete properly and struggled to recognise my 5's
- almost flawless 4 is almost + but other wise no mistakes at all

What do you use to normally calculate mathematics?

- Calculator
- head – i only do simple stuff
- My head
- fingers, other people
- my head — paper or pencil — not complicated stuff
- a graphical calculator and computer programs
- an abacus (I don't 'do' maths)
- pencil and paper — a calculator if I really need one — often computers

Which aspects make it **better** than how you normally calculate mathematics?

- You don't have to find/understand all the buttons as you do on a scientific calculator
- I am more comfortable using comps these days so I'm not scared of using it. Also, it's a dream come true, someone giving me the answer just by writing the equation
- 2 to power of what = 28 are made much easier as you don't have to rearrange anything to do the calculation
- I can see how the sum is working and edit it at will. I can draw how I see the sum in my head
- the possibility to change the calculation without starting all over again
- the way you use it is much more intuitive and saves you time in terms of writing the answer down

- the problem does not have to be converted by me into a format comprehensible by a calculator and can do stuff I couldn't do on a calculator
- it requires an input method I am not familiar with
- no thinking about brackets or trying to find numerical keys on a keyboard — can see the computation as it's done. — can edit it, or add more steps to the computation, (in the middle of the expression!)

Which aspects make it **worse** than how you normally calculate mathematics?

- You have to cross out things when you've finished with it rather than just simply pressing cancel, you have to be careful about where you write numbers and signs, can be a bit confusing
- not as quick for simple calculations
- takes the fun out of using your head
- you can spend more time writing perfectly than doing the sum
- it didn't recognise my numbers frequently
- learn to recognise 8's and = signs. Maybe it easier to delete stuff - maybe somewhere on screen to press to clear the screen
- slow to recognise after input — can end up making the same error several times in a row as I try to enter something and it gets it wrong

What did you think about the user interface? Feedback and editing?

- excellent — didn't notice it that much
- v. good I found it hard to use the pen
- easy to use, edit. impressed
- I've always liked pens better than buttons
- very simple, really intuitive (esp. the delete gesture) nice how it adds in and calculates placeholders

Did you enjoy using the system and what could be improved?

- Yes but I got a bit confused at first.
- Include letters rather than ? ie.  $2^x$  rather than  $2^?$
- Yes — the handwriting recognition could be better — if it were able to distinguish similar symbols and throw up a warning — the user could re-input the symbols
- Yes, and clean the window — a possibility for deleting everything on the screen
- the method of deleting

- yes — a lot — could do with delete all — clear page — or equivalent — extend it to cope with multiple expressions — want to see several results at once

Are there any other comments you would like to make?

- get a good degree dude!
- from a teachers point of view: would be great fun to try this out on pupils
- i like the explosions — Maths teachers would love it
- it works really well



## Appendix C

### Royal Society — Briefing notes

# Team briefing notes

## Resources you should read

<http://www.cs.swansea.ac.uk/calculators/>

<http://www.cs.swansea.ac.uk/calculators/timetable.html>

Meet at the Strand Palace Hotel on Sunday evening 3 July at 6pm for a briefing meeting and to have some fun — we'll meet in the bar! See

<http://www.strandpalacehotel.co.uk/>

For those of you travelling, we will pay your expenses; so keep hold of all those the receipts. The hotel is booked from Sunday to Friday for you (Harold Thimbleby, Will Thimbleby, Will Harwood, Andy Gimlett, Matt Jones, Adam Powell).

At the meeting we'll go over all details, and hopefully we'll have photographs of our mock up so we can see how everything fits together.

Our contact mobile phone numbers are

07747790414      Harold

07818038777      Will

## What is the Royal Society Exhibit?

The Royal Society Summer Science Exhibition is a four-day exhibit of the UK's top science and technology research. The researchers exhibit their research. There will be 24 different exhibits from a wide range of different sciences, all exhibiting at the RS in London.

## What does it involve?

Talking to the public, discussing science, talking about calculators, and demoing the interactive gesture calculator.

There will be about 4,000 people over the course of the week, these will range from 16+ school kids to pensioners, to engineers, scientists, politicians and teachers, thus covering a whole range of people, most of who will be motivated and interested in what we are doing.

It will be very busy!

## Key objectives

First, it is really important that you have fun and enjoy the event. It will be exciting and very busy, and your excitement and fun will be infectious. *We don't know all the answers, and listening to people will be part of the science — and this is a point worth emphasising.*

We would like you to bear in mind the following secondary objectives:

- Communicating the fun of computer science.
- Evaluation, both as a science project.
- Evaluation, as a 'public understanding' project.
  - There is an evaluation form (with prizes!) for evaluation.
- Possibilities of research or development funding.
- Corporate contacts (we have a patent).
- Educational contacts, either for funding or for teachers.
- People who want to do PhDs.
- Museums who want to work with us to 'ruggedise' the display.
- Opportunities for further talks or exhibitions.
- Opportunities for articles.
- Opportunities for press coverage.
- New ideas.
- We have a letter to be given to 'good' contacts we want to see again!

We will have one (or more) ideas books, so either you or visitors can write down new ideas about any aspect of the work. Note that the ideas book is different from the evaluation form.

We'll give you briefing papers etc on Sunday.

### **What will be demonstrated**

The main thing being demonstrated is the gesture-based calculator running on white boards (ours are SMARTboards). However the principles, the science and getting people to think, are important. Instead of getting people just to use the whiteboards try and get them to suggest ways that calculators could be improved or to point out problems they currently have. Then show them our solution.

We want to get across that we are doing real science; we don't know all the answers; the work is not finished. We want feedback from visitors. We have got an evaluation form and a competition for the best suggestion each day — with a prize of an iPod. We would really like to get lots of feedback.

Also, if you have a visitor who is 'prestigious' we would like to collect testimonials from them and/or get permission to get testimonials from them later.

The exhibition stand will have a suggestions box to return these forms.

### **Good examples to show**

#### **Mathematics**

Simple things like  $4 \times 3$  or  $6 + 12$  just to give the idea of handwritten mathematics

The ability to overwrite and correct a simple example eg.  $4 \times 3 \rightarrow 4 \times 32$



The two-dimensional nature of the mathematics eg.  $\frac{2}{3} 3^4$

The ability to drag bits of the equation around eg.  $\frac{12x^3}{4} \rightarrow \frac{12}{4x^3}$  (moving the  $x^3$ ) or  $2^3 \rightarrow 3^2$  in one movement.

The fact that it is declarative eg.  $4x3=20$  and how it is corrected.

The ability to solve for unknowns eg.  $4x=20$  even deeply inside equations.

You don't need  $=$ , or it can be on the left or right ( $=3+5$ , or  $3+5$ , or  $3+5=$ ) and even you can have several, as in  $2x=3x=4x=12$

### Complex Mathematics

Square roots, factorials and powers eg.  $\sqrt{12}$ ,  $4!$ , solving for unknowns  $\sqrt{\quad}=25$ ,  $!=124$ ,  $2^{\sqrt{\quad}}=64$

Complex numbers  $\sqrt{-4}$ ,  $e^{i\pi}$ ,  $2^{\sqrt{\quad}}=-64$

Multiple equals signs  $2^{\sqrt{\quad}}=2^{\sqrt{\quad}}=-64$  or  $3^{x^4}=\frac{1}{2}=$

It can handle factorials (eg.  $!=5040$ ) and continued fractions (eg.  $1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + 1}}}$ ) nicely.

### More Interaction

Using the clock to undo and revise what has already been done

Using the number toggle to hide and show what solutions are, eg for classrooms or teachers, can also get people to guess and enter a number, rewind for another go.

Using the dock to store parts of equations, and dragging in equations from the dock into equations you are editing.

### Tips (some obvious and others not so)

Two of the comments made frequently at past RS Exhibitions have been

*"Exhibitors should be more forthcoming"*

and

*"More enthusiasm from exhibitors."*

Please be enthusiastic and open; if you are tired take a break. Most people want to be talked to and won't start a conversation themselves.

Make sure you get to play with the exhibit yourself. Play with the mechanical calculators, and the other exhibit stuff. The gesture based calculator does have some quirks, so practice writing on it and learn how to reset it — touching the cloud on the bottom left.

Try to pull people in when talking about the exhibit. Start with a question like “What problems have you had when using calculators?” or “Who’s got a calculator on their mobile phone?” We will have a display with several problems for calculators, and a pile of calculators for folk to try.

Getting people to try to do sums on mobiles is a great way to show some of the problems with calculators. Even simple sums like  $4x-5$  tie people in knots, and it leads great into talking further about the problems or our solutions.

Do not eat, sit down or get tied up using the calculator yourself whilst you are on the stand.

Try to involve everyone and if someone is taking up a lot of time, try to get them to fill in a feedback form, leave contact details, come back later. You do not have to demonstrate everything to everybody. Have fun!

---

Any immediate queries — please email Will [will@thimbleby.net](mailto:will@thimbleby.net) before the exhibition!  
Or give us a ring on our mobiles!

See you Sunday!

## Appendix D

### Royal Society — Evaluation form

Win an iPod  
shuffle every  
day for the best  
suggestions

# Weapons of Maths Construction Feedback

Date

Age

☐ <16 ☐ 16-18 ☐ 18-25 ☐ 25-45 ☐ 45+

Sex

☐ M ☐ F

Highest mathematical qualification

Occupation

Contact details

(required for prize draw)

☐ Can we contact you?

All data will be anonymised and treated in confidence.

How do you do mathematics / sums?

☐ Calculators ☐ Spreadsheets ☐ In my head ☐ Paper

details....

Do you have problems with the current mathematical method you use?

☐ Y ☐ N

Would you have said 'yes' before visiting our exhibit?

☐ Y ☐ N

details....

Is it better or worse than you current method?

☐ Better ☐ Worse

details....

I enjoyed using it

(disliked it — loved it) ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

I thought it was helpful

(unhelpful — very helpful) ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

details....

How could it be improved? Any other suggestions?

details....

Would you like to write something supporting our work, which we could acknowledge and quote in e.g. articles or for research proposals?

out of space? you can use the back of this form...

—thankyou

Please fill in this form, and hand it back to one of the Weapons of Maths Construction exhibitors, drop it off in the suggestions box, or mail it to us at Will Thimbleby, Department of Computer Science, Swansea University, Swansea, SA2 8PP

## Appendix E

# Royal Society — Results

The following results table contains all the data from the Royal Society Feedback Form (Appendix D). In the results table on the following pages the keys in the table header have the meanings listed in the table below. Interesting feedback from comments or other data are highlighted in the results table in grey.

Key	Meaning
Age	Age (1-5) <16, 16-18, 18-25, 25-45, 45+
M/F	Gender
Qualification	Highest mathematical qualification
Occupation	Occupation
How	How do you do mathematics / sums? (1/0) Calculators, Spreadsheets, In my head, Paper
?	Do you have problems with the current mathematical method you use. (y/n)
<	Would you have said yes before visiting our exhibit. (y/n)
B	Is it better than you current method. (y/n)
E	Enjoyment (1-5)
H	Helpfulness (1-5)
Comments	Comments from any part of the feedback form

Age	M/F	Qualification	Occupation	How	?	<	B	E	H	Comments
4	m		Project Manager	1111	n	n		4	4	teaching mechanism should be improved to show working/construction as teacher would on bb
5	m	O	Retired Nurse	1011	y	n	y	3	4	variables (cp coefficients & remainder theorem) and graphs
2	f	AS	Student	1011	n	n	y	4	4	
2	f	AS	Student	0001	n	n	y	4	4	
1	m	KS3	Student	0011	n	n	y	5	5	
3	m	Uni	Student	1111	n	n	y	5	5	more functions & spreadsheet function - like new interface
3	m	Masters	IT	1111	n	n	n	5	5	calculators faster good for teaching not on a daily basis
5	f	GCSE	Retired	0011	n	n	y	5	5	"use what if examples, incredible!"
2	f	GCSE	Student	1011	n	n	y	5	4	"more enjoyable, it was fun to play and see it come on board!"
2	f	GCSE	Student	0011	y	n	y	5	3	"don't like calculators, put in sin cos tan"
2	f	GCSE	Student	1001	n	n	y	4	3	formulas letters and symbols
2	f	AS	Student	1001	n	n	y	4	3	radians and graphs
1	m	KS3	Student	0011	n	y	y	5	5	"this is very great exhibition, 1000 times better,"
2	m	GCSE	Student	1011	y	n	y	4	4	smaller scale
2	m	GCSE	Chief	0010	n	n	y	5	5	"it's like a toy, more colours"
2	m	GCSE	Student	1011	y	n	y	4	3	"really fun to use, makes it unnecessary to do lots of repeat calculations, cube roots and words"
2	m	GCSE	Student	0010	n	n	y	5	3	"quite fun, allows you to follow through a progression of sums, cool sound effects, h/w"
1	m	Advanced Higher	Student	1001	y	y	y	3	3	"less boring-j, more active, would get better if I was familiar"
4	m	A	Teacher	0010	y	y	y	5	5	"mostly in my head, sometimes need to ask students to help, head is intimidating for some students who can't keep up, I can see myself using this as an alternative to students calculators"
1	m	7	Student	0001	n	y	y	5	5	I loved it.
1	m	6	Student	0010	n	n	y	5	5	It's already good.
1	m	SATS	Student	1111	n	n	y	4	2	Cool
1	f	8	Student	1001	n	y	y	5	5	"a lot easier, fun"
1	f	SATS	Student	0010	n	n	y	4	3	can be confusing
2	m	Further Maths A	Student	1001	n	n	y	5	5	can't be improved
3	m	Further Maths A	Student	0010	n	n	y	4	1	"it's great! it's brilliant, better than pen and paper"
5	m	BA	Retired	1111	n	n	y	4	4	letters
1	m	8	Student	0001	n	n	y	4	5	"xl can be audited, new is easier to use"
2	m	GCSE	Student	1011	n	n	y	4	4	"great alternative to normal calculators, popup suggestions, show working, different colours as tools of memory"
1	m	GCSE	Student	0011	n	n	y	5	4	No its wicked.
1	f	GCSE	Student	0011	n	n	y	5	4	"confusing at first, mathematical structures and working, on the whole, utter brilliance"
3	f	GCSE	Student	0001	n	n	y	5	5	"it is better because you can work out the equation very quickly, it takes less time to get the answer, provide it to all students at colleges and university"
2	m	AS	Student	1000	n	n	y	5	5	you don't need brackets
2	m	GCSE	Student	1010	n	n	y	4	4	"friendly staff, logs, trig, background grid lines for guidance"
4	f	GCSE	Student	0010	n	n	y	3	3	"it makes maths more fun, convenient format"
2	m	GCSE	Student	1011	n	n	y	4	4	"it visualises the internal workings of abstract calculations, fun, as it is won-
2	f	AS	Student	0010	n	n	y	5	4	derfull Fun! Engaging and importantly visible!"
2	m	AS	Student	0010	n	n	y	5	5	
4	m	Uni	Professor	1111	y	y	y	5	5	





2	f	A	Student	1001	n	n	y	5	4	less complicated than pressing buttons on a calculator and easier than using paper - how about a kb for use without a touch screen
4	m	A	Research Analyst	1111	n	n	y	5	5	"much more efficient - formulas, simultaneous eqs.it is brilliant, shows how simple it can be to use a powerful application."
1	f	KS3 SATS	Student	1010	y	y	y	4	4	"its quicker, corrects mistakes and simple to use"
1	f		Student	1011	n	n	y	5	5	"portable, easier and keeps track of previous calculations"
1	m		Student	1010	y	n	y	5	5	more colourful
1	m		Student	1010	n	n	y	5	4	
1	m	SATS	Student	0010	y	n	y	5	5	mobiles put it in with little pens or make calculators with little pens
1	m	PHD	Director	1011	n	n	y	5	5	very good for education
4	m	Degree	Teacher	1111	n	n	y	5	5	"good for demonstrating, classwork examples"
4	m	GCSE	Science Teacher	1000	n	n	y	5	5	fantastic for those that learn by moving touching and doing. Kids will love it and find it more enjoyable than traditional methods
4	m	GCSE	Marketing Exec	1111	n	n	y	5	5	easier to use faster than my brain
3	m	GCSE	Student	0010	y	n	y	4	4	calculus
3	f	BSc	Student	1000	n	n	y	4	4	
4	f		Research Scientist	1100	y	y	y	5	5	
1	f		Student	1111	y	y	y	4	3	confusing
1	f	O	Teacher	1011	y	y	y	5	5	"number pyramids, pairs added are higher up till the answer at the top"
5	f	GCSE	Student	1011	n	n	y	5	4	people may rely on it too much
2	f	GCSE	Student	1010	n	n	y	5	4	
2	f	AS	Student	1010	y	y	y	3	4	"far better easier and more reliable, add binary"
1	f	SATS	Student	1111	y	y	y	5	5	brilliant for children
4	f	O	Director	0110	y	y	y	4	4	much more efficient
2	f	GCSE	Student	1011	y	n	y	5	5	"really good and fun, after this I do believe I had a problem"
1	m	GCSE	Student	1010	n	n	y	5	5	I thought it was great and should be used in classrooms
4	m	BSc	Student	1011	y	y	y	5	4	"rational display, geometry"
2	f	GCSE	Student	1001	y	y	y	4	4	easier to use!!!
2	f	GCSE	Student	1111	y	y	y	5	5	"it makes the process fun with the images and sounds, voice recog, more colourful, cartoon characters"
4	f	O	Consultant	0001	n	n	y	5	4	
5	f	IB	Consultant	1011	y	y	y	5	5	
1	f	SATS	Student	1011	n	n	y	5	5	easy to use and fun to
1	f	SATS	Student	1011	n	n	y	5	5	"very fun to use and helps work things out, much easier and very entertaining"
5	m	Uni	Mgt Consultant	1011	y	y	y	4	4	"voice, pictures eg cows/apples"
1	m		Student	1110	y	y	y	5	5	"super love the design and the clock, rather than undo function, more colourful, easy to use calculates quickly does nearly everything you need"
4	f	Uni	Mgt Consultant	1111	y	y	y	5	5	a way of moving concepts and arguments around
4	f	Masters	Student	1011	n	n	y	5	5	allow problem solving
4	f	PhD	Research Associate	1011	n	n	y	4	4	it was fun aswell as useful
4	f	GCSE	Student	0010	n	n	y	5	5	algebra
2	f	GCSE	Student	0010	n	n	y	5	5	It was fun to use and very different
1	f	GCSE	Student	0001	n	n	y	5	4	
2	m	GCSE	Student	1000	n	n	y	5	5	algebra
2	m	GCSE	Student	1011	n	n	y	5	5	

2	f	GCSE	Student	1011	n	y	5	5	you can visualise the calculation algebra
2	m	GCSE	Student	1011	n	y	5	5	it is more accurate
2	f	AS	Student	0001	y	y	4	3	may help students to concentrate in class
1	f		Student	1011	y	y	5	4	
4		GCSE	Writer	1000	y	y	4	4	I was very impressed
5		GCE	Investor	1000	n	y	5	5	recognition
5	f	BSc	Retired Teacher	1011	n	n	4	2	different
1			Student	1010	n	n	4	4	
4		O	Journalist	1010	n	y	5	5	
5	m	A	Academic	1010	y	y	5	5	
4	m	A	Biologists	1110	y	y	5	5	symbols
3	m	A	Grant Officer	1110	y	y	5	4	"I didn't think there was an easier way till now, the possibilities are endless."
5	f	GCSE	MD	0100	y	y	5	5	
5	f	CSYS	Project manager	1111	n	y	5	5	simple mode with no powers etc.
4	m		Student	0001	y	y	5	4	"it was fun, better accuracy"
1	f		Student	0010	n	y	5	5	approx = and exact answers
5	f	BSc	Teacher	1111	n	n	3	3	
3	f	Higher AS	Research Officer	1111	n	y	3	3	
2	m	A	Student	1011	n	n	3	2	
2	m	BSc	Building Surveyor	1000	n	y	5	5	
5	m	AS	Student	1110	n	y	4	4	
2	m	A	Computer Program-	1011	y	y	4	4	like seeing full calculation
4	f	A	mer						
4	f	A	Web Designer	0100	n	y	5	5	algebra
3	f	A	Student	1001	n	y	4	5	smarter
1	m		Student	0001	n	y	4	5	smaller
4			Head of Interactive Media	0010	y	y	5	5	
5		BSc	Maths Teacher	1111	n	y	4	4	"good because it gives an answer whatever and these can be discussed, box where you want a number, recurring decimals, algebra, rationals"
4		GCSE	Teacher	1000	y	y	4	4	
4	f	O	Writer	1011	n	y	4	4	
4		A	Librarian	1111	n	y	5	5	sounds
4	m	A	Doctor	1000	n	y	4	4	"this is a huge improvement, very interactive"
5	m	Uni	Retired Chemist	1010	y	y	5	5	"clunky but potentially brilliant, it leads to more questions which is great"
4	m	A	Research Manager	1110	n	n	5	4	
5	m		Legal Advisor	1100	n	y	5	5	It is very instinctive and fast. It's great I feel like Tom Cruise in Minority Report - Bravo!
4	m	Post Grad	Student	1111	n	y	5	5	
1			Student	0010	y	y	5	4	clock is cool
5	m	MA	Teacher	1011	n	y	5	5	
2	f	GCSE	Student	1011	n	y	4	4	percentages and do words
5		O	Accountant	1111	y	n	5	3	geometry
4		Uni	Insurance	1101	n	n	2	3	"h/w improvement, copy + paste"
4		Higher	Research	1111	n	n	5	4	
3			physicist						frustrating
4	f	PhD	Astronomer	1011	n	n	2	2	

2	m	GCSE Stats	Student	0001	n	n	y	4	4	% and words certainly more fun
1	f	A	Student	1011	n	n	y	5	4	
5	f	NVQ	University Researcher	1000	n	n	y	2	4	
4	m	GCSE Dip	Community Worker	1100	y	y	y	3	4	
5	m		Civil servant	1101	y	y	y	4	3	long hand explanation of sum
1	m		Student	1111	n	n	y	4	3	precision and smaller algebra and h/w
1	m		Student	0011	n	n	y	5	3	confusing
4	m		Software Eng	0011	n	n	y	4	2	
5	f		Retired	1011	n	n	y	5	5	
5			Retired	1111	y	y	y	4	3	
5			Architectural	1000	n	n	y	5	4	
4		A	Professor	0011	n	n	y	4	3	you can see what is happening
3		GCSE	Student	0100	n	n	y	4	3	this is fun to use and could be very helpful in encouragin learning / fun with numbers
5			Civil servant	1010	n	n	y	5	5	great project I think it will help kids take more interest in maths lessons
3	m	BSc	Student	0001	y	y	y	5	4	
5	n	O	Science Teacher	1011	y	y	y	5	5	
4	m	GCSE	Doctor	1000	y	y	y	3	4	great fun
5	m	A	Research Assistant	1111	y	y	y	4	4	unit conversions
2	f	GCSE		1111	y	y	y	3	3	
1	f	KS4	Student	0010	n	n	y	5	5	
1	f	GCSE	Student	1011	n	n	y	5	5	it's better: you can touch it. give more explanations
4	f	BSc	Surgeon	1111	y	y	y	4	3	stats version
5		PhD	Psychotherapist	1000	n	n	y	3	2	
5		Uni	Physics	0011	n	n	y	4	5	instructions
4	m	A	Medical	1111	n	n	y	4	4	"more visualisations, piecharts graphs, loci"
4		GCSE	Project Exec	0101	n	n	y	4	4	looks wicked!
3	m	GCSE	Graphic Design	1100	n	n	y	5	5	
5	m	Degree	Actuary	1011	n	n	y	4	4	showed me some limitations of calculators
5	m	A	Retired	1111	y	y	y	4	4	
2	m	AS	Student	1001	n	n	y	4	4	
3	m	Masters	Consultant	1011	n	n	y	3	3	
4		BSc	Civil servant	0111	n	n	y	4	4	better for visualisation
2		GCSE	Student	1111	n	n	y	5	4	thought and voice based
4	m	A	Engineer	1001	n	n	y	4	4	faster and portable
2	f	GCSE	Student	1111	n	n	y	3	3	
1	f	GCSE	Student	1011	n	n	y	4	4	
2	m	AS	Student	1001	n	n	y	5	4	
2	m	AS	Student	1011	y	y	y	4	4	
2	m	GCSE	Student	1000	n	n	y	5	4	great for young people needs more for a levels
1	m	Level 7 Sats	Student	1010	n	n	y	5	4	
2	m	AS	Student	0010	y	y	y	5	5	
1	m	AS	Student	1011	n	n	y	4	3	"it was fun to use and easier. I enjoyed using it. Make a smaller one."
1	m	AS	Student	1011	n	n	y	5	5	"novel and useful, more functions"
2	f	GCSE	Student	1011	y	y	y	5	4	I don't know how my calculator will work so I tend to do all sums in small stages and write answers down at each stage - this takes ages. I like the size and clarity of it and how easy it is to use (as well as removing the problem above)

2	2	m	A2	Student	1011	n	n	n	4	I expect to see these in future
2	4	f	A	Student	1011	n	n	y	3	turing back time a brilliant idea
3	3	f	AS	Doctor	1111	y	y	y	4	"a fantastic piece of software throw out your casio" - could do with graphing"
2	2	m	AS	Student	1111	y	n	y	5	toolbar for symbols
2	2	f	GCSE	Student	0011	n	n	n	1	"a nice "teacher" he was very entertaining and whatever he was demonstrating was easy to enjoy"
2	2	m	GCSE	Student	1000	n	n	y	4	I think it is great
2	2	f	AS	Student	1111	n	n	y	3	very impressed
2	2	f	GCSE	Student	1000	n	n	y	5	on the market!
2	2	m	AS	Student	1011	n	n	y	5	way way better
2	2	m	AS	Student	1011	n	n	y	5	much more accessible and powerful than any other calculator I have seen
2	2	m	AS	Student	1011	n	n	y	4	it shows you where you've gone wrong
2	2	f	GCSE	Student	1000	n	n	y	5	
2	2	f	A	GP	1011	n	n	y	5	
2	2	f	GCSE	Student	1000	n	n	y	4	
2	2	f	PhD	Director	1011	n	n	y	3	
2	2	m	GCSE	Psychologist	1011	n	n	y	5	
2	2	m	GCSE	Student	1000	n	n	y	5	
2	2	m	Further Maths A	Maths Student	1011	n	n	y	5	
2	2	f	GCSE	Student	1011	n	n	y	4	
2	2	f	GCSE	Student	1011	n	n	y	5	
2	2	f	GCSE	Student	1011	n	n	y	4	
2	2	f	SATS	Student	1011	n	n	y	4	
2	2	m	AS	Student	0010	y	y	y	3	
2	2	m	AS	Student	1011	n	n	y	4	
2	2	m	AS	Science Writer	0011	n	n	y	4	
2	2	f	AS	Student	1010	n	n	y	5	
2	2	f	GCSE	Student	1110	n	n	y	4	
2	2	f	AS	Student	0010	y	y	y	5	calculators seem clumsy and hard to use - the new method is genius - when can I buy one in the shops (I would have done A level maths)
2	2	f	AS	Student	1000	n	n	y	5	"show the process, bases other than 10"
2	2	f	HNC	Retired Engineer	1110	n	n	y	5	powerful tool for encouraging full mathematic competence in primary and middle school students. At my current level I would have little use for the communal calculator as it stands. However I can see enormous potential. Very useful for thinking aloud. - could be important. If several WMCs were networked locally or nationally or even internationally scientists and mathematicians would better be able to communicate their ideas. surely individual prose recognition would not go amiss.
2	2	f	GCSE	Student	1011	n	n	y	4	can't be broken by ignorant people
2	2	f	GCSE	Student	1010	y	y	y	5	smaller
2	2	m	Further Maths A	Marketing	1000	n	n	y	3	the order in which you input commands is calculator specific.
2	2	m	GCSE	Student	1111	n	n	y	4	keyboard for writing
2	2	f	AS	Student	1111	n	n	y	4	"it allows the user to do their sums more easily and is very user friendly. I liked the touchscreen & graphics, it makes maths & numbers more interesting"
2	2	f	AS	Student	1001	n	n	y	4	speed adjust
2	2	m	A	Financial contractor	1100	n	n	y	2	
2	2	m	A	Student	1011	y	y	y	5	
2	2	m	A	Research Scientist	1011	n	n	y	4	

5	m	BSc	Retired Engineer	1011	n	n	y	4	4	great as a learning tool
2	m	OU	Author	1111	n	n	y	5	5	
4	m		IT Manager	1111	n	n	y	5	4	spring mass visualisation
5	m	scholarship	Retired lecturer	1000	n	n	n	4	3	brilliant
2	f	GCSE	Student	1010	n	n	y	5	5	"red for computer not very suggestive (try light grey), cross out erase, sin cos
4	f	PhD	Research	1000	n	n	y	5	5	by dragging from toolbar"
5	m	BSc	Engineer	1010	y	y	y	4	4	
3	f	GCSE	Student	1010	n	n	y	5	5	amazing I have no idea how you came up with it! Smaller and ink-editing idea
4	f	A	Science Communicator	1011	n	n	y	4	4	will kids think even less??
5	m	BSc	Teacher	1111	y	y	y	4	5	quicker and more lucid and responsive
2	m	GCSE	Student	1000	y	y	y	4	5	"great because it stores loads of back-workings, very helpful and easy to master.
5	m	O	Accountant	1111	n	n	y	5	5	3-8 ink editing"
5	f		Retired Teacher		n	n	y	5	5	"brilliant simple in concept, was so encouraging to meet such a charming, elo-
2			Student	1001	n	y	y	5	5	quent young man, so enthusiastic and good at simple explanation"
5		O	Chartered Surveyor	0011	n	n	y	4	3	ditch ocr and go for pallet - quicker and more reliable
5	f	Higher	Civil Servant	1011	n	n	y	5	5	
5		A	Administrator	1000	n	n				I'm interested in encouraging your people to study maths (+sci) at uni. If
4		A	Teacher	1000	n	n				you would like to contact us to discuss our stem e-mentoring project (vivi-
4	m	GCSE	Education Policy	0111	n	n	y	5	5	enne.themakeeping@thebrightsidetrust.org)
5	m	BSc	Teacher	1111	n	n	y	5	5	is there anything more simple!?"
4	m	MSc	Science Teacher	1001	n	n	y	5	5	"very clever, it's inclusive, very visual"
5	m	A	Teacher	1111	n	n	y	5	5	"coloured backgrounds, great for schools, real wow factor"
5	m	O	Teacher	1111	n	n	y	5	5	more animation
4		A	Education Manager	1111	y	y		4	4	physical engagement inc. movement offer engages learners more esp. in subjects
4		BSc	Science Communicator	1110	y	n	y	5	5	like maths and sci. stat analysis
5		A	Physics Teacher	1111	n	n	y	3	3	"fun to play with, some sort of keep red action"
5		BSc	Teacher	1111	n	n	y	3	3	larger
4		PhD	Hydrogeologist	0111	y	n	y	5	5	get it in schools
4	f	GCSE	Teacher	1011	n	n	y	5	5	arbitrary precision
4		CSYS	Teacher	0100	n	n	y	5	5	
4		A	Headteacher	1111	n	n	y	4	4	explanation of sums
1		GCSE	Project Manager	1110	y	y	y	3	4	excellent for the advertising of math
4		GCSE	Student	1011	n	n	y	5	5	
5	f	CSE	Builder	1010	y	y	y	4	3	
5	f	A	Teacher	1011	n	n	y	4	4	
5	f	BSc	Teacher	1111	y	n	y	4	4	does it help understanding
4	m	A	Project Manager	1110	n	n	y	4	4	
5	f		Teacher	1010	n	n	y	4	4	
5	f		Nursery Supervisor	0011	n	n	y	5	5	chn are inspired by such interactivity
4		O	Teacher	1001	n	n	y	5	5	

	O	Headteacher	1011	y	y	5	5
5							
5	m	Teacher	1011	n	2	3	
4		Accountant	1111	n	4	4	
5	m	Teacher	0100	n	n	4	3
5	f	Science Teacher	1011	n	y	4	4
4		Head of Science	1111	n	y	5	4
5	A	Teacher	1111	n	y	5	4
	MSc	Teacher	1010	y	n	5	5
4	BSc	Teacher	1111	y	y	5	5
4	MSc	PhD	1110	y	y	4	4
4	f	Science Teacher	1011	n	5	3	
4	CEE	Research Scientist	1111	n	y	5	5
4	GCSE	Headteacher	1111	n	y	5	4
5	m	Cert Ed	1011	y	y	4	5
3	m	Marine Biologist	1100	y	y	4	4
4	f	Maths Teacher	1111	n	3	3	
3	GCSE	Science-tech	1010	n	y	5	4
4	PhD	Teacher	1110	y	y	5	5
4	PhD	Research	0010	y	y	5	3
5	m	Senior Lecturer	0110	n	y	4	4
5	m	Retired Teacher	1110	y	n	5	5
5	f	Teacher	1011	n	y	5	5
2	m	Student	1011	y	y	4	4
2	m	Student	1011	y	y	4	4
2	m	AV Technician	1010	y	n	5	5
2	f	Student	1000	n	y	5	4
3	AS	Student	1010	n	y	5	5
3	A	Student	1011	y	y	5	5
2	f	Student	1010	n	n	5	4
2	f	Student	1001	y	n	4	4
2	f	Student	1001	n	y	5	5
5	O	Director	1010	n	y	5	5
5	A	Game Designer	0011	n	n	4	4
2	AS	Student	0001	y	y	5	5
2	m	Student	0001	y	n	5	5
2	m	Student	1001	y	n	5	4
5	O	Teacher	1011	y	y	5	5

5	BA	Artist	1011	n	n	y	5	5	"better by far. It's difficult to describe why, but it seems to fit my brain's way of working better. Might work well for children who struggle with current teaching methods. I've never seen anything that's brought a smile to my face while doing addition, but this has. For that reason alone, I want one! You could change a generation's view of maths with this - it really must go to schools. The younger the better."
4	Uni	Research Student	1101	n	n	y	5	5	it's fun
2	A	Student	1111	y	y	y	5	4	trig
2	AS	Student	1000	y	y	y	5	5	great
2	AS	Student	1000	n	y	y	5	5	needs calculus
2	A	Student	1000	n	y	y	5	5	couldn't be better
2	A	Student	1000	y	n	y	5	5	Fascinating piece of software - I enjoyed trying to cheat it!
2	A	Student	1001	n	n	y	5	5	I love it because it calculates as I would on paper only much much faster! (& more correct)
2	AS	Student	1001	y	y	y	5	5	It corrects mistakes and is faster.
2	A	Student	1011	y	y	y	5	5	Solving is much simpler. Allows you to work without losing your train of thought having to use a calculator. Also easier to check answers. Made maths even more fun. Would be good to solve logs and graphs would be a good way of visualising and understanding things.
4	BSc	Molecular Biology	1100	n	n	n	3	1	"use a two tap delete, and make it backward compatible so users familiar with calculators can use it immediately."
1	GCSE	Student	1000	n	y	y	5	5	make it quicker
2	A	Student	0010	n	y	y	4	5	GUI is fantastically intuitive. Well done! I think a combination of this with a keypad for everyday entry would make it fantastic.
3	Uni	Engineer	1011	n	n	n	5	4	
2	GCSE	Student	1001	n	n	y	5	5	excellent no need for improvement
1	SATS	Students	1000	n	n	y	5	5	Negative numbers are annoying in calculators. I like the interactivity and that you can see what you are doing.
2	AS	Student	1011	y	n	y	4	4	I'd like to use it.
2	A	Student	1011	n	y	y	5	5	
5	HNC	Chartered Accountant	1111	n	n	y	4	4	"excellent, no improvements"
1	SATS	Student	1000	n	n	y	5	5	"the exhibition is amazing, I love it gets me excited. Better than anything I've seen before"
2	A	Student	0010	y	n	y	5	5	exhibition helped me realise how cumbersome (mentally) calculators/spread-sheets are
5	PhD	Nuclear Engineer	0110	y	n	y	4	5	write on it as well
1	year 4 primary	Pupil	0001	y	y	y	5	5	a simple handbook would be good for massmarket
5	BSc	Science broadcaster	1011	y	y	y	4	4	easier to use and see when you've gone wrong
2	GCSE	Student	1011	y	y	y	4	4	
5	A	Professor	1000	n	n	y	5	5	It's fun and so causes learning to stick.
2	AS	Student	0010	n	n	n	5	5	great fun and informative! I want one!
1	SATS	Pupil	0010	y	n	y	5	5	it was fun but sometimes didn't understand my writing. Make it more colourful and have something like office assistant.
1	SATS	Student	1111	n	n	y	5	5	sometimes I'm not clear enough with my working and this can lead to the wrong answers (with original calcs) you can rearrange formulas and this can make questions and equations easier as you can see how things are linked. Simple and easy to follow it encourages learning for children and is great for class demos. add in novelty characters and more colours and fonts
1	KS3	Student	1011	y	y	y	4	5	
1	KS3	Student	1011	y	y	y	4	5	

1	KS3	Student	1011	y	y	y	4	4	it was simple and therefore helpful and fun.
4	A	Science Teacher	1011	y	y	y	5	5	Very intuitive. An excellent teaching (fun resource)
2	GCSE	Student	1011	y	y	y	3	3	
5	A	Doctor	1101	y	y	y	3	3	
2	AS	Student	1010	y	y	y	5	5	needs improved h/w recognition
4	Masters	Scientific Attach	0011	n	n	n	4	4	it was interesting to use although it takes some getting used to
2	GCSE	Student	1011	n	n	n	5	5	add in symbols like sum and log and graphs
2	AS	Student	1111	n	n	n	4	4	looks exciting
2	GCSE	Student	1111	n	n	n	3	3	probably better for school kids
3	AS	Student	1001	y	y	y	5	5	nice and hands on no real need for improvement
2	AS	Student	1001	y	y	y	4	4	merges my two current methods together - very handy!
3	AS	Student	1010	n	n	n	5	5	
2	f	Student	1010	n	n	n	4	4	I thought it was really cool
2	AS	Student	1011	n	n	n	5	5	it's more efficient
1	A	Post-Grad	1000	n	n	n	5	5	"more intuitive, main problem is speed of recognition"
4	GCSE	Student	1011	n	n	n	4	4	
2	m	Post-Grad	1011	n	n	n	5	5	
5	m	Student	0111	n	n	n	4	4	it needs to know when it guessed incorrectly
4	BSc	Housewife	1111	n	n	n	4	4	
5	GCSE	Consultant	1011	n	n	n	5	5	
5	PhD	Church worker	1111	n	n	n	3	1	
3	A	Student	1011	n	n	n	4	3	"great fun to use as it is so different, it can't recognise a lot of mathematical symbols like dx/dy, how about circle and tap delete"
3	GCSE	Student	1011	n	n	n	5	4	be able to use it on normal computers
2	AS	Scholar	1010	y	y	y	4	4	before I hadn't realised how inaccurate and problematic calculators can be. I thought it was very liberating to use such a fun system.
2	GCSE	Student	1010	y	y	y	5	4	Better? Hell yeah!
5	BA	Retired Engineer	1111	y	y	y	4	4	I can't wait for casio to produce one.
1	HNC	Retired Engineer	0011	n	n	n	5	5	
5	NCEA	Unemployed	1000	n	n	n	5	5	
2	GCSE	Student	1011	n	n	n	4	4	
2	GCSE	Student	1010	y	y	y	4	4	it is so annoying when mistakes are made just because of misplaced brackets etc. Way better you can see what you are doing and avoid unnecessary mistakes. The clock could show you how long it would HAVE taken using a normal calculator! Best calculator I've seen and keeps you fit (need to stretch etc.) Could add an animated assistant like the paperclip. it allows you to see how it works things out. Exciting and different.
1	m	Student	0010	n	n	n	4	5	
1	SATS	Student	1001	n	n	n	4	4	"best of both worlds, power of the calculator and the ease of use of paper"
2	AS	Student	1001	n	n	n	5	4	set it up form a young age
2	AS	Student	1010	y	y	y	4	4	get it on the market! NOW
1	GCSE	Student	1010	n	n	n	5	4	
1	f	Student	1000	y	y	y	4	3	
2	m	Student	1001	y	y	y	4	5	
1	A	Student	1011	n	n	n	4	4	
1	GCSE	Student	1110	n	n	n	4	5	
1	m	Business Consultant	0100	y	n	y	4	3	it was great. Big screen for big equations. Liked sound effects and magic finger and clock
5	A	Student	0010	y	n	y	5	5	I like writing down as it is much faster.
5	BSc	Retired	1111	n	n	n	4	4	
3	A	Student	1010	n	n	y	5	5	
4		Doctor	0001	n	n	n	5	4	



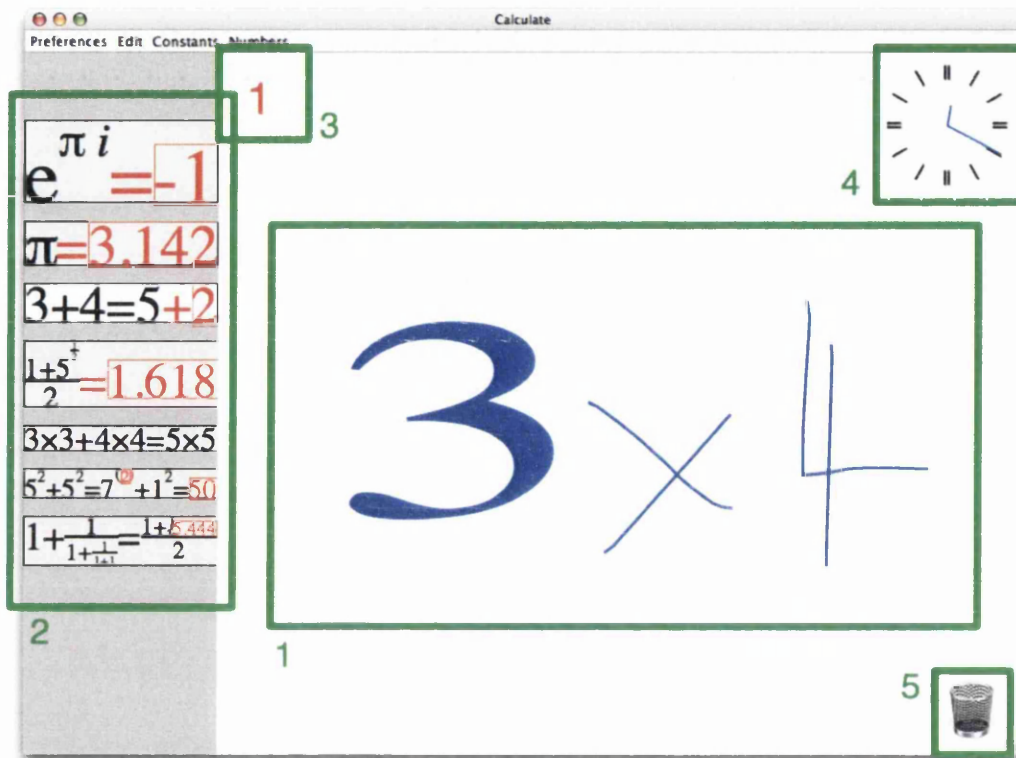
5	m	AS	Retired	1010	n	n	n	3	3	2	you can't control what no. will change - not efficient
2	m	AS	Student	1011	n	y	y	4	5	5	handwriting could be better
2		AS	Physics Teacher	1011	n	y	n	5	5	3	"limited functions available, - leave a line by line history up the board for derivations"
4		A	Science Museum	1111	y	y	y	5	5	5	"I want to do an exhibition at the Science Museum, more than one calculation at a time"
4		A	Student	1000	y	n	y	5	5	5	very funky
1	m	GCSE	Student	0010	n	n	y	5	4	3	
1	f	GCSE	Student	1000	n	n	y	5	5	4	
4	f	GCSE	Exhibition developer	1000	n	n	y	5	5	5	link to excel
2		AS	Student	1000	n	n	y	5	5	5	
5	m		Banker	1111	n	n	y	5	5	5	"As an ex Technical assistant, Royal Artillery, I am sure this has potential for the Army - working out ranges etc where laser technology may not be available."
2	m	A	Student	1010	n	n	y	5	5	5	Brill.
5	f	AS	Chief Scientist	1100	y	y	y	5	5	5	
2	m	AS	Student	1010	n	n	y	5	5	4	
2	f	AS	Student	1010	n	n	y	5	5	5	
4		HSC	HighSchool Teacher	1000	n	n	y	5	5	5	"fantastic teaching aid especially for higher level maths, I like the way it sets everything out clearly you can see how it got the answer"
2	m	AS	Student	1011	n	n	y	5	5	5	cube roots needed
5		BSc	Programmer	1111	n	n	y	5	5	3	the name realcalc has been used for a commercial product
2		Higher	GP	1001	n	n	y	5	5	4	too many in the queue
2		AS	Student	1011	n	n	y	2	4	4	
2	m	AS	Student	1000	n	n	y	4	4	4	
2	m	AS	Student	0001	n	n	y	5	5	5	
4	m	AS	Lecturer	1100	n	n	y	4	4	4	annotation to reflect what the program is doing
4	m	A	Student	1000	n	n	y	4	4	4	

## Appendix F

# Calculator manual

# TrueCalculator Manual

## The interface



### 1. The equation editor

Equations are written here, which the calculator solves. Figure 1 shows the calculator in the process of solving  $3 \times 4$ .

### 2. The equation dock.

Equations can be dragged here from the equation editor. These are recorded and saved.

### 3. Hide/show numbers.

This button toggles whether computed results are shown.

### 4. History.

The clock records the changes made in the equation editor. It can be used to review what has happened and to undo changes.

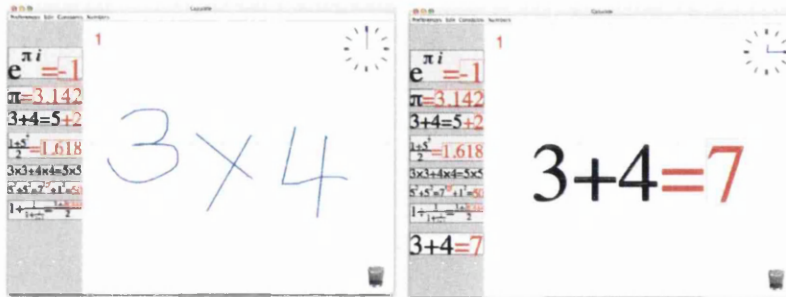
### 5. Trash.

By dragging to the trash saved equations in the dock and parts of the equation editing can be deleted.

## Solving an equation

To solve an equation write it in the equation editor, just as it is written on paper.

TrueCalculator uses handwriting and expression recognition to understand what you have written. It then morphs your input into typeset output with the answer inserted for you.



As a user enters an equation, typeset characters replace digits and symbols. This enables the user to recognise when a character has been misrecognised and stop to undo the error. Once the user has finished entering the equation or pauses the calculator morphs the annotated, typeset, input into a neatly formatted equation.



The calculator recognises these symbols and mathematical operations:

- Digits
- E, pi and i
- Plus, times, minus, divide
- Brackets
- Equality
- Square root
- Exponents
- Factorial

## Incomplete equations

The calculator automatically completes equations for the user so that they are automatically correct. This means that the calculator can morph and display a meaningful equation before the user has completely finished. It also shows the user where they are missing operands and gives the user time to think about what they want to write.

$$\frac{4^2}{1} = 16$$

## Editing an equation

### Writing

Once an equation has been recognised and morphed it is still editable. A user can continue to write on top of the typeset equation. The altered equation is reparsed to recognise the new equation.

$$\frac{4^2}{2} = 8$$

$$\sqrt{\frac{4^2}{2}} = 2.828$$

The computer added corrections in red are removed as soon as a user starts editing. A user can therefore write over the top of the computer's correction. By editing the equation in this fashion a user can build up a mathematical expression in stages, checking at each point that the correct expression is recognised.

### Drag and drop

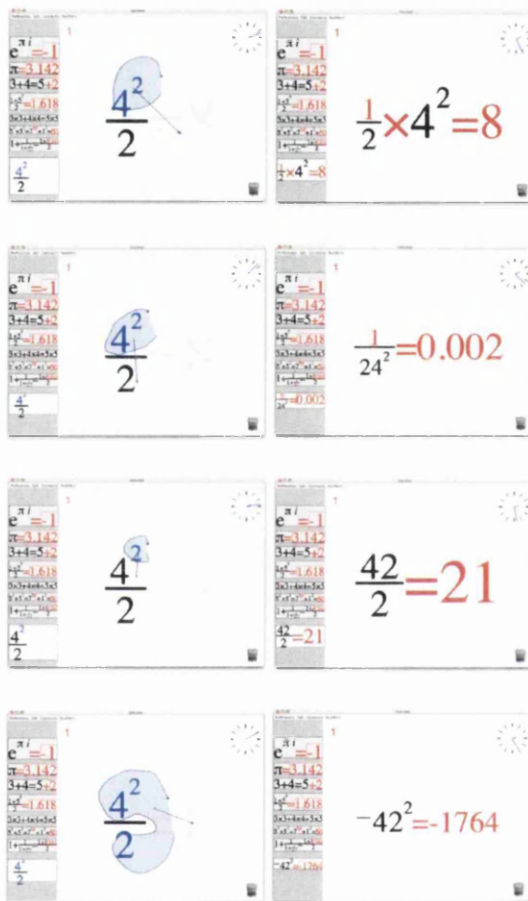
An equation is also edited in the equation editor by drag and drop. Selection is done syntactically; a user can select non-semantic or non-meaningful parts of an equation. For example a user can select the first and last digits of a number, or the numerator and denominator but not the divisor line.

$$\frac{4^2}{7}$$

To drag and drop parts of an equation, first part of the equation is circled. The calculator automatically highlights the area and symbols you are selecting. Once part of the

equation is selected the pen or finger is lifted, then dragged from within the selected area to where the user wants to drop the selected part of the equation. When let go the selected contents are cut from the equation and pasted at the drop point as if they were very small.

Thus with this method parts of the equation can be moved around easily. Mathematics can be dropped underneath square roots or divisors, or placed as exponents. However a divisor bar cannot be dragged on top of an expression because the calculator does not know how large you want the bar to be. The solution is to drag the expression underneath the divisor bar.



## Deleting

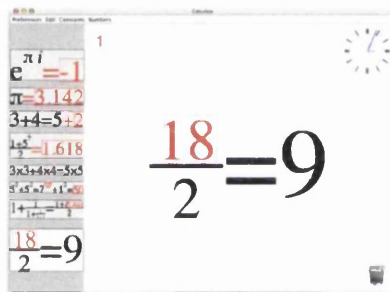
Part of an equation can be deleted by dragging it to the trash. The selected part of the equation is removed and the equation is reparsed. A cloud of smoke is used to show that you have deleted part of the equation.



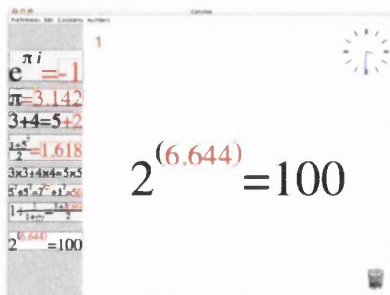
## Equal opportunity

The calculator uses a method, of solving partially complete equations, called equal opportunity. This allows both incomplete equations to be parsed sensibly, as described in the section *Incomplete equations*, and allows the calculator to compute more complicated results in a way which is simple and makes sense to the user.

With a normal expression entered by the user, the calculator adds an equality followed by the answer. If the user enters an equals sign then it is possible to write on both sides of the equality. This is where equal opportunity is useful. The computer corrects incomplete equations so that the least amount of change takes place.



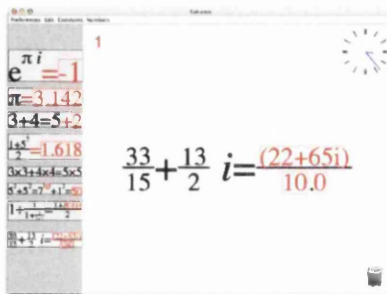
Thus simple problems are solved without any rearrangement for the calculator. And more complex problems can be solved without an understanding of the mathematics.



In the above example a bracket is used to indicate that the calculator should put a number here. Without the bracket a simple addition or subtraction on the right hand side is used to ensure the equation is correct. In this way the calculator is declarative, no equation ever shown to the user is mathematically incorrect.



Multiple missing values are sensibly filled in, for example rational numbers are used for divisors. (Note: it is necessary to indicate that a horizontal line is a divisor by placing a bracket or decimal point above or below it)



### Using the dock

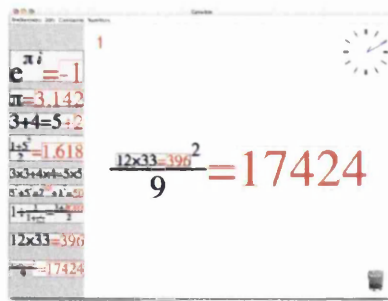
The equation dock on the left hand side of the screen can be used to store and retrieve equations. The equation editor always shows the contents of one of the equations stored in the dock. This equation has a white background, the rest of the equations have grey backgrounds.

To retrieve an equation from the dock, you simply click on it. The equation editor will then switch to that equation.

Dragging to the dock saves the selection creating a new item in the dock. The selection can be either an equation or part of an equation. The selection is copied to the dock and the equation being edited remains unchanged.

In the dock clicking and dragging on the equations drags the equations around. To delete the equations these equations can be dragged to the trash.





An equation can also be inserted into the currently editing equation by drag and drop. It appears as it does in the dock, and is immutable. More than one copy of the same equation can be inserted into the editing equation.

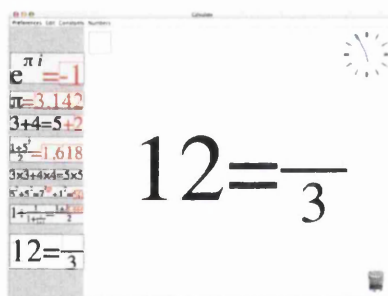
### ***Using the history***

By rotating the hand of the history clock, the entire equation history can be viewed. This is similar to a “scrubber” or slider on a movie player. The smooth morphing and movement of the symbols and expression can be played backwards and forwards at any speed.

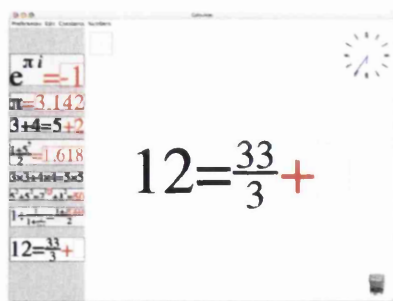
In this fashion a user can review the entire working to arrive at a solution. A user can also use the clock to undo mistakes, once they have released the clock and started writing the backwards “time-travel” is made permanent and the future recording is erased.

### ***Hiding the answers***

The numbers shown button, toggles the computer-generated numbers on and off. When the numbers are not shown they are drawn as empty dashed boxes. This enables a teacher or presenter to pose questions about the mathematical formulae without revealing the answers.



Whilst the computer-generated numbers are hidden, the calculator remains usable. This enables a user to attempt to enter the answer to find out if it correct. If it is not correct the calculator has to add in a correction, which is also hidden. When the user gets the correct answer no correction is added and the whole equation is black.



## Appendix G

### Dock equations

The dock text file syntax is solely a positional syntax. Each character represents a symbol added to the mathematical expression at the current cursor. Every symbol moves the cursor right except curly or square brackets which shift the cursor up or down respectively. Any symbols written within these special brackets take up a single symbols width.

Thus, {12}[3]- is recognised as the symbols 12 raised, the symbol 3 lowered and a - symbol, all horizontally aligned with the same width. Which when parsed becomes  $\frac{12}{3}$ .

The initial dock equations text file is:

```
3+4=5
3*3+4*4=5*5=
32+{9}[5]-*=
=2.54*(+12*)
1+{1}[1+{1}[1+{1}[1+1]-]-]-={1+R}[2]-=
P=/
!=1*2*3*4*5
0=e{iP}
(1+ [100]- []){12} =(1+[100]- []){1}]
```

Which gives the resulting mathematical expressions:

- $3 + 4 = 5$
- $3 \times 3 + 4 \times 4 = 5 \times 5 = ?$
- $32 + \frac{9}{5} \times ? = ?$
- $? = 2.54 \times (? + 12 \times ?)$
- $1 + \frac{1}{1 + \frac{1}{1 + 1}} = \frac{1 + \sqrt{?}}{2} = ?$
- $\pi = ? / ?$
- $?! = 1 \times 2 \times 3 \times 4 \times 5$
- $(1 + \frac{?}{100})^{12} = (1 + \frac{?}{100})^1$

## Appendix H

# Recdit draft paper & timeline of chapters

This section includes a draft paper on Recdit, a novel text editor inspired by some aspects of the calculator. The majority of this thesis was written in this editor, and following the draft paper are the timelines generated by Recdit for the creation of each of the chapters of this thesis.

# Recdit: A Text Editor With a History

Will Thimbleby  
FIT Lab  
Computer Science  
Swansea University  
Singleton Park  
Swansea, SA2 8PP, UK  
will@thimbleby.net

## ABSTRACT

A novel text editor, Recdit, provides a complete character by character history of text documents is introduced and discussed. Recdit provides the ability to see an entire document's history and to "scrub" through it like a movie.

Recdit also provides highlighting and graphs that tracks and shows overviews of a document's edits and changes. The ability to do this provides many novel uses. These features are discussed and are useful for both single and multiple authors, both of which Recdit supports.

**ACM Classification H5.2** [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

**General Terms** Design, Experimentation

**KEYWORDS:** Undo, history, versioning, track changes, collaborative writing

## INTRODUCTION

Tracking the changes in text documents is an essential task [3], especially when multiple users are editing the same document. Recdit provides a character by character recording of the entire history of text documents. This history provides the ability to extensively track changes in text documents.

The primary contribution of this paper is the combination of a complete edit history, its visualisation and a controlling user interface. These provide novel ways of tracking document changes and interacting with them.

Some of the features of Recdit provides are:

- Multiple concurrent authors
- A complete edit history
- A slider user interface to control viewing the history
- Trails which highlight the last few edits
- Graphs providing an overview of the entire history
- Sideways text layout that provides more structure for the text

Recdit is a fully working text editor designed for Mac OS X. This paper was written in Recdit of which every edit can be reviewed. The application and paper are available to download from <http://will.thimbleby.net/truertext/>

## INSPIRATION

This text editor was inspired primarily by seeing users enjoying the undo-clock in the pen-based interactive calculator presented in [6, 7].

The calculator instead of providing a discrete step-based undo similar to most modern user interfaces, provides an undo that is linear and smooth. The undo user interface of the calculator is presented to the user as an analogue clock that allows the user to manually set the time by rotating the clock's hands.

Users really enjoyed interacting with the undo-clock when using the calculator. Like most undo systems, users did not often use the undo feature, but they still liked to play with it and enjoyed the interaction.

This success led to the original design question for Recdit: "What would a text editor look like if it had a similar undo-clock?"

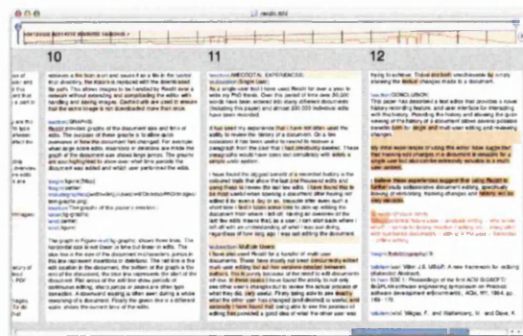


Figure 1: Recdit's user interface.

Time-machine computing [5] describes a whole system containing multiple applications which can be navigated in time, Recdit provides the same navigation in a more focused and refined user interface specifically for textual documents.

## DESIGN

### Implementation

To provide the similar ability to rewind the edits to a text document as the calculator provides for equations, Recdit records every single event as the user type. The recorded event data includes a timestamp, the current user, and the edit data itself. As a user types, each modification, the document records an event, every single character press is recorded as a individual event.

The edit history of a document is recorded in the saved files so that the history of a document is not lost between editing sessions.

When a document is opened in Recdit the document replays its creation from the start. This allows keyframes of the document's state to be recorded throughout the its creation. To jump to a location in history the document is "wound" forward from the nearest keyframe by replaying the edits made between the keyframe to the new location. The keyframes that are stored every couple hundred edits allow jumping to any location in the document's history to be very fast and allow scrubbing through the history to be immediate and interactive.

### User Interface

A screenshot of editing this paper in Recdit is shown in Figure 1. The text is laid out sideways in individual pages with page numbers at the top. Some highlighting of this text is visible showing the last few edits made to this paper when the screenshot was taken. At the top of the window is a graph of the documents state throughout its creation and a slider that controls the currently displayed version of the document in history.

If the user does not interact with the history features of the editor then the editor interacts identically to a simple text editor.

The main user interface for interacting with the history of the document is provided by the slider at the top of the document window. Using this slider it is possible to jump to any point in the document's history and to scrub back and forth to see the changes as they were made. Dragging the slider left moves the document back in history, up-to the beginning of the document, and dragging it right forward in history. This works like scrubbing through a movie. As the slider is moved the document is updated instantly, the state of the document and the slider are never inconsistent.

A slider is used instead of the undo-clock used by the calculator because it provides instant interaction for moving to specific points. Another benefit of a slider is that it can be overlaid on-top of graphs showing the state of the document at any time. A disadvantage of a slider is that as the document has a history in the thousands or tens of thousands edits the slider becomes more inaccurate, each pixel of the slider's position representing many edits.

Using the slider to scrub back and forth in the history of the document shows the edits that created the document. This can help a user to understand the process of the document's creation.

By replaying a document's creation when it is opened, the user sees the document recreated from the beginning character by character. This provides the user with a fast-forwarded reminder of how the current state of the document was reached.

### MULTIPLE USERS

Group editing documents with multiple authors even with version control systems or concurrent group editors is not a simple task. Other than the technical problem of providing distributed access to the same document, one of the main problems of multi-user editing is keeping track of the changes made by other users [3]. Simple versioning is often used to provide a basic tracking of changes.

Recdit provides networked multi-user concurrent editing capabilities. A document can be served from a server to multiple individual users that connect to the document. This allows multiple users to concurrently edit the document at the same time.

By recording the history of a document Recdit creates the potential to review other user's changes. Allowing a user to potentially rescue paragraphs deleted by other users and to "see" their changes, to see how and what they wrote, corrected and deleted.

### UNDO vs. HISTORY

Most undo systems create a tree of edits, usually each branch of the tree except the main current branch is lost. Several interesting undo frameworks have been proposed like US&R [9] and multi-user undo systems like [1]. Recdit is not focused on implementing an undo system. The history and undo are distinct, which makes both simpler to interact with.

The history records all the undo and redo changes, because the undo system is external to the history of a document. Moving back in time it is possible to see mistakes made and undone.

Unlike undo the history of a document is immutable, once an edit has been made it is recorded forever. Although it is possible to view the history of a document it is not possible edit the document in the past.

The history of the document can be represented as a sequence of states on a time based axis. The version of a document that existed at any point in time can be retrieved. No state the document was previously in is irretrievable, at no point is any data ever lost.

### TRAILS

As the user types in Recdit a coloured trail is left behind. This appears as a light background colour behind the text, which can be seen in Figure 1. As the a continues to type this colour fades out over time until it disappears after one thousand edits. This means that at any point in the documents history the highlighted trail provides a overview of what the last one thousand edits are. Deletions are not highlighted because they do not leave behind any text.

A benefit of only colouring the last few edits is that the majority of the document looks normal, and the gradual fading of the highlighting provides a good overview of what edits



were made and when they were made.

Edits by different users are highlighted with different colours, so it is easy to see who wrote what in the last one thousand changes in a group authored document.

### SIDEWAYS LAYOUT

Recdit lays the document out sideways, splitting the document into pages. The pages are laid out sideways in order to make best use of current widescreens which are becoming more popular and to provide additional structure to the text. This is to provide more structure to the document, such that it is easier to find where a part or page of the document is.

The pages the text is split into are sized so that they are a good size for reading and editing. Page-breaks between sections means that modifying one section will not affect the page layout of another section.

The document can also be zoomed in and out smoothly using another slider. This allows Recdit to provide a overview of the whole document, which is useful to show where edits are being made located in the document as changes are replayed. This is shown in Figure 2.



Figure 2: An overview of this entire paper, as a user can zoom it.

### LATEX

Recdit does not support rich text. All edits and the history of the document are recorded as plain text. Recdit instead provides inbuilt  $\text{\LaTeX}$  syntax colouring and built in PDF generation. This provides any rich text support. Using  $\text{\LaTeX}$  means that the architecture of Recdit is simpler, because it only needs to work with plain text. It also means that the typesetting and layout capabilities can be very powerful.

To provide network based editing support for images, without including the images in the plain text, Recdit provides a `geturl{http://...}` macro that retrieves a file from a URL and saves it as a file in the `/tmp` directory. This allows images to be handled by Recdit over a network without extending by handling and recording images. (Cached URLs are used to ensure that the same image is not downloaded more than once.)

### GRAPHS

Recdit provides graphs of the document size and time of edits. The purpose of these graphs is to allow quick overviews of how the document has changed. For example when large scale edits, insertions or deletions are made the graph of the document size shows large jumps. The graphs can be highlighted to show over what time periods the document was edited and which user performed the edits. Edit wear and read wear [4] introduced graphs that answer different questions based on edit location instead of time. Combinations of these visualisations could be interesting.

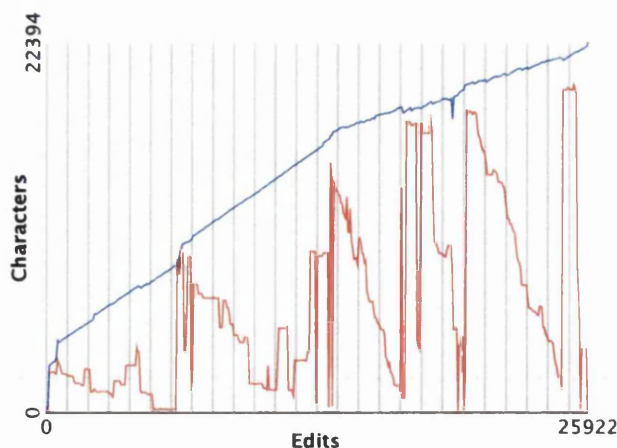


Figure 3: A graph of this paper's creation.

Figure 3 shows two graphs that are drawn of the editing history of this paper. The same graphs can be seen underneath the slider in Figure 1. The horizontal axis on this graph is not linear in time but linear in edits. The top line that travels from bottom left to top right is the size of the document in characters. A steady increase in this line represents typing, jumps in this line represent insertions or deletions, edits that create large changes in the character count. In Figure 3 this line tails off towards the end, as more of the edits made were correcting.

The lower line is the edit location in the document. The start of the document is represented by the top line, the end of the document is the x-axis. When this line is near the bottom of the graph the edits are being made at the end of the document. Flat areas in the edit line are periods of continuous typing, sharp spikes are often small corrections. A downward slope is often seen when a user is editing the document from start to end. Repeated edits like this create a sawtooth graph, like that in Figure 3.

### ALTERNATIVES

There are tools that currently provide some capabilities for tracking changes in text documents. These tools provide simple functionality, they are not capable of Recdit's character history or able to "scrub" through history.

- Microsoft Word provides a feature called *track changes*. Additions are highlighted and deletions are scored out. *Track changes* can be tedious to use, it does not show the actual edits and it creates documents that when the changes are shown, are increasingly hard to read.
- Various web based editors, like *EditLive* and *Google Docs*, provide similar tracking changes functionality to Microsoft Word. These editors often provide good group authoring support.
- *diff* is a typical file comparison utility that is used to show the changes between two documents. *diff* is often used to provide change tracking when combined with version management systems, like *subversion*. The results of *diff* can be confusing when there are large edits or even simple



rearranging. Other similar tools are more capable [2].

- Wikis provide web based multi-user authored pages, some of these provide the ability to track the changes between versions. The history of these pages can contain thousands of edits and users and can provide interesting insights into the authoring of the pages [8].
- Ad hoc emails and conversation provide the majority of change tracking for most co-authored documents. The change information is usually passed between authors in an unstructured form, with multiple versions of the same file in different places.

## ANECDOTAL EXPERIENCES

### Single User

As a single-user tool I have been using Recdit for over a year to write my PhD thesis. Over this period of time over 30,000 words have been entered into many different documents (including this paper) and almost 200,000 individual edits have been recorded.

It has been my experience that I have not often used the ability to review the history of a document. On a few occasions it has been useful to rewind to locate and recover a paragraph from the past that I had previously deleted. The process is to rewind the history until the section that was deleted is located, select and copy that section, then fast forward to the current state of the document and paste the copied section. These paragraphs would have been lost completely if I was only using a simple undo system.

The biggest benefit of a recorded history I have found is the coloured trails that show the last one thousand edits that make it easy to review the last few edits. I have found this to be most useful when opening a document after having not looked at the document for even a day or so, because after even such a short time I find it takes some time to pick-up the flow of editing the document from where I left off. Having an overview of the last few edits means that, as a user, I can start back where I left off with an understanding of what I was just doing, regardless of how long ago I was last editing the document.

### Multiple Users

I have also used Recdit to author several multi-user documents. These were mostly not concurrently edited but edited using ad-hoc versions emailed between authors, this has been partially necessary because of the need to edit documents off-line. In these cases I have found the ability to not only see other user's changes but to review the actual process of what they did, very useful. Firstly being able to see exactly what the other user has changed (and deleted) is useful, and secondly I have found that being able to see the process of editing has provided a good idea of what the other user was trying to achieve. These are both impossible by only showing the textual changes made to a document.

## CONCLUSION

This paper has described a text editor that provides a novel history recording feature and user interface for interacting with this history. The history combined with trails and graphs that provide overviews of the document history. These allowed several possible benefits both for single and multi-user

editing and reviewing changes.

My experiences of using this editor have suggested that tracking edit changes in a text document is both valuable for a single user and also extremely valuable in a multi-user context.

I believe these experiences suggest that using Recdit to further study collaborative document editing, specifically looking at versioning, tracking changes and history for different domains, will be very valuable.

## AVAILABILITY

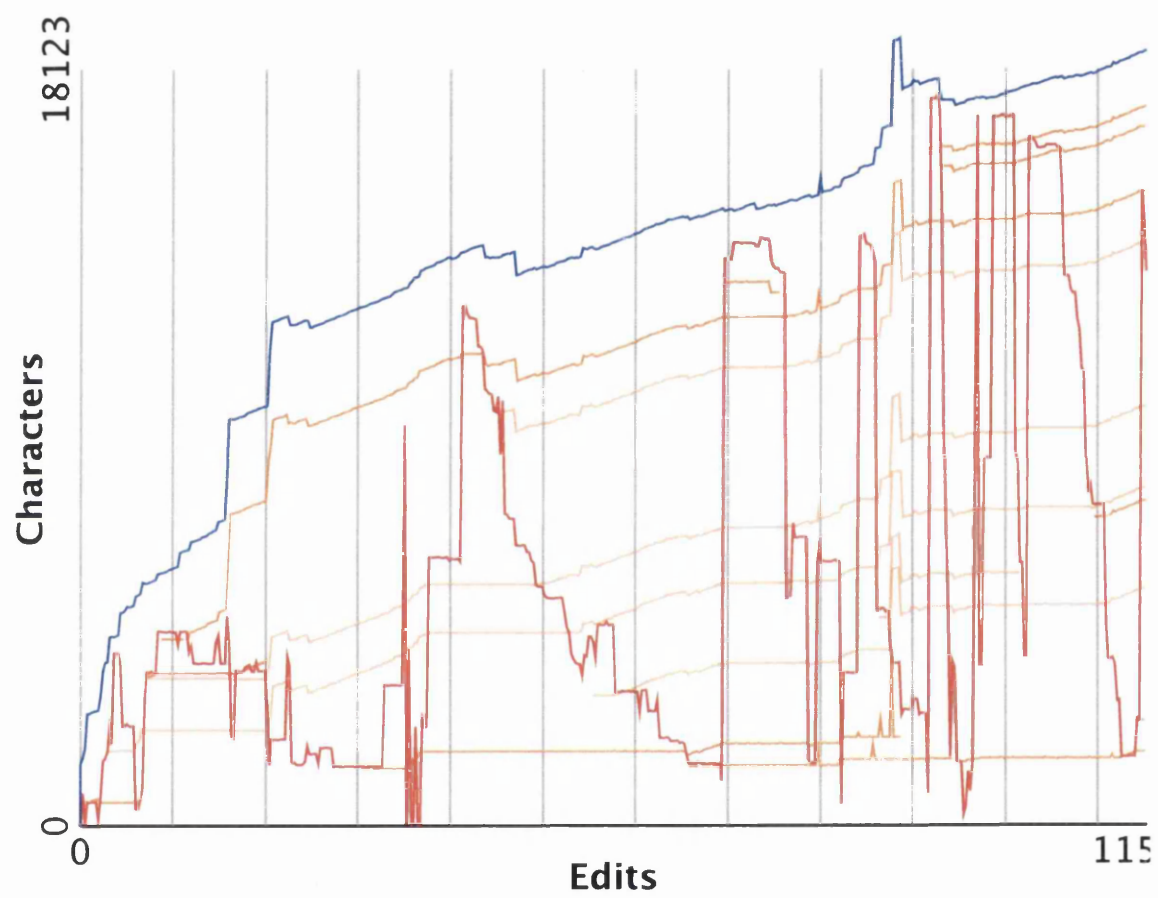
The Recdit application, this paper and a movie of its interaction are available at <http://will.thimbleby.net/truetext/>

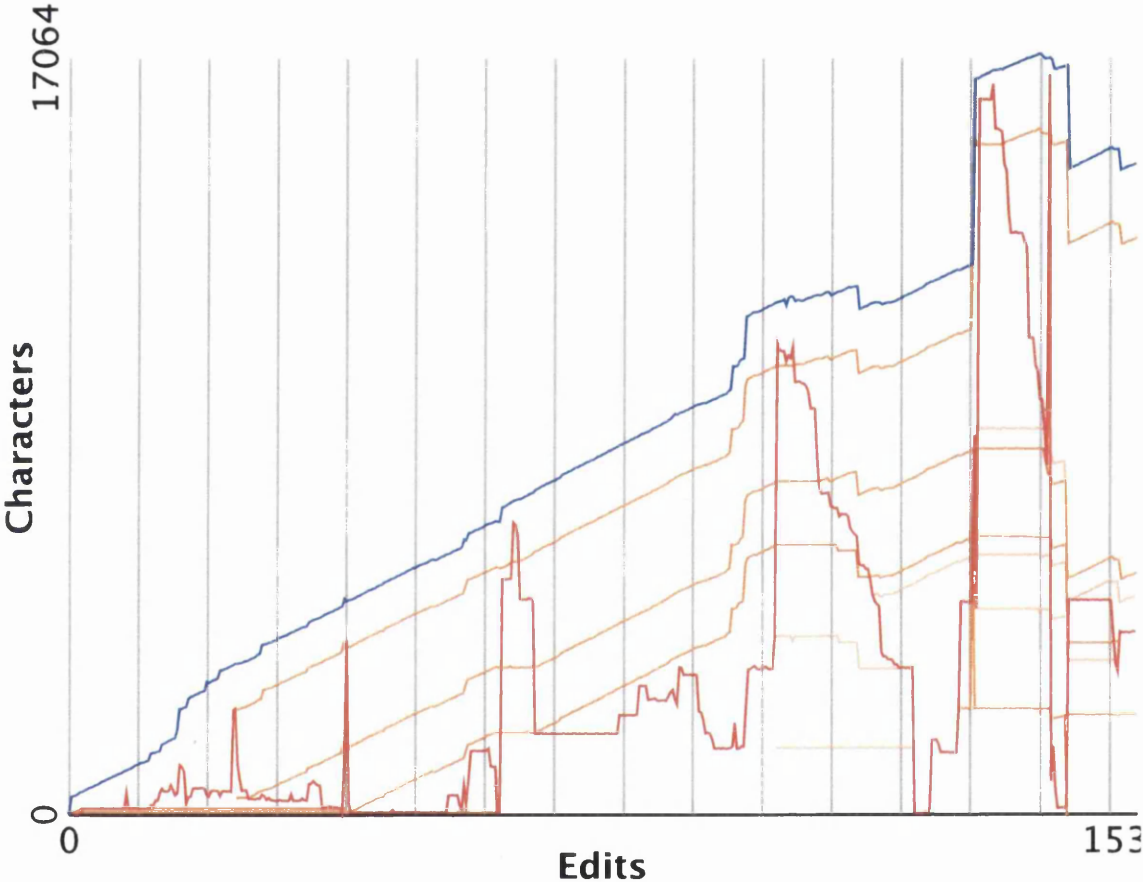
## ACKNOWLEDGEMENTS

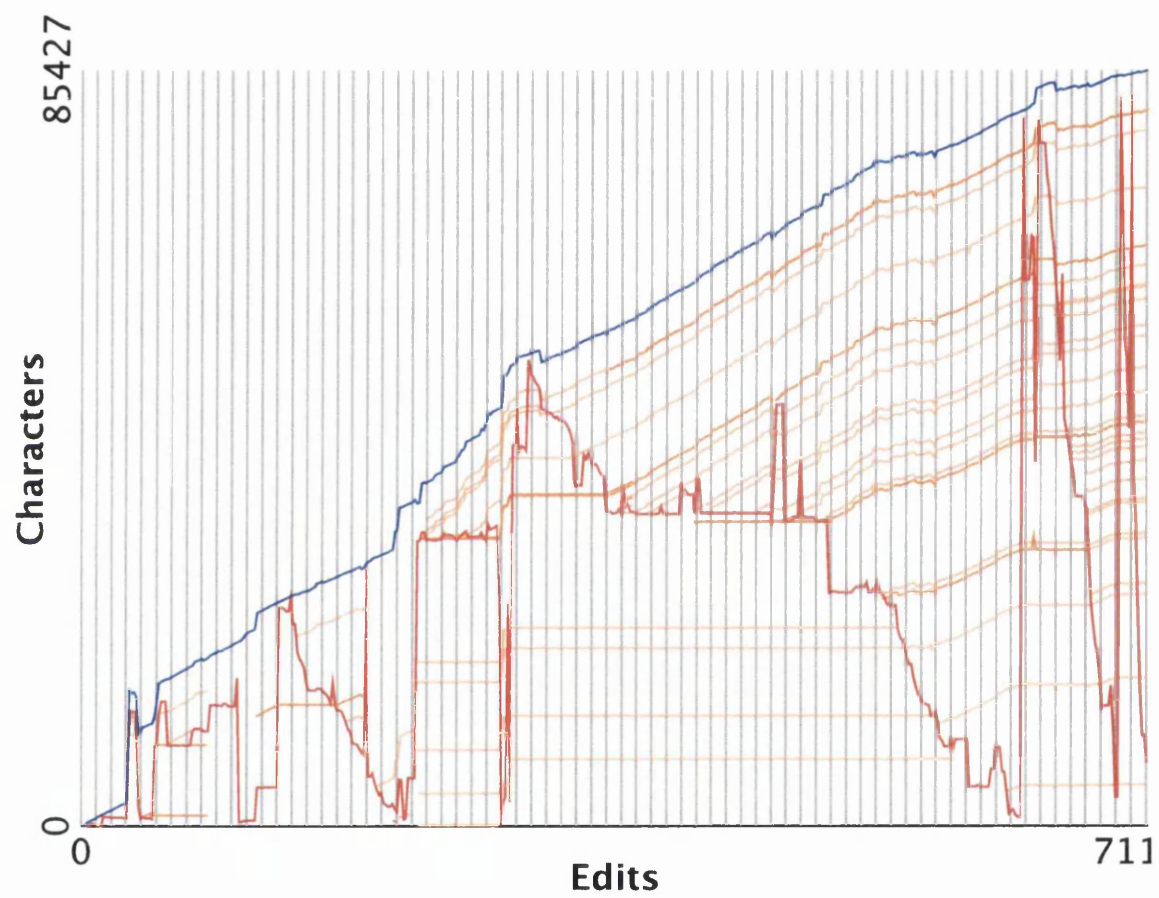
Will Thimbleby is supported by a Swansea University PhD scholarship and by Microsoft Research Cambridge

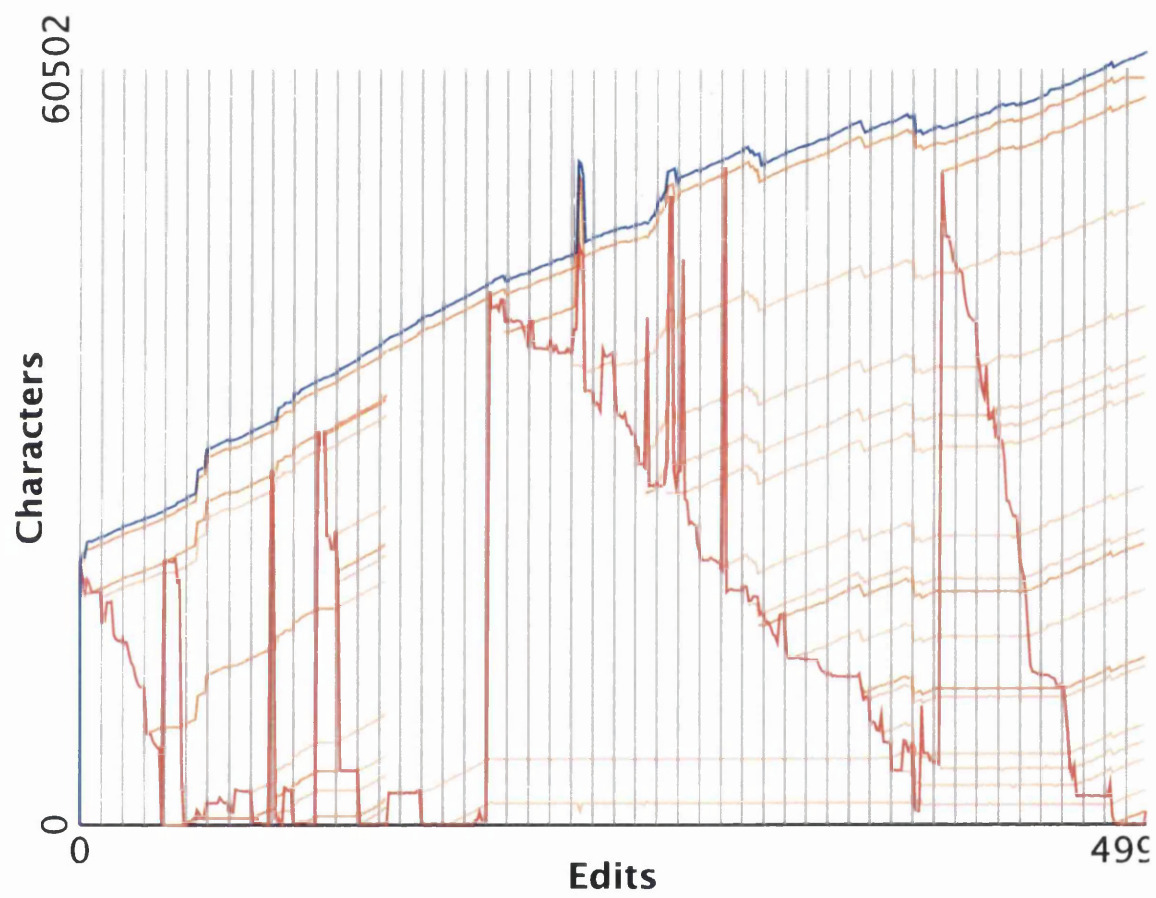
## REFERENCES

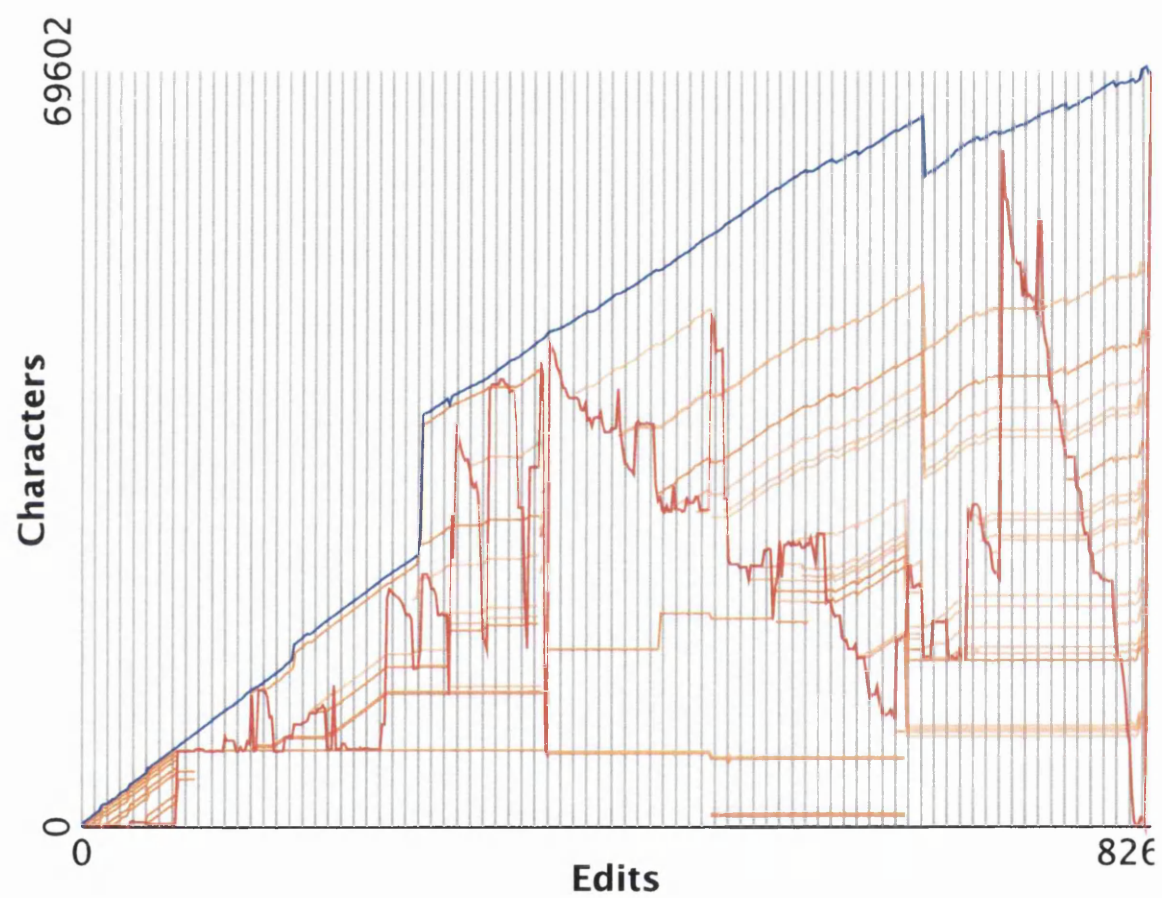
1. Chengzheng Sun. Undo as concurrent inverse in group editors In *ACM Trans. Comput.-Hum. Interact.*, Vol 9, Number 4 ACM, NY, (2002), pp. 309–361.
2. Christine M. Neuwirth C.M., Chandhok R., Kaufer D.S., Erion P., Morris J. and Miller D. Flexible Diff-ing in a collaborative writing system In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, ACM, NY, (1992), pp. 147–154.
3. Hee-Cheol Kim Eklundh, K.S. Collaboration between writer and reviewer through change representation tools In *System Sciences, 2002. HICSS*, IEEE, (2002), pp. 531–540.
4. Hill W. C. and Hollan J. D. and Wroblewski D. and McCandless T. Edit wear and read wear In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, (1992), pp. 3–9.
5. Rekimoto J. Time-machine computing: a time-centric approach for the information environment In *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*, ACM Press, (1999), pp. 45–54.
6. Thimbleby W. A Novel Pen-based Calculator and Its Evaluation In *Proceedings 3rd ACM Nordic Conference on Human-Computer Interaction*, ACM, NY, (2004), pp. 445–448.
7. Thimbleby W., Thimbleby H. A novel gesture-based calculator and its design principles In *Proceedings 19th. BCS HCI Conference*, Vol 2, BCS, (2005), pp. 27–32.
8. Viégas, F. and Wattenberg, M. and Dave, K. Studying co-operation and conflict between authors with history flow visualizations CHI, Vol. 6, No. 1. (2004), pp. 575–582.
9. Vitter J.S. US&R: A new framework for redoing (Extended Abstract). In *SDE 1: Proceedings of the first ACM SIGSOFT/SIGPLAN software engineering symposium on Practical software development environments*, ACM, NY, (1984), pp. 168–176.

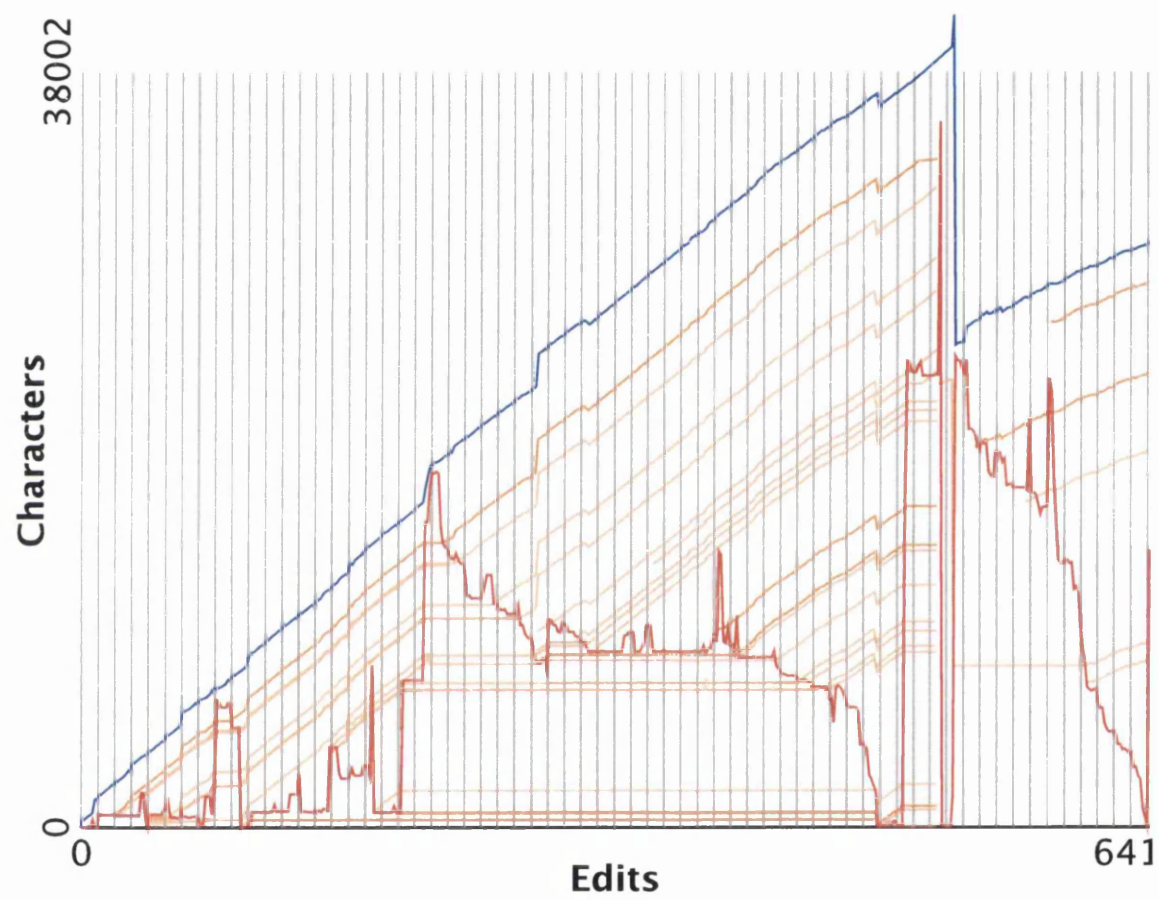




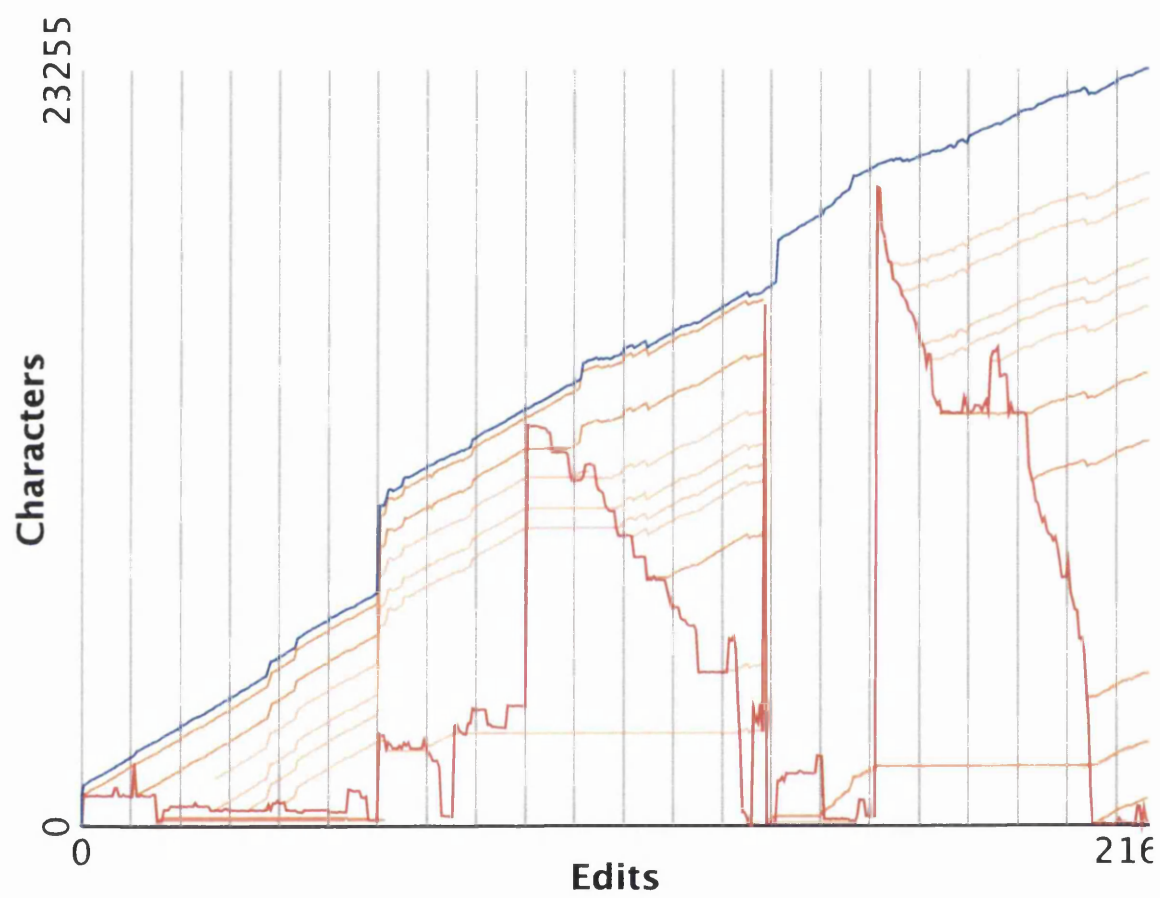




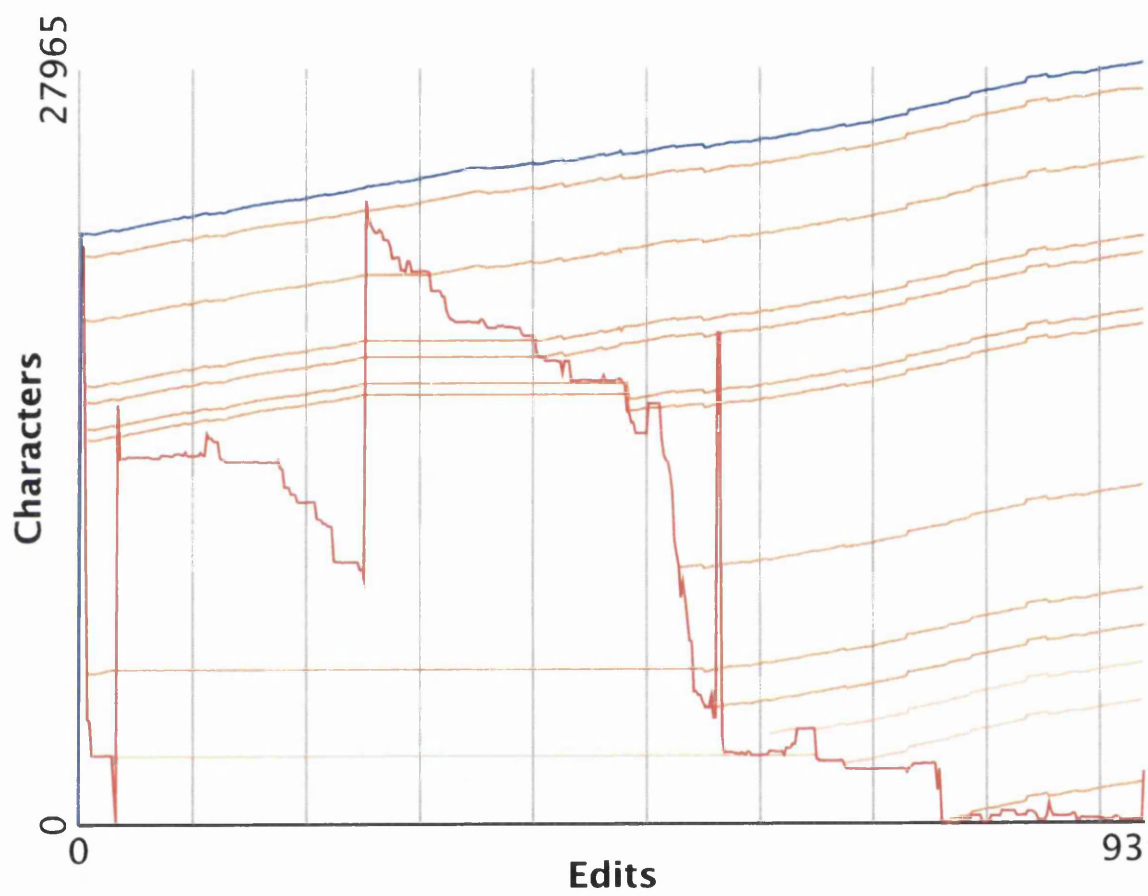


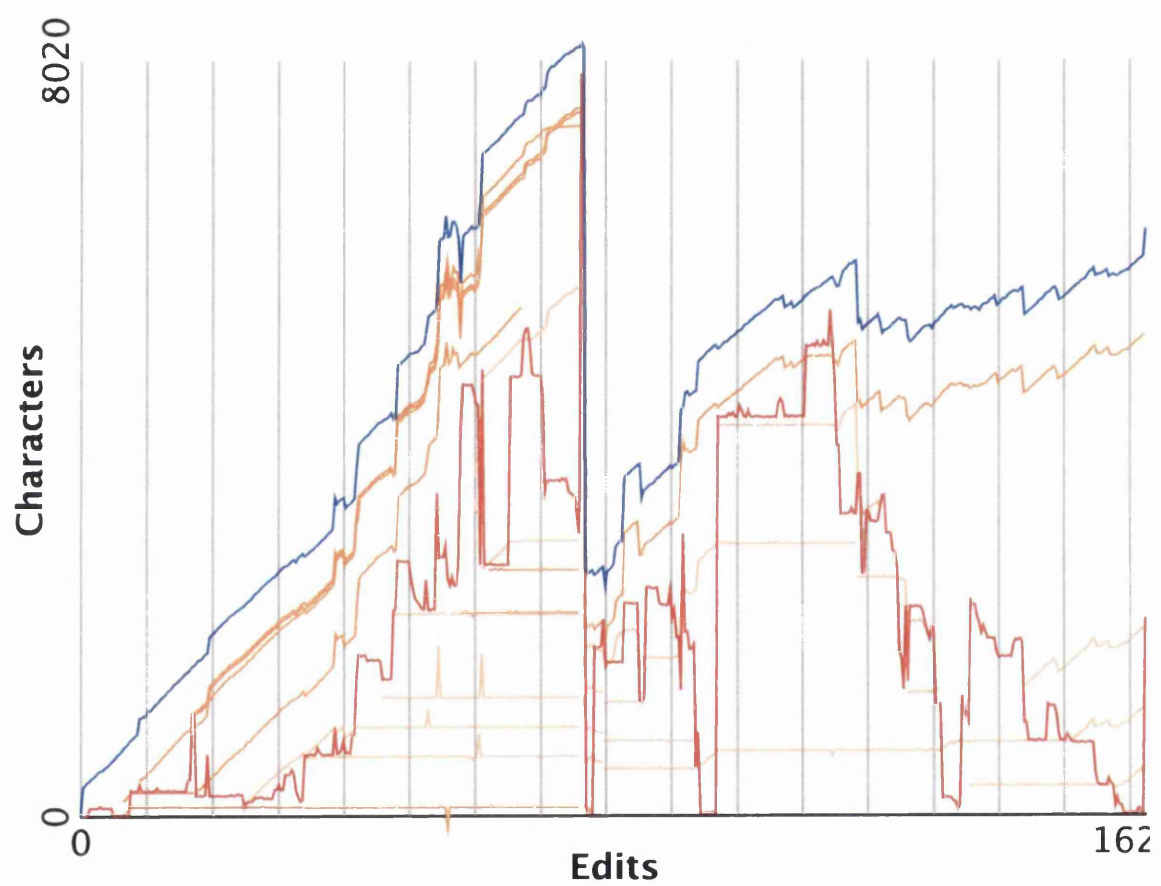


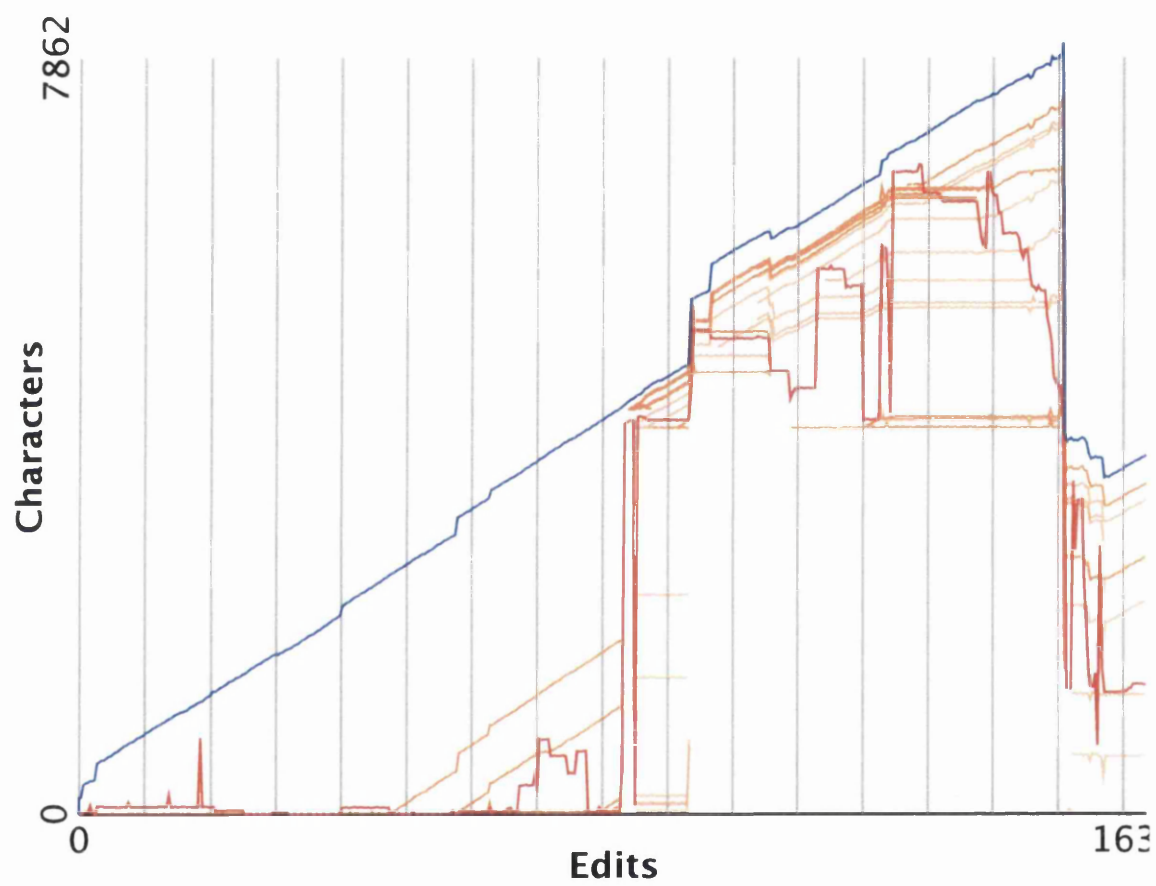












## Appendix I

### Published papers

#### I.1 A Novel Pen-based Calculator and Its Evaluation

# A Novel Pen-Based Calculator and Its Evaluation

William Thimbleby

now at Department of Computer Science  
Swansea University, Singleton Park, Swansea, SA2 8PP  
will@thimbleby.net

## ABSTRACT

A novel calculator, ideal for interactive whiteboards and pen-based devices, is introduced and evaluated. The calculator provides a natural, dynamic method of entering conventional expressions by handwriting and provides continual feedback showing the calculation and results. The user interface adjusts and copes with partial expressions, morphing the expressions to correct position and syntax. Gestures are also used to edit and manipulate calculations. The user interface is declarative, in that all displays, even with partial user input, are of correct calculations.

The new calculator is faster for more complex expressions and importantly, gives users more confidence in its use. The majority of users said that they would prefer to use this calculator rather than their conventional calculator.

## Author Keywords

Handheld calculators, gesture input, novel interfaces.

## ACM Classification Keywords

H5.2. Information interfaces and presentation (e.g., HCI): User Interfaces.

## INTRODUCTION

Imagine writing a calculation down on paper, and the paper magically working out the answers. We have built a calculator that works like this, which is ideal for pen based user interfaces, or for interactive whiteboard use in classrooms. This paper discusses the design and its evaluation. (It will be demonstrated in the conference.)

## Overview

Refer to Figure 1 at the top of the next page, which shows screen snapshots of the new calculator in use. We first show a user doing the sum  $3+6$ . In the first screen shot, they have written  $3+6$ , in the next the calculator is catching up with them and has already rendered the 3 and + in a printed font. In screenshot 3 the calculator has morphed the input into a

Paper presented at the ACM NordiCHI Conference 2004, the biannual Nordic HCI conference, held in Tampere, Finland. Cite as W. Thimbleby, "A Novel Pen-Based Calculator and Its Evaluation," *Proceedings of the third Nordic conference on Human-computer interaction*, pp 445–448, 2004.

nicely typeset equation. The user then clears the screen, screenshot 4, using a cross-out, X, gesture; the feedback from deletion, the 'smoke' feedback is also visible. The final two, 5 and 6, screenshots show the user entering  $2/3 \times 4$ , the declarative calculator ensures the answer is correct, and the interface morphs the answer into a nicely typeset and readable equation.

## History and Motivation

We have always used instruments to aid our mental arithmetic and to help us with mathematics. Somewhere around 200AD, the abacus was invented, and in the 1970s with the development of electronics, electronic calculators became popular. For the most part their design copied earlier mechanical calculators. Now, thirty years later, when desktop and handheld computers can do almost anything, today's calculators merely imitate early electronic calculators. The calculator provided in the Start menu by Microsoft is less powerful, and less expressive, than a 10 year old handheld calculator! Yet computers today could do a lot better than just simple imitations of mechanical calculators.

The majority of current research on expression recognition has been directed towards that of expression entry [1,2,3], although there have been attempts to marry expression entry with calculation (for example, the PenCalc project [4]) Yet, none of the existing implementations have attempted to use expression recognition itself as a user interface for a calculator.

The calculator presented here, and its design extend the domain of calculator user interfaces into the 21<sup>st</sup> century. Rather than relying on obsolete metaphors that dictate awkward and unnatural mathematical entry, this calculator provides a natural interface that is designed to (and does) function like pen and paper — or, rather, paper that does mathematics magically.

## PEN-BASED USER INTERFACES

The main advantage of a pen-based system is the familiarity of the interface. The majority of users are already competent at writing with a pen. This advantage is greater with mathematical expressions, because the majority of mathematical work is still done on paper with a pen or pencil. Using a pen-based system to enter expressions is a natural progression, as it means that anyone can use it with little or no training. Pens can replicate the complete functionality of both keyboard and mouse, enabling computers with a sole input device.

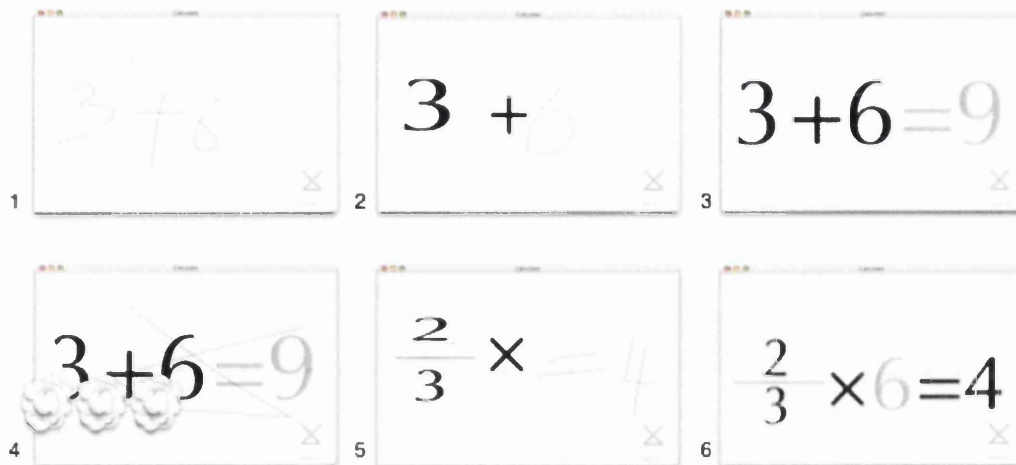


Figure 1. Screenshots of the calculator's progress computing,  $3+6$  and then  $2/3 \times 4$ .

Meyer [6] gives a good detailed overview of the whole technology, including both a history of pen based computing and more technical aspects of the hardware and software.

#### IMPLEMENTATION

The implementation is written in Java, and is split into two modules. The first provides basic symbol recognition using a model-matching algorithm. The second recognises the equations using a recursive descent algorithm based on Chang [5]. The interaction and user-interface are layered on top of these two modules

#### USER INTERFACE REQUIREMENTS

Ideally a user interface for mathematical input should provide a superset of paper's functionality, allowing a user to use the interface in the exact way they would use paper. The key features of the paper metaphor are outlined below.

##### Sketching

Ideally a user could draw rough sketches. Often when solving a problem a user will not jump straight onto their computer and solve it, but jots down diagrams or figures first. The system presented does not implement sketching, however a solution is to specify areas for diagrams and mathematics.

##### Expressions

The system should allow the user to enter expressions as they would on paper, without unnatural restrictions. For instance, the user should not be forced to enter the expression in a strict fashion.

##### Editing

It should be possible to edit expressions at any time. Both input and output expressions (that is, an expression just entered, and one that has been computed) should be treated in the same manner. High-level editing, such as rearrang-

ing, insertion and deletion should be possible. To implement these with a pen-based interface, without leaving the paradigm of pen and paper, requires special gestures that are assigned to each of the editing functions. For example, a scribble is used for deletion. Character editing is different and involves correcting the computer guess at the semantics of a set of strokes.

##### Feedback

The system should always keep the user informed about what is going on, providing appropriate timely feedback.

#### THE DESIGN

Our new calculator uses a single canvas for mathematical expressions, which enables us to create a completely modeless interface that is intuitive and natural. The user interface is shown in Figure 1. The operation of the user interface was developed from Thimbleby [7].

The one adornment of the interface is a delete gesture reminder in the bottom right corner.

##### Expressions

The system imposes small timing constraints. It requires the strokes of contiguous symbols to be written sequentially within a small amount of time. This allows the interface to recognise the user's input on-line as they enter it. These small restrictions were found to be unobtrusive, and not to affect the user's writing style.

##### Editing

After testing several different ways of editing expressions, it was found that for the majority of mathematical expressions, the easiest way to edit them was to delete and rewrite portions of the expression. This keeps the interface very simple. (Dragging or pop-up menus would create areas of the screen that function differently from each other.) By allowing only a simple delete gesture, no mode changes are

necessary. Every part of the screen or virtual paper acts like paper: every click and drag draws.

**Feedback**

The visibility of the system’s status is provided through two kinds of feedback: annotation and morphing. As the user is writing, the system can process in the background. As a symbol is recognised the user is informed of this recognition by visual feedback: a typeset character stretched to the stroke’s bounding box replaces the written strokes.

Morphing starts after a short time delay from when the user stop writing. This halts when the user starts writing stopping it from distracting the user and from rearranging expressions as they are trying to enter them. The morph formats the entered expression into a correctly typeset equation by moving the symbols as little as possible from the user’s writing. The morph provides continuity between the user’s input and the typeset equation that allows them to continue to edit and use it.

**EVALUATION**

A total of nine participants, 5 students and 4 members of the public, took part in the usability testing. These ranged in ability from a mathematics student to people who rarely use mathematics. The testing comprised of a number of mathematics questions based on old GCSE papers. Users were given time to familiarize themselves with the interface. When they were happy, they were observed and recorded whilst attempting to complete the questions using the new system and their own pocket calculator or one provided (a Sharp EL-531GH DAL).

An observer was present and users were encouraged to discuss problems with them, afterwards the users filled in a short anonymous questionnaire.

Measurements were recorded of the time taken and the number of errors or problems encountered entering expressions. The questionnaires provided a better general impression of the ease of use.

**RESULTS**

**Time on Task**

For the simpler sums, like  $9 \times 2/3$ , a handheld calculator was much faster than the new system. An average of 24 seconds for the new system compared to 10 seconds for the traditional handheld calculator. This was expected. The majority of users were familiar with handheld calculators, and had used them over many years.

Two of the tasks were actually faster on the new system. Calculating Figure 2, was faster (an average of 49s to 79s) because users could enter it “as they saw it” rather than having to search for buttons on a calculator.

$$\frac{2^2}{21-4}$$

**Figure 2. A simple equation.**

For the task “What power of two is 28?” *every* user was able to complete the task on the new system, yet most struggled to solve it on a handheld calculator. Solving it is easy, in a similar way to Figure 1 picture 6, using the novel declarative approach from [7].

Thus this new calculator enables users to perform mathematics that they could not do before. Furthermore, it is faster for more complicated expressions because users did not have to rearrange the expression in their head. This was the not even the case for those who knew the rearrangement  $\log 28 \div \log 2$ .

**Accuracy**

A large part of the time taken to complete the tasks was taken up with recovering from symbol recognition errors. Currently the accuracy percentage in this prototype (81.1%) is poor, but easily improved. This significantly lowers the usability of the system. Expression recognition caused only a very few errors, mostly caused by short divisor bars.

However, when calculating mathematics, input accuracy is not the most important consideration; output accuracy is. **No user got the wrong answer for any question with the new system.** In contrast several unnoticed mistakes were made with the traditional calculators.

By displaying the computed equation in an easily understandable two-dimensional format, it provided the feedback necessary to understand what was being computed. Thus users knew when their calculations were wrong with the new calculator.

**Ease of Use**

The overall impression from users was that the new calculator was *easy* to use. Typesetting and feedback though morphing successfully allowed the user to understand what the calculator was doing.

No user had trouble editing expressions using the delete gesture. Other editing functions like cut and paste were never missed and users liked the modeless interface and the simplicity of one function.

**Fun**

Several users wanted to carry on playing with the system and asked when they could get their own copy.

**FURTHER WORK**

During both testing and design many ideas were developed that provide possible avenues for further development. The more interesting are outlined here.

**Extended Vocabulary**

Expanding the number of symbols recognised to include symbols like  $\pi$ , letters, and other Greek characters, would enable the system to handle more complex expressions.

Additionally, extra functional vocabulary would allow the system more power and expressiveness. For instance trigonometric functions, user defined constants, logarithms, and factorials.

**The User Interface**

Further additional features of paper (for instance, sketching) would add to the usability.

Users specifically requested two additional features. A clear button clears the whole screen, a similar metaphor to starting a new page. This could be provided as a simple gesture or an external button.

Secondly, users found that there sometimes was not enough room to enter their additional symbols into an existing expression. Two solutions for further work would be, the addition of an insert space gesture that adds in a gap into an expression and the re-morphing of an expression as a user writes to accommodate the user's input.

**CONCLUSION**

At its most abstract, this paper described a novel pen based interface for any application. This paper described and evaluated the pen-based interface for a dynamic, on-line mathematical calculator.

New user interface concepts for the computation of mathematics were introduced, including:

- The combination of pen user interface with an on-line calculator.
- Extra space added to calculations, like longer division bars, to aid the addition of more symbols.
- The use of a single delete gesture to edit expressions, making the calculator completely modeless and providing the user with an extremely simple interface.

The comparative user testing comparing the new system with traditional calculators showed that:

- Answers produced by the new calculator were more accurate. In contrast, users failed to notice when a traditional calculator gave them a wrong answer — errors that they noticed when using the new calculator.
- Users were able to calculate the answer to problems using the new calculator that they could not solve using traditional calculators.
- Users are able to obtain accurate answers and have greater confidence in those answers compared to results from traditional calculators.

- A pen based calculator is more intuitive, fun, and easy to use than traditional calculators.
- The pen is a suitable device for entering and editing mathematical expressions. Additionally, more complex editing operations than delete are neither necessary nor missed.
- For complex calculations, the new design was faster than using traditional calculators.

It is hoped that the creation of this new calculator will prompt people to rethink the methods by which we do mathematics. (The calculator is available on the web for others to build on.) Calculators are currently restricted by obsolete metaphors, as the testing and creation of this new calculator has shown.

Ultimately, the calculator should be ported to and tested on pen based, handheld computers and tablet PCs, as well as in school classrooms (e.g., using projectors and touch-sensitive whiteboards) where they would be an ideal way of teaching mathematics to children.

We are confident that the prototype described in this paper charts a course in the right direction.

**AVAILABILITY**

A movie of the calculator and the Java application are available at <http://www.ucl.ac.uk/usr/will/>

**REFERENCES**

1. Dorothea Blostein and Andy Schüerr. Computing with graphs and graph transformation. *Software Practice and Experience*, **29**(3):1–21, 1999.
2. Yuko Eto and Masakazu Suzuki. Mathematical formula recognition using virtual link network. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition (ICDAR '01)*, page 762. IEEE Computer Society, 2001.
3. Steve Smithies. *Freehand formula entry system*. Master's thesis, University of Otago, Dunedin, New Zealand, May 1999.
4. Pencalc <http://www.cs.ust.hk/pencalc/>.
5. S. Chang. A method for the structural analysis of two-dimensional mathematical expressions. *Information Sciences*, **2**(3):253–272, 1970.
6. A. Meyer. Pen computing: a technology overview and a vision. *SIGCHI Bulletin*, **27**(3):46–90, 1995.
7. H. Thimbleby. A new calculator and why it is necessary. *The Computer Journal* **38**(6):418–433, 1995.



## **I.2 A Novel Gesture-based Calculator and its Design Principles**

# A novel gesture-based calculator and its design principles

Will Thimbleby & Harold Thimbleby  
Department of Computer Science  
University of Wales Swansea  
[will@thimbleby.net](mailto:will@thimbleby.net) [harold@thimbleby.net](mailto:harold@thimbleby.net)

**A novel calculator, designed primarily for interactive whiteboards and pen-based devices, provides a better task fit than conventional approaches. The calculator provides a natural, dynamic method of doing calculations by handwriting using conventional notation. This paper discusses the calculator's underlying design principles, which collectively create a coherent and innovative 'look and feel.' The principle set could be used to help improve user interfaces for other domains.**

*Calculators, Design principles, Gesture based interfaces, Equal opportunity, Whiteboard interaction.*

## 1. INTRODUCTION

There is evidence that handheld calculators are difficult to use and are fundamentally non-mathematical [2, 5]. A simple example is that operators often have to be transposed by the user (consider a sum like calculating  $4 \times -5$ , which has to be entered as  $4 \times 5 \pm =$  on most calculators). Although user interfaces for calculators are constrained by ergonomics (screen legibility, button size), their implementing technology has no such restrictions. This is particularly true when handheld calculators are simulated on PCs — and the technology has moved on considerably since the 1970s, which was the determining era for handheld calculators.

We have developed a new calculator with improved task fit with mathematics, and thus we have broken out of the traditional design approach. Pleasingly, the calculator is very engaging to use — where as another paper [7] discusses its evaluation, this paper presents its new design principles. The success of the new design suggests that the principles, in their particular combinations, might be usefully applied in other domains.

### 1.1 Overview of the interface

Imagine writing a calculation down on paper, and the paper magically working out the answers. Our calculator works like this, using an approach that is ideal for gesture-based user interfaces, for handhelds with pens to interactive whiteboard use in classrooms. The calculator is written in pure Java and runs under Windows, Linux and MacOSX, and it works with standard hardware such as Mimio, SMARTboard, or Wacom tablets — it is somewhat tedious to use it with a trackpad or mouse, as it uses normal handwriting as its mode of input. It can be downloaded from <http://www.cs.swansea.ac.uk/calculators/>, where a movie of it in use is also available.

A different approach can be seen in [8], which includes some useful background this short paper does not have space for. No calculator known to us is as versatile and interactive as ours; indeed, our calculator was selected as an exhibit at the UK Royal Society Summer Science Exhibition in July 2005, and we hope at the conference to report on further insights from evaluation based on its exposure to 4000+ users.

### 1.2 Evaluation

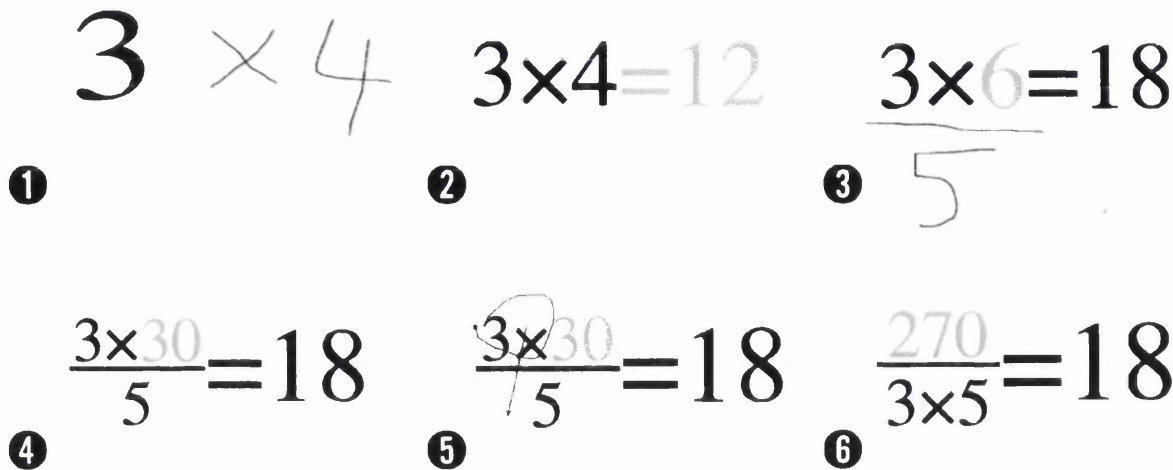
Without exception everyone (100s of people) who has used the new calculator has liked using it, and many users have found it easier to use than their own handheld calculators. Users find it fun and enjoy using it. Users with little mathematical skill enjoy using our new calculator, and some have grinned when getting it to change, for example,  $2^3=8$  to  $3^2=9$ . Sophisticated mathematical users have also enjoyed exploring issues such as why  $6!+9!$  is divisible by 100, or pushing the calculator's arithmetic (e.g., finding  $\pi$  from  $e^{ix^2}=-1$ ).

It is important to look at why the design forms such a successful user interface. The principles presented in this paper summarise our principle-led design of the calculator, and are informed by the user testing and evaluation of the prototype. We have reported elsewhere on an empirical evaluation [7].

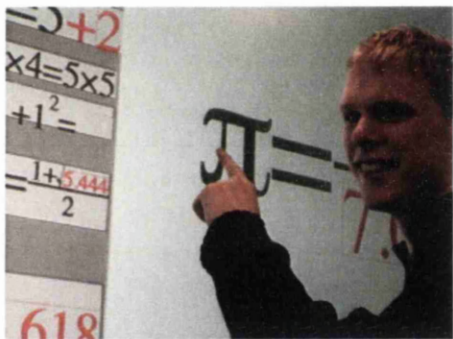
## 2. HOW IT WORKS

It is very hard to capture the look and feel of an interactive program, especially an innovative one, in a static medium like paper. This section therefore merely gives a hint of the calculator's capabilities. Figures 1.1–1.6 show a sequence of screen snapshots of it in use. They first show a user doing the sum  $3 \times 4$ ; in the first screen shot, the user has written  $3 \times 4$  and the calculator is "catching up" with their handwriting and has just rendered the 3 in a

typographically neat font. User input is handwritten blue and 'dries' black, thus it is never confused with what is already on the screen as it is written.



**FIGURE 1:** A sequence of six consecutive screen shots of the calculator solving various equations. The thin 'sketchy' text (e.g., see Figure 1.1) was written by hand, and as the calculator recognises the handwriting, it is morphed into typeset mathematics (compare Figure 1.1 and 1.2).



**FIGURE 2:** Using the calculator on a back-projection SMARTboard (6' diagonal) that permits writing using the tip of a finger.

In Figure 1.2, next, the calculator has morphed all the user's input, and immediately combined it with the output (here, '=12') and displayed it all as a typeset equation. The output generated from the calculator is shown in grey in this paper (though typically it is a colour like red in real use). The user continues to edit the equation and by the time of Figure 1.3, they have deleted the 4 and written '=18'. Effectively this poses the question "three times *what* is eighteen?" making the calculator compute " $3 \times ? = 18$ ". Additionally in Figure 1.3, we can also see the user continuing to edit this solved equation as if it were their own input; they are starting to divide the left hand side by 5.

By Figure 1.4 the calculator has morphed these changes and the combined the typeset output and the user's input into another neatly typeset equation, now showing a generated 30. Had a user wished to perform this calculation on a conventional calculator, they would have had to have entered it in a particular order and with a final = sign, such as  $18 \times 5 / 3 =$ .

In Figures 1.5 and 1.6, the user "drag selects" the " $3 \times$ " from the previous screen and drags it below the division line. (This is an "ink" edit, the " $3 \times$ " is not a syntactically nor semantically meaningful unit — see below.) Finally, Figure 1.6 shows the result of this edit, and it is mathematically instantly correct — thus providing a solution to  $?(3 \times 5) = 18$ . What has been done in one gestural operation on the new calculator would have needed around 14 keystrokes on a conventional calculator that permitted last-calculation editing (e.g., so-called twin line display calculators); moreover, at every step except the last, on a conventional calculator, the expression would be wrong, whereas on the new calculator *every* intermediate step is a valid calculation.

The calculator has other features, which are not the concern of this paper; for example, there is a wastebasket to delete anything by conventional drag & drop; a dock (visible in Figure 2 on the left) can be used for selecting, storing equations and values; there are some features used for teaching purposes; and there is an 'analogue clock' that is used for undo.

### 3. PRINCIPLES

The basic style of interaction is called *equal opportunity*, and it certainly lends itself to arithmetic calculations [5] and other applications [3]. What is new here is the effective combination of two dimensional, WYSIWYG, gesture based, instant behaviour, and morphing, that *taken collectively* make a coherent set of features that combine extremely well for the task domain (and perhaps for other domains, unfortunately beyond the scope of a short paper). But, further, the design introduces new design principles: *ink editing* and (non-trivial) *instantly declarative interfaces*.

#### 3.1 Standard HCI principles

The calculator was designed bearing in mind a range of important but conventional HCI principles: it should fit the task domain (and all that that implies) and reduce the user's short-term memory workload; etc. Conventional calculators do very much worse in supporting these principles. Additionally, our new calculator design draws on standard concepts such as undo, affordance, modelessness, and avoiding error messages (by invariantly ensuring correct partial evaluation).

WYSIWYG usually means that when you print, you get what you have on the screen. For interaction, what you see is what you *have* got is more important [6] (i.e., WYSIWYH), a variation that is a principle for interaction, not for quality display or printing, for example. For calculators the interaction problem is worse, if something is calculated based on a hidden preference of an implicit operator, will the user ever realise?

Our calculator always uses explicit operators where there could be a misunderstanding over the implicit annotation or operator, for example, it inserts an explicit multiplication between a ")" and a "(" . Rather than leave the user with their hand-written input, the calculator converts everything to a typeset, well laid-out mathematical expression, and this allows the users to know with certainty what is being computed, instead of wondering whether they have entered it correctly, whether the computer is recognising their handwriting correctly, or whether there is some invisible mode or data affecting the result.

In short, there is no hidden information or state, and all visible information is used. The calculator shows *exactly* what is being computed, thus there is no confusion for the user. Although these and other familiar principles (e.g. undo) have been consciously combined in an unusually coherent way, space precludes a full discussion of their application.

#### 3.2 Ink editing — not syntactic editing

Instead of forcing a user to think syntactically about the structure of how the mathematics works to edit an expression (even if the average user knows what that means!), the new calculator lets a user interact flexibly with the actual ink used. Of course, conventional calculators *very* severely limit what editing is — it is limited to appending characters or deleting the last number or the *entire* calculation.

The new calculator does not restrict what is selected to be adjacent or to have any particular syntactic structure; it is easy, for instance, to select alternate digits out of a number and move them elsewhere in an equation. Although that seems contrived, the ease and naturalness is important: consider editing 31.416 to 3.1416, which can be done directly by moving either the decimal point or the second digit. (In contrast these two operations would, if permitted, be very difficult in a syntactically constrained editor.) On conventional editing calculators, what is a single operation here has to be broken down into a more tedious sequence of operations like delete-move-insert or equivalent.

Although the two-dimensional notation of mathematics implies many syntactic relations, the new calculator does not impose any. Instead, ink editing allows the user to edit their work naturally as a picture, in a way that is impossible with any one-dimensional (conventional) representations. This flexible ink editing of two-dimensional notation allows the user to rearrange and edit mathematics semantically or syntactically, in a way that can be very close to how the user thinks about the abstract mathematics. The advantages of picture editing were forcefully described in [1].

WYSIWYH means that the semantics of any expression is directly linked to the syntactic "ink." For example, drawing a horizontal line might mean either a division sign or a subtraction, depending on what the user means. The calculator disambiguates, by allowing either interpretation, *which can be changed at any time*. If numbers are written above or below a subtraction line, the line becomes a division symbol; and if they are both deleted, it reverts to subtraction.

#### 3.3 Instantly declarative

The essence of an *instantly declarative* interface is that it cannot show the user something that is false, *ever*. For example, an instantly declarative calculator could never show " $3+4=15$ ." The display has to be correct without any further user action; and instantly. The benefits for the user are obvious: there is no confusion for the user, the input and output *always* correspond. The interface feels natural and immediately responsive to user input.

This approach means that "=", "Go" (and similar) buttons are redundant; they only slow the user down (sometimes users get stuck, waiting for things that will not happen until they do something which they don't know they are

supposed to do). An instantly declarative interface has to cope with partial and incomplete input and respond fully in a timely fashion. Using *equal opportunity* [3] enables us to handle incomplete input or partially complete equations. Requiring complete input can lead to a very modal user interaction. This may be suitable in some domains (e.g., with safety related issues), but it is in principle unnecessary. Conventional calculator designs never escaped this unnecessary modality of requiring complete input.

### 3.4 Output = input = everything

A declarative calculator, both from our informal and empirical evaluations, is superior. But the power of a declarative interface is only fully realised when combined with a two-way equivalence between the user's input actions and the system's output. This added to the *equal opportunity* that treats output and input equally, creates a uniquely usable interface. Users are suddenly able to solve problems, such as 'what power of 2 is 56?' (i.e.,  $2^x=56$ ) *directly* that they might have no idea of how to solve otherwise, and which would be impossible without circumlocution — and would be impossible to do correctly without prior experimentation on a calculator, for even within-brands do advanced arithmetic calculations differently!

The 'output = input' concept works smoothly with a calculator, because the output and input are the same format and can be combined in the same expression. With our calculator, a user can replace the computer output with their own, and nothing will change. This means that if the user writes the correct answer in then the calculator shows no extra work, and it means that if the user writes a wrong sum like "3+4=15" the calculator corrects it. When users use it they find that as one user put it their old calculators are "nagging and pedestrian fusspots."

### 3.5 Continuous feedback

The visibility of the system's status is provided through two kinds of feedback: *annotation* and *morphing*. These together provide clear feedback about exactly what is happening with the user's input and the calculation. Throughout the calculation the calculator morphs the input into a neatly typeset output equation. Without this linking of the output to the input the user has a jarring experience that leaves them wondering where the output came from. The morphing provides continuity between the user's input and the typeset equation that allows them to continue to edit and use it. Certainly, the animation in the morphing is visually seductive.

## 4 CONCLUSIONS

Although the new calculator furnishes a very simple user interface to a boring application (who is *really* interested in calculators?) it is very engaging — and this is true despite the calculator's prototype implementation's shortcomings, particularly its imperfect handwriting recognition.

We started with a principle-led design, but ended up with a user interface that is surprisingly effective, one that is fun and engaging to use, and one that uses and develops a new style of interaction. Whether the new style of interaction can be successfully generalised into other domains remains to be seen, but certainly the individual principles that led the design can be used to their benefit.

## ACKNOWLEDGMENTS

Will Thimbleby is supported by a Swansea University PhD scholarship. Harold Thimbleby is a Royal Society-Wolfson-Research Merit Award holder, and acknowledges this generous support.

## REFERENCES

- [1] R Bornat & H Thimbleby, (1992) "The Life and Times of Ded, Display Editor," *Cognitive Ergonomics and Human Computer Interaction*, pp225–255, J. B. Long & A. Whitefield, eds, Cambridge University Press.
- [2] P. Cairns, S. Wali & H. Thimbleby, (2004) "Evaluating a Novel Calculator Interface," *Proceedings BCS HCI Conference*, 2, edited by A. Dearden & L. Watts, Research Press International, pp9–12.
- [3] C Runciman & H Thimbleby, (1986) "Equal Opportunity Interactive Systems," *International Journal of Man-Machine Studies*, 25(4):439–451.
- [4] H Thimbleby (1996) "A New Calculator and Why it is Necessary," *Computer Journal*, 38(6):418–433.
- [5] H Thimbleby (2000), "Calculators are Needlessly Bad," *International Journal of Human-Computer Studies*, 52(6):1031–1069.
- [6] H Thimbleby (1983) "'What You See is What You Have Got'—A User-Engineering Principle for Manipulative Display?" First German ACM Conference on Software Ergonomics, *Proceedings ACM German Chapter*, 14:70–84.
- [7] W Thimbleby, (2004) "A novel pen-based calculator and its evaluation," *Proceedings Third Nordic Conference on Human-Computer Interaction*, ACM NordiCHI, pp445–448.
- [8] J. LaViola, Jr. & R. Zeleznik (2004) "MathPad2: a system for the creation and exploration of mathematical sketches" *ACM Transactions on Graphics*, 23(3):432–440.

## **I.3 Mathematical Mathematical User Interfaces**

# Mathematical Mathematical User Interfaces

Harold Thimbleby and Will Thimbleby

Department of Computer Science, University of Swansea, SWANSEA, Wales  
harold@thimbleby.net, will@thimbleby.net

**Abstract.** Taking *Mathematica* and *xThink* as representatives of the state of the art in interactive mathematics, we argue conventional mathematical user interfaces leave much to be desired, because they separate the mathematics from the context of the user interface, which remains as unmathematical as ever. We put the usability of such systems into mathematical perspective, and compare the conventional approach with a novel declarative, gesture-based approach, exemplified by *TruCalc*, a novel calculator we have developed.

## 1 Introduction

*TruCalc* is a new calculator, with a gesture-based handwriting recognition user interface. This paper reviews its design principles and relates them to the requirements of mathematical user interfaces.

## 2 The Development of Mathematical User Interfaces

For thousands of years, we've been doing maths by using pencil and paper (or equivalent: quill and scroll, stick and sand—whatever). When calculating devices were invented, this helped us do calculations faster and more reliably, but we still did maths on paper. Comparatively recently, computers were invented, and for the first time we could replace pencils with typed text and get results written down automatically, and then, later, we replaced paper with screens. Mathematics displayed on screens can be manipulated more freely than ever before, yet most calculators running on computers emulate mechanical devices.

Turing famously presented a formal analysis of what doing mathematics entailed [17]. He argued any pencil and paper workings could be reduced, without loss of generality, to changing symbols one at a time from a fixed alphabet stored on an unbounded one dimensional tape. Symbols are changed according to the current state of the device, the current symbol on the tape, and elementary rules. The Turing Machine, which can be defined rigorously (and in various equivalent forms), was a landmark of mathematics and computing. Indeed, the Church-Turing Thesis essentially claims that all forms of computing, and hence mathematics, can be 'done' by a Turing Machine in principle.

Turing introduced his machine with the following discussion:

“Computing is normally done by writing certain symbols on paper. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will

be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper.”

A. M. Turing [17]

Here, Turing’s use of the term ‘computing’ is historical; he is referring to human computation on paper.

While Turing is formally correct, good choice of notation is crucial to clear and efficient reasoning. Moreover, almost all notations (for example, subscripts) are two dimensional, as suits pencil and paper—and the human visual system. One view of the present paper is that the power—the ‘Turing equivalence’—of typical mathematical user interfaces has blinded us to the importance of notation and interactive notation properly integrated with the way the user interface works. Users put up with one-dimensional and other limitations to interaction because the deeper ideas appear sufficiently well supported. A very interesting discussion of Turing Machines and interaction is [3], but the focus of this paper now turns to the design of interactive mathematical systems.

## 2.1 Conventional Mathematical Interaction

Without loss of generality, mathematicians use pencil, paper and optionally erasers. Pencils are used to draw forms, or to cross them out. Typically, adjacent forms are related by a refinement. Harder to capture formally, the mathematician’s brain stores additional material, which is typically less organised than the representation on paper. One might argue that much of the mathematician’s work is to find a relation between what is in their head and marks on paper. This is an iterative process. Finally, the concepts and previously unstated thoughts are mapped to some representation such as  $\text{\LaTeX}$ , so that the organised and checked thoughts can be communicated effectively to other brains.

When this process is computerised, the forms are linearised into some character sequence. A string, typed onto ‘paper’ or a VDU left to right, is transformed by the computer inserting the values of designated expressions. A typical handheld calculator is an example of this style of interaction, though most only display numbers and not the operators—one of their limitations is that the user does not know whether the display is the current number being entered or a result from a previous computation.

Around the 1970s, the sequential constraint became relaxed: the underlying model remained incremental as before, but the user could ‘scroll back’ and edit any string. Now the values computed may have no relation to the preceding strings, because the user may have changed them: the old output may be incorrect relative to the current string.

More recently, from the late 1980s on, the user interface supported multiple windows, each separately scrollable and editable, each with an independent user interface much like a typographically tidied up 1970s VDU. Of course, this gives enormous flexibility for managing various objects of mathematical concern (proofs, tactics, notes...) [10], especially when supplemented with menus and keyboard commands, but the generality and power should not distract us from



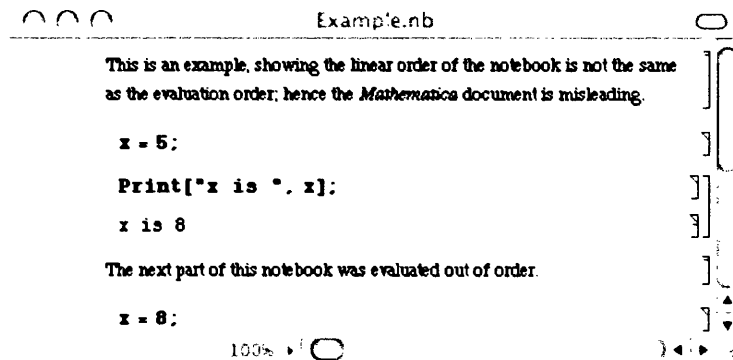


Fig. 1. Example of problematic interaction in *Mathematica*

the relation of the user interface to doing the mathematics itself. Normally we focus on the maths, and ignore the interface; it is just a tool to do the maths, not of particular mathematical interest itself.

Consider *Mathematica* [18]. A *Mathematica* notebook is a scrollable, editable document representing the string. Certain substrings in the notebook are identified, though the user can edit them at any time and in any order. A set of commands, typed or through menu selection, cause *Mathematica* to evaluate the identified substrings, and to insert the output of their evaluations. It is trivial to create *Mathematica* notebooks with confusing text like that shown Figure 1, which illustrates the inconsistency problem (is  $x$  5 or 8?) as *Mathematica* separates the order of the visible document from the historical order of editing and evaluation. In the example above, the  $x = 5$  may have been edited from an earlier  $x = 8$ ; the `Print` may have been evaluated after an assignment  $x = 8$  evaluated anywhere else in the notebook; or the `Print` may have been edited from something equivalent to `Print["x is 8"]`—and this is not an exhaustive list. In short, to use *Mathematica* a user needs to remember what sequence of actions were performed. (In fact, *Mathematica* helps somewhat as it can show when a result is possibly invalid.)

Although the presentation can be confusing, the flexibility is alluring. While the mathematician can keep the editing and dependencies clear in their head, the notebook (or some subset of it) will make sense.

*Mathematica* and many other systems add notational features so they can present results in conventional 2D notation. Instead of writing a linearised string, such as  $1/2$ , the user selects a template  $\frac{\square}{\square}$  from a palette of many 2D forms. The  $\square$  symbols can then be over-typed by 1 and 2, to achieve (in this example),  $\frac{1}{2}$ . Such mechanisms allow the entry of forms such as

$$\int_0^{\infty} \sin x^2 e^{-x} dx \quad \text{and} \quad 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}}$$

as shown with relative ease. However, a problem is that the template continues to exist even though the user cannot see it. A simple example illustrates the

problem: editing  $\frac{1}{2}$  to 12 is difficult, because the initially hidden template will reappear explicitly in intermediate steps such as  $\frac{1}{12}$  or  $\frac{1}{1}2$ .

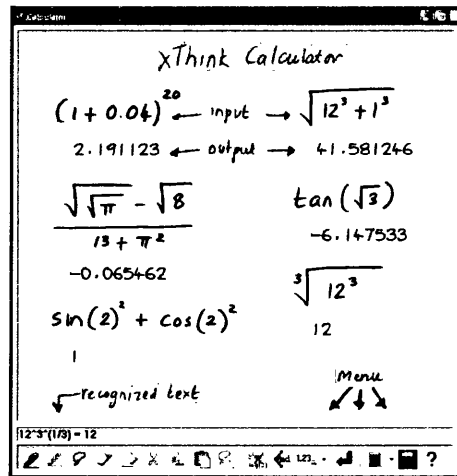
In *Mathematica* a function `TraditionalForm` achieves the inverse: presenting evaluations using standard 2D notation. While these 2D notations look attractive (and indeed are considerably clearer for complex formulae, especially for matrices, tensors and other such structures), they do not alter the semantics or basic style of interaction.

Padovani and Solmi [5] provide a good review of the interaction issues of using 2D notations, such as *Mathematica* and other systems use. They argue that 2D notation requires a model, namely the internal representation of the structure, which is not visible in the user interface. Hence, for the user to manipulate the 2D model new operations are required. The model itself is not visible, so inevitably 2D notation introduces modes and other complexities. That is, it looks good, but is hard to use. Editing operations are performed on non-linear structures (e.g., trees), but the displayed information does not uniquely identify the structure. Like the criticisms of *Mathematica* above, to use a 2D structure requires a user to remember how they built it; worse, what the user has to remember (Padovani and Solmi argue) does not correspond with the user's mental image of the mathematics being edited.

*xThink* is a different mathematical system [19], and its model is directly based on a 2D representation. *xThink* recognises the user's handwriting in standard notational format, and can compute the answer which is displayed adjacent to the hand-written sum. Provided *xThink* recognises the user's writing reliably, the internal model of the formula is exactly what the user wrote. Nothing is hidden. In this sense, *xThink* solves the problems Padovani and Solmi elaborate, though not all of the problems we attributed to *Mathematica* (as we shall see below).

A typical "page" from *xThink* is shown in Figure 2. Its advantage over *Mathematica*'s template-based approach is the ease and simplicity of entering mathematics, however its interaction style retains the problems of *Mathematica*'s—there is no guarantee the 'answers' are in fact answers to the adjacent formulae, and furthermore *xThink* has introduced new handwriting recognition problems; that is, the formula evaluated may not *ever* be one that was thought to have been written down!

*xThink* and *Mathematica* are only two examples, selected from a wide range of systems. Maple [2], for example, is closer to *Mathematica* in its computer algebra features, but closer to *xThink* in its handwriting recognition. However, Maple uses handwriting recognition to recognise isolated symbols which are written in a special writing pad—whereas *xThink* allows writing anywhere, but the writing has to be selected (by drawing a lasso around it) before it can be recognised. *xThink*, *Mathematica* and Maple are PC-based systems, and there are also many handheld mathematics systems, such as Casio's ClassPad [1], which allow pen-based input. However, rather than review individual systems, this paper now turns to principles underlying mathematical interaction.



**Fig. 2.** Example of *xThink*, showing natural handwriting notation combined with calculated output. Picture from *xThink*'s web site [19]; the original is in several colours, making the input/output distinctions clearer than can be shown in greylevels. In the picture, *xThink* has just parsed a handwritten  $\sqrt[3]{12^3}$ , shown its interpretation at the bottom of the screen (as  $12^3 \wedge (1/3) = 12$ ), and has inserted a result in a handwriting-like font below the formula.

## 2.2 Principles for Mathematical Interaction

With such a long and successful history of procedural interaction it is hard to think that it could be improved; systems like *Mathematica* are Turing Complete (upto memory limitations). Interactive mathematical systems, such as *Mathematica* and *xThink*, are clearly very powerful and have a very general user interface. The book *A = B* [6] gives some substantial examples of what can be achieved.

It is interesting to observe that the representations these mathematical system work with are *not* referentially transparent nor are they declarative. That is they only do mathematics that is 'delimited' in special ways, and the user has to 'suspend disbelief' outside of the theatre that is so delimited. As a case in point, we gave the example above of *x* not having the value it appeared to have (see Figure 1); even allowing for the semantics of assignment, there is no model like lvalues and rvalues that maintains referential transparency [9], without some subterfuge such as having a hidden subscript on all names—which, of course, must exist in the users' mind (if at all) if users are to do reliable mathematical reasoning.

Such Fregean properties as referential transparency<sup>1</sup> are key to reliable mathematical reasoning. Another is his idea of 'concept' that has no mental content, that is, a concept is not subjective. Most interactive systems *require* the user to conceptualise (i.e., make a mental model of) the interaction; they have modes, hidden state dependencies, delays, separated input and output and so on.

<sup>1</sup> Quine introduces the term referential opacity but attributes the idea to Frege [7].

It is ironic that modern mathematical systems are so flexible that they compromise the core Fregean principles—though [12] shows, under broad assumptions, any string-based (i.e., Turing equivalent) user interface interaction properties such as modelessness and undo are incompatible. Modelessness is, of course, an HCI term covering issues such as side effects, referential transparency, declarativeness, substitutivity, etc. Essentially, a purely functional interface is modeless; if one cannot have modelessness and undo (under the assumptions of [12]), any such user interface must be compromised for mathematical purposes. Such observations beg questions: is it possible to modify the style of interaction to preserve the core mathematical properties—and what would be gained by doing so?

### 3 Modern Mathematical Interaction

We will use *xThink* below to make a side by side comparison with our novel interface, *TruCalc*, to highlight the difference between a truly mathematical system and one that is not.

**Note.** *xThink* is a commercial application available from [19] (PC only), and *TruCalc* from [16] (Mac, PC, Linux).

Both our calculator and *xThink*'s calculator from first glance appear to do the same things. In fact *xThink*'s calculator seems to be more powerful, it can handle annotation, multiple sums, more complex mathematics. Yet ignoring a bullet point comparison and the superficial similarity of the two programs, they are in fact very different.

Both calculators provide a user interface based on handwriting recognition. But this is where the similarity ends!

Our calculator, *TruCalc*, was designed from generative user interface principles [12]; in contrast, *xThink* seems to merely add the idea of utilising the affordance [4] of pen and paper without escaping *Mathematica*-style problems.

To better illustrate the differences between these two superficially similar interfaces we will describe the interaction a user employs to solve a simple sum, along with the potential pitfalls.

#### 3.1 *xThink* vs. *TruCalc*

A first example problem we compare finding the value of  $(4 + 5)/3$  in *xThink* and in our calculator, *TruCalc*. In both, the user starts by writing the sum on the screen, using a pen (or using their fingers on suitable touch-sensitive screens).

- 
- 1a** In *xThink*, the user must press a button to change *xThink* into selection mode. The user can then select what they have written. They must now press another button to get the selected handwriting recognised. The handwriting is recognised and represented in a separate window, which the user must read to check the accuracy of the handwriting recognition. If the handwriting is misrecognised by *xThink* then without checking the small text at the

bottom of the screen the user can easily be fooled into thinking they have the correct answer. The text at the bottom of the screen is both small and linearised, losing the benefit of the handwritten 2D notation—for example Figure 2 shows the cube root of twelve cubed being calculated, it is printed as  $12^{\sim}3^{\sim}(1/3)=12$ .

- 1b In *TruCalc*, as the user writes, the hand-written characters and numbers are converted to typeset symbols *without any further user action*. The user feels as if they are writing in typeset characters, and confirming recognition is as natural as checking your own handwriting is legible.
- 
- 2a In *xThink*, to determine the answer, the user must now press another button to evaluate the recognised formula, and the answer is then displayed somewhere on the screen. In Figure 2 all such answers have been positioned under their respective formulae.
  - 2b In *TruCalc*, the typesetting *includes* solving the equation. In this case, the screen will show a typeset  $\frac{4+5}{3} = 3$ —the user wrote  $\frac{4+5}{3}$  and the computer inserted  $= 3$  in the correct position.
- 
- 3a In *xThink*, to determine the answer, the user's input must be syntactically complete (an expression). For example, to find the value of  $\sqrt{4}$  the user must write exactly this (and it must be recognised correctly).
  - 3b In *TruCalc*, answers are provided even with incomplete expressions, as well as with equations. For example, to find the value of  $\sqrt{4}$  the user can write  $\sqrt{\phantom{x}}$  then 4, or 4 then  $\sqrt{\phantom{x}}$ , and they can write  $=$  if they wish. In any case, the value 2 or  $=2$  is also displayed. Furthermore, if the user wrote  $\sqrt{\phantom{x}} = 2$ , then *TruCalc* would insert 4 appropriately, thus solving a type of equation where *xThink* would require the user to write  $2^2$  (which is notationally different).
- 
- 4a In *xThink*, the user's handwriting can be altered and hence make the answer (here, 3) invalid—and it will remain invalid until the handwriting is re-selected, recognised and re-evaluated (and the old answer removed). Or several answers may accumulate if the user evaluates formulae and does not remove old answers.
  - 4b In *TruCalc*, as typesetting *includes* solving the equation, the user could continue and write  $=$  or  $= 3$  themselves. In particular, if they wrote an equation, such as  $\frac{4+}{3} = 3$ , *TruCalc* would solve it, and insert (in this case) 5.
- 
- 5a *xThink* provides no other relevant features for the purposes of this paper.
  - 5b In *TruCalc*, the editing of the user's input is integrated into its evaluation. Thus the user can then continue to write over the top of this morphed equation, adding in bits that they consider are missing. For example, if the RHS 3 is changed to 30, the display would morph to  $\frac{4+86}{3} = 30$ . It is possible to edit by inserting, overwriting and by drag-and-drop to a bin to delete a selection, or to other parts of the equation to move it. In all cases, the equation *preserves* its mathematical truth, as *TruCalc* continually revises it. *TruCalc* also provides a full undo function, which animates forwards and backwards in time—also showing correct equations.
-

### 3.2 In-Place Visibility

With *TruCalc* the replacement of the user's handwriting with typeset symbols not only provides an immediately neat and tidy (and correct) equation but also provides immediate visible feedback of what was recognised. The displayed typeset equation *is* the equation that the answer is shown. This in-place visibility removes confusion and misunderstanding over what the calculator is doing, and whether it has misrecognised bad handwriting.

In our experiments with *TruCalc* [14], one of the outstanding results was that whilst users made intermediate errors, *no* user stopped on a wrong answer. We believe this was because the calculation they were performing was entirely visible and unambiguous to them in an in-place 2D notation.

Without in-place visibility, the user may be unsure which results correspond with which inputs. This compromises mathematical reliability; the user has to rely on their head knowledge.

### 3.3 No Hidden State; Modelessness

Hidden state and modes compromise mathematical reasoning. Hidden state affects how to interpret input and output; specifically, modes are hidden state (e.g., knowledge of history) in the user's head that is needed to know how to control the user interface predictably.

Typically, a system does not show what mode it is in, but the mathematical interpretation of its display depends on the user knowing some hidden state. For example, in *xThink* to erase or move parts of the equation the user has to select different tools at the bottom of the screen, then when they have finished they have to remember they are in a special mode and reselect the pen tool. The *xThink* interaction style makes this cumbersome approach unavoidable in principle. The relative meanings of displayed results obviously changes when other images are modified; simply, they may become wrong.

The *xThink* user also has to be aware that once they have finished an equation they have to do more (press several buttons, select their text) this time switching mental modes from "entering" to "getting the answer." If they don't change modes (or if they don't change through the modes appropriately, or select inaccurately), there is either a wrong result or no result for the problem.

With *TruCalc* there are *no* hidden modes or state, and no user context switching. Not only is there no menu of different tools but there is no need to switch mental modes or to pause and press an **Enter** button to make things work. This greatly simplifies the user's mental model and reduces the effort required to use the calculator. *TruCalc* does have a few modes, for example a dragging mode, but these are clearly visible and they are directly initiated and controlled by the user.

Note that in-place visibility and modelessness together give a very strong—and easy to use—interpretation of WYSIWYG (what you see is what you get), as proposed in [11].

### 3.4 Instant Declarativeness

A system may show the mathematically right answer when the user asks for it; but until they ask for computation, the mathematics is strictly incorrect (or possibly shows a representation of a meta-‘undefined’). In *TruCalc* the results are ‘instantly’ correct, with no user action required.

Declarative programming was popularised through Prolog. Essentially, the programmer writes true statements, ‘declaring’ them, and Prolog backtracks to solve the equations (sets of Horn clauses in Prolog). Prolog is thus a declarative language—though its user interface isn’t.

Likewise, *TruCalc* is declarative. The user writes equations (or partial equations, taking advantage of the automatic syntax correction), and these are declarations that *TruCalc* solves (by numerical relaxation).

In Prolog, the user has to enter a query, typically terminated by a special character. Until that character is pressed, the output (if any) is incorrect. This inconsistency within the interface is what we are used to, even to the extent of accepting the sort of inconsistencies illustrated in Figure 1. But it requires the user to remember the past; they haven’t pressed return or some other special character or menu selection yet. If they forget confusion happens.

*TruCalc* extends declarativeness to *instant declarativeness*, that is, an interface that is always true all of the time. No matter what the user writes the answer shown is *always* correct.

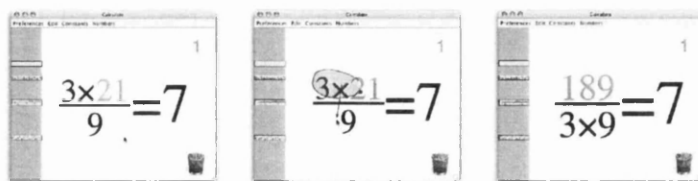
An instantly declarative interface implies that the calculator has to be showing something that is correct even if the user has not finished entering everything, or has a currently incorrect edit. Thus the calculator also has to cope intelligently with partial expressions like  $\div 3 + 2$ . In our case the calculator fills in place holders that alter the expression as little as possible. There are also problems like  $1/0$  or overflow like  $10^{10^{10}}$ —these too can be handled by correction (such as showing  $1/0$  as  $1/(0 + 1)$ ; see [13]), or by changing the algebra implemented by *TruCalc*.

This instant declarativeness removes the disparity between the input and the output, removing an enormous potential for user confusion and it also removes the need for the user remembering having to press the “equals” button (or some other change mode button) to get an answer.

The implementation of instant declarative user interfaces is only slightly more complex than conventional user interfaces; at least two threads are required, one for the user input, one for processing. Processing restarts every time the user extends or changes the input; in *TruCalc* there is a short delay, which allows the user to write an expression fluidly without visual interference from it being morphed into recognised text until they finish or pause.

### 3.5 Equal Opportunity

The power of *TruCalc*’s implementation of instant declarativeness combines powerfully with equal opportunity [8]. Unlike *xThink*, *TruCalc* does not distinguish in principle between the user’s input and its own output. Each has ‘equal opportunity’ in the equation. This makes it possible to write on both sides of an equality.



**Fig. 3.** Example of drag and drop interaction in *TruCalc*, shown as three consecutive screen-shots. Initially, the user has written  $\frac{3 \times 21}{9} = 7$ ; next, the user drags the  $3 \times$  numerator to the denominator; finally, *TruCalc* provides the correct numerator. The *only* user interaction to achieve this transformation is to draw the loop (shown in the middle figure) and drag it. Had the user had dragged the  $3 \times$  to the wastebasket, it would have been deleted, and the equation would be corrected to  $\frac{54}{9} = 7$ . (If a loop is drawn not containing anything to select, it is recognised as a zero).

The ability to change either the answer or the question lets a user solve problems simply that they would have struggled with otherwise. For example, “what power of 2 is 100” can be solved directly without logarithms. (For example, the user writes  $2 = 100$ , which is corrected to  $2 = 100 - 98$ , then writes a decimal point as the exponent of 2, which is where they want the answer.  $2 = 100 - 98$  then morphs to  $2^{6.643856} = 100$ .)

Equal opportunity is not in itself a feature that is required for a highly mathematical user interface, but it is a natural generalisation (from expressions to equations) that significantly increases the power of the user interface for the user.

### 3.6 Rearranging

In *xThink*’s calculator it is possible to delete things or move them around but it is always an awkward process involving many mode changes and it is fairly limited in what it achieves. Moreover, *any* editing in *xThink* breaks the relation between written input and calculated output, and the user has to remember to re-evaluate an edited formula. Hence, in *xThink* the ability rearrange introduces modes and hidden state.

In *TruCalc* the ability to drag and drop an arbitrary part of the equation elsewhere is synchronised by *TruCalc*’s ability to morph the result into a new typeset equation. It is therefore possible to move parts of the equation around without regard for their size or shape, and the user *always* sees a fully correct equation.

More specifically, in *xThink* drag-and-drop is achieved by choosing the selection tool, drawing around the object, then dragging, then choosing the next tool to use; however, once moved, the formula typically needs explicitly selecting, recognising, and evaluating, as further steps for the user. In *TruCalc* drag-and-drop is achieved by drawing around an object and moving it. No mode change is required, and no action needs to be taken to evaluate the new formula. Figure 3 illustrates some simple examples.



$12=1$  *TruCalc* has just recognised a handwritten 1, and shown the (at this moment) correct equation  $1 = 1$ ; the user is now writing 2 by hand.

$12^3=12$  *TruCalc* has recognised the 2; the user is writing 3 as an exponent.

$12^3=1728$  *TruCalc* has recognised the 3, and updated the RHS of the equation.

$\sqrt{12^3}=1728$  The user is writing a  $\sqrt{\phantom{x}}$  around the  $12^3$ . Of course, the user could equally have started by writing the  $\sqrt{\phantom{x}}$ , and then writing inside it.

$\sqrt[3]{12^3}=41.5$  The  $\sqrt{\phantom{x}}$  is recognised, the RHS is updated, and the user has started to write 3.

$\sqrt[3]{12^3}=12$

Fig. 4. A step-by-step, broken-down example of using *TruCalc* on the sum that *xThink* is shown solving in Figure 2, showing how a single equation changes as the user writes on it. This brief example does not show drag-and-drop, nor equational calculations. However, notice that *TruCalc* provides continual correct feedback; there are no hidden modes, no special commands—*TruCalc* just ‘goes ahead’ and provides in-place answers. The user feels as if they are writing in a formal typeface (here, Times Roman). This brief example does not show how *TruCalc* would handle solving equations, for instance if the user dragged the 12 onto the RHS. Had the user written an = themselves on the left of their formula, then the answers would have been shown on the LHS.

#### 4 A Demonstration of *TruCalc*

Because *xThink* is not highly interactive, ironically, its screen shots (such as Figure 2) make it easier to understand than screen shots of *TruCalc*! *xThink*’s screen shots show handwriting input, the recognised input (shown in the bottom pane), and the result. Figure 2 shows several such examples. It looks straight forward—except, as we showed in Section 3.1, *constructing* the interesting display of Figure 2 requires transitions between many modes, and hence possible user errors. Figure 4 shows *TruCalc* solving the problem that *xThink* is shown solving in Figure 2; however, *xThink* solves the equation in one step and requires changing modes, whereas *TruCalc* solves continually, in place, and needs no modes at all. (In this short paper we do not illustrate how *TruCalc* can solve equations more powerfully than *xThink*—by combining rearranging with equal opportunity; see [13] for examples.)

#### 5 Other Features of *TruCalc*

*TruCalc* provides other features that make it more powerful and easier to use. These features support, but are semantically unrelated to the highly interactive

way it does mathematics. Further discussion of *TruCalc*, beyond the scope of the present paper, can be found in [14] and [15].

### 5.1 Ink Editing

In *xThink*, the user writes a formula *then* asks for it to be recognised. In *TruCalc*, the formula being written is *continually* being recognised. This permits a very powerful, and natural, interaction style we call ink editing.

If the user writes ‘−’ it is recognised as a minus sign. If they write 2 above it, the minus sign becomes a division bar. If they cross it out by a vertical stroke, it becomes a + sign.<sup>2</sup> None of these natural ink editing operations makes sense in a batch recogniser.

### 5.2 Dock

*TruCalc* provides a dock, with functionality similar to the dock in Mac OS X. That is, a whole or partial equation can be dragged to the dock, and it will be stored as an item. Conversely, any item in the dock can be clicked on, and it will replace the current equation. If an item is dragged out, it ‘comes out’ as a picture representing its value. Hence an equation such as  $1 + 2 = 3$  might be dragged out of the dock and used, say, as an exponent, as in

$$2^{\boxed{1+2=3}} = 8$$

(the subequation is boxed, as it cannot be edited except by recalling it from the dock); such dock items can be used in many places in any other equation. The dock serves as a convenient declarative memory for the user.

The dock would be a very natural way to extend *TruCalc* to have variables, at least if entries in the dock could be named. Indeed, dock entries might be associated with URLs, and be able to represent internet resources—such as the current dollar/euro conversion rate, or standard numbers and equations, and so on.

### 5.3 Optionally Hidden Answers

*TruCalc* shows correct answers at all times, just as we have described it. However, for use in teaching, it is possible to hide the answer, and show an empty box. This indicates to a student that their answer is wrong or incomplete, and some correction is still required. Here is an example:

$$2 + \square = 3$$

where normally it would show  $2 + 1 = 3$ .

<sup>2</sup> The current implementation of ink editing is not complete; for example you cannot edit − to 4, or edit . to ! in the obvious ways yet.

## 5.4 Undo

*TruCalc* provides the ability to undo edits and alterations by means of a clock metaphor. A user grabs the clock hands and can ‘rewind the time,’ and as they do so the symbols and numbers animate back through time exactly as they were morphed. The morphing provides a temporal continuity between the different steps of the calculation, and it can be played backwards and forwards (i.e., undo and redo).

## 5.5 Possible Extensions to *TruCalc*

*TruCalc* can be extended in many ways. We give a few examples:

1. The dock could be on a web site, and made multiuser so several people can collaborate. The dock could also have a palette of functions (log, sin etc) that, like the current equations, could be dragged into the working equation.
2. The back-end could be replaced with (for example) the *Mathematica* kernel so it was extensible. Currently, *TruCalc* only does complex numerical arithmetic; it could provide an interface to anything *Mathematica* etc can do.
3. Unlike *xThink*, *TruCalc* currently provides no way for a user to write things that are *not* recognised; formulae cannot be annotated, arrows cannot be drawn, and so on. A teacher would probably like another colour which can be used to draw freely with but which *TruCalc* does not interpret.

There are many obvious developments: complete handwriting recognition, to extend *TruCalc* to standard function notation (such as log); restrictions for teaching purposes (*TruCalc* uses complex arithmetic); multiple equations on the screen, like *xThink*. And so on.

However, what *TruCalc* does is show how effective—both reliable and indeed enjoyable (see §6.1)—a user interface for mathematics can be when the interaction, the user interface, itself respects the principles of mathematics.

## 6 Mathematical Mathematical Interfaces Lead into HCI

HCI is the science and art of making user interfaces more effective (and enjoyable) for humans (though HCI techniques have also been used to improve user interfaces for farm animals!).

*TruCalc* allows the user to write an equation  $e$  involving complex numbers from  $\mathbb{C}$  and elementary arithmetic operators. *TruCalc* has no variable names, but uses slots; thus, in conventional terms, the equations can contain variables without repetition—future versions of *TruCalc* may include variable names as they are of course useful for many purposes, not least in providing mnemonics for the slots as currently used.

The variety of solutions  $S$  of  $e$  is intended to be  $S(e, \mathbb{C})$ , except the current version implements  $\mathbb{C}$  by  $\mathbb{C}_J$ , the obvious approximate representation of  $\mathbb{C}$  using pairs of Java double precision floating point numbers.

With these clarifications, we can express some important HCI issues:

1. What should *TruCalc* do when  $S(e, \mathbb{C}_J)$  does not determine a unique solution? Currently *TruCalc* uses heuristics to try to find solutions that are principal values, identities of operators, and so on. For example  $\times = 10$  will be solved by  $10 \times 1 = 10$ , using the right identity of  $\times$ . On the other hand,  $10^{\frac{1}{2}} \times 10^{\frac{1}{2}} = 10$  has no solution as currently implemented, because *TruCalc* effectively tries to solve  $1/x = 0$ .
2. What should *TruCalc* do when  $S(e, \mathbb{C}_J) = \emptyset$ ? *TruCalc*'s solution is to show ? symbols (or  $?+?i$ ); however, an earlier version [13] modified the equation so that at least one solution could be found. Neither solution, we feel, is entirely satisfactory, since  $S(e, \mathbb{C}_J) = \emptyset$  can occur as a transient step in entering a solvable equation—for example, to enter  $1/0.1$  either requires contortions or the intermediate step  $1/0$ .
3. What should *TruCalc* do when there is a *humanly*-obvious algebraic solution, but  $S(e, \mathbb{C}_J) = \emptyset$ ? For example, the very easily entered LHS

$$2^{2^{2^2}} = ?+?i$$

fails because it is a 19,729 digit decimal number, which is in  $\mathbb{C}$  but not in  $\mathbb{C}_J$ —but the equation could be solved as

$$2^{2^{2^2}} = 2^{65536}$$

or in many other equivalent symbolic ways. Which is best? Should the user have choices, and if so, how? A symbolic approach would also be a good way to solve equations the user enters containing  $1/0$  terms.

4. Can users choose  $S(e, \mathbb{R})$ ,  $S(e, \mathbb{Q})$ ,  $S(e, \mathbb{Z})$ ,  $S(e, \mathbb{N})$ , for instance for elementary teaching? What about  $S(e, \mathbb{Z}_{12})$  for clock numbers, or  $S(e, F_p)$ , and other interesting domains, say predicate logic or even chess?
5. Improving the handwriting recognition would allow the solution of larger classes of equations, for instance that include transcendental functions.
6. *TruCalc* uses  $=$  as an operator over  $\mathbb{C}_J$ , not  $\mathbb{C}$ . This can result in (apparently) peculiar results such as the following:<sup>3</sup>

$$\begin{aligned}\pi &= 335/113 \\ \pi &= 3.142 \\ 3.142 &= 1571/500 \\ \pi &= 3.142 - 4.073 \times 10^{-4}\end{aligned}$$

Perhaps *TruCalc* should use an operator  $\simeq$  when the equality is approximate? (Although results that are approximate in  $\mathbb{C}_J$  may be exact in  $\mathbb{C}$ !)

<sup>3</sup> The last example shows  $4.073 \times 10^{-4}$  which in an earlier version would have been presented in the standard Java format as  $4.073E - 4$ , a 'buggy' notation, because a user could not enter  $E$  themselves, so it failed equal opportunity. Here, equal opportunity is seen to be a *generative* design principle: given the existing features, it suggested improvements.

7. *TruCalc* could explicitly show, where it is the case, that numbers are approximate. For example,  $\pi =_{[3]} 3.142$  could be the notation to indicate the equality is correct to three decimal places. If the user changed the subscript 3, they would be changing the precision of the displayed number. Chaitin however suggested that it would be more in keeping with the direct manipulation style of *TruCalc* to allow the user to drag the righthand extension of decimals: so if the user drags the ‘...’ to the right in the equation  $\pi = 3.142\dots$  it could become  $\pi = 3.141592653589793\dots$ ; and dragging the ‘...’ left would put it back to  $\pi = 3.1\dots$ , for example.

In summary, an interesting part of the ‘HCI of *TruCalc*’ can be expressed as the relation between  $S(e, \mathbb{C}_J)$ , the solutions the implementation provides for an equation  $e$ , and  $S(e, \mathbb{H})$ , what the user expects.

## 6.1 Enjoyment

Finally, it surprised us that *TruCalc* was fun to use—we had developed it from principles and had not anticipated the strong feeling of engagement it supports. It integrates body movement, handwriting, and instant *satisfaction*, that children and post-doc mathematicians find exciting. Elsewhere we have reported on our usability surveys, a topic that is beyond the scope of this paper [14]. More recently *TruCalc* was exhibited at the Royal Society Summer Science Exhibition, where it was used by thousands of visitors, children, parents, teachers, to math postdocs. An exit survey was completed by 420 participants (and we insisted that anybody who took a survey form completed it, to avoid under-reporting of negative results) had 90% **liked** or **really liked** *TruCalc*, and nobody (0%) **disliked** it.

## 7 Conclusions

Current leading mathematical systems are capable of a remarkable range of mathematics. With *Mathematica*, a market leading example of an interactive computer algebra system, we are able to solve problems we could not do without it. It is easy to confuse these mathematical capabilities with usability. So much power seems harnessed that the power seems usable.

This ‘power leverage’ blinds us to the fundamental non-mathematical nature of these user interfaces. Often clear mathematical principles like referential transparency and declarativeness are lost in modes, history dependence, context sensitivity, and so on. The failure of these principles in conventional mathematical user interfaces undermines our ability to reason reliably or mathematically.

*xThink* makes use of the affordance of pen and paper to create an interface that solves partially some of the interface issues. But it still ignores basic mathematical principles when applied to interaction. It gains the affordance of paper, at the expense of introducing evaluation modes (and uncertainty in the handwriting recognition).

We have shown in *TruCalc* that it is possible to create an interface that supports basic principles throughout the user interface; it has no hidden state, is modeless, instantly declarative, and so on—or in Frege *et al.*'s metamathematical terms, substitutional, referentially transparent, and so on. Adhering closely to these mathematical principles do not compromise the power of *TruCalc*; it is in principle as powerful mathematically as *xThink* and other conventional systems (though obviously the two systems vary in detail, such as in the choice of built-in functions they support)). Further, we have shown that by supporting these principles that the calculator is easier, more enjoyable, fun and usable—a paradigm shift in usability.

**Acknowledgements.** Harold Thimbleby was supported by a Royal Society-Wolfson Research Merit Award, and Will Thimbleby by a Swansea University studentship. The design of *TruCalc* is covered by patents. Paul Cairns, Greg Chaitin, James McKinna, John Tucker and very many anonymous participants in demonstrations and lectures gave us very useful comments. The Exhibition of *TruCalc* at the Summer Science Exhibition at the Royal Society was funded by EPSRC under grant EP/D029821/1, and Gresham College.

This paper was originally an invited talk at the Mathematical User-Interfaces Workshop 2006 (<http://www.activemath.org/~paul/MathUI06>), but did not appear in the proceedings.

## References

1. Casio, Casio ClassPad 300 Resource Center (2006), <http://www.classpad.org>
2. Garvan, F.: The MAPLE Book. CRC Press, Boca Raton (2001)
3. Goldin, D.Q., Keil, D.: Persistent Turing Machines as a Model of Interactive Computation. *Foundations of Information and Knowledge Systems*, 116–135 (2000)
4. Norman, D.A.: Affordances, Conventions and Design. *Interactions* 6(3), 38–43 (1999)
5. Padovani, L., Solmi, R.: An Investigation on the Dynamics of Direct-Manipulation Editors for Mathematics. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 302–316. Springer, Heidelberg (2004)
6. Petkowšek, M., Wilf, H.S., Zeilberger, D.:  $A = B$ . A K Peters (1996)
7. Quine, W.V.O.: Word and Object. MIT Press, Cambridge (1960)
8. Runciman, C., Thimbleby, H.: Equal opportunity interactive systems. *Int. J. Man-Mach. Stud.* 25(4), 439–451 (1986)
9. Tennent, R.D.: Principles of Programming Languages. Prentice-Hall, Englewood Cliffs (1981)
10. Théry, L., Bertot, Y., Kahn, G.: Real Theorem Provers Deserve Real User-Interfaces. In: Proc. Fifth ACM Symposium on Software Development Environments, pp. 120–129 (1992)
11. Thimbleby, H.: What You See is What You Have Got—A User-Engineering Principle for Manipulative Display? First German ACM Conference on Software Ergonomics. In: Proc. ACM German Chapter, vol. 14, pp. 70–84 (1983)
12. Thimbleby, H.: User Interface Design. Addison-Wesley, Reading (1990)
13. Thimbleby, H.: A New Calculator and Why it is Necessary. *Computer Journal* 38(6), 418–433 (1996)

14. Thimbleby, W.: A Novel Pen-based Calculator and Its Evaluation. In: Proc. ACM NordiCHI 2004, pp. 445–448 (2004)
15. Thimbleby, W., Thimbleby, H.: A Novel Gesture-Based Calculator and Its Design Principles. In: Proc. BCS HCI Conference, vol. 2, pp. 27–32 (2005)
16. Thimbleby, W., Thimbleby, H.: TruCalc (2006), <http://www.cs.swan.ac.uk/calculators>  
<http://www.cs.swan.ac.uk/calculators>
17. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. In: Proc. London Mathematical Society, Series 2, 42, 230–265 (1936/7) (corrected Series 2, 43, 544–546 (1937))
18. Wolfram, S.: The Mathematica Book, 4th edn., Cambridge (1999)
19. xThink, xThink Calculator (2006), <http://www.xThink.com/Calculator.html>  
<http://www.xThink.com/Calculator.html>

## **I.4 Internalist and Externalist HCI**



# Internalist and Externalist HCI

Harold Thimbleby  
Swansea University

Wales  
SA2 8PP  
+44 1792295393

harold@thimbleby.net

Will Thimbleby  
Swansea University

Wales  
SA2 8PP  
+44 1792295393

will@thimbleby.net

## ABSTRACT

The history of technology, as a discipline, supports alternate points of view termed *internalist* and *externalist*, which terms highlight an approximately similar division in points of view within HCI. Conventional HCI is externalist, rightly concerned with human-centered issues; but externalism risks ignoring important internalist issues. A successful human-computer system is better if it is successful from *both* perspectives.

This discussion paper argues that the externalist view, while necessary and immensely useful, is not sufficient—and in the worst case, risks eclipsing innovation from internalist quarters.

## 1. INTRODUCTION

David Nye's review of the history of technology [14] uses the clear terms *internalist* and *externalist*, applying them to styles of historical analysis.

Why did the internal combustion engine triumph over the alternatives, horse, steam and electric? An internalist might emphasize the power-to-weight ratio of the internal combustion engine; an externalist might emphasize the lower cost of the Ford Model T and the dramatic impact cost had on a growing market. An internalist, then, considers the technology as such.

- **Externalism** is focused on the world external to the user interface: *human*-interaction and e.g., observation, evaluation, cognition, etc.
- **Internalism** is focused on the world internal to the user interface: *computer* interaction and e.g., logic, engineering, computation, etc.

An example illustrating human-computer interaction issues is Tracy Kidder's classic *The Soul of a New Machine* [10]. The book traces the development of a computer, the Datta General Eclipse MV/8000, all the technical issues, right up to the point that the finished product is brought to market. Then the book ends, just when the external world of the computer and its possible use starts to get interesting. The book takes an internalist view.

Of course both views are needed in a balanced discussion, and indeed Nye provides a masterful analysis. We believe Nye's internalist/externalist terms from the history of technology have

value in distinguishing major styles in the way HCI is viewed, presented and undertaken.

Clayton Lewis proposed a similar, but, psychological distinction for HCI, that of *inner* and *outer HCI* [13]. Here, inner and outer refer to cognitive processes and human behavior respectively. Lewis emphasizes the potentially fruitful interplay of inner and outer HCI. Curiously, while his the terms "inner" and "outer" might at first seem to cover everything, Lewis *excludes* the computer (or other interactive system)—he simply does not mention it in his conception of HCI! It is as if the interactive system is a given, taken for granted, rather than a legitimate object of study in its own right.

Similarly in the "Kittle House Manifesto" [3] Carroll suggests that academic psychology has had no impact on interactive design practice, and that major innovations in practice (e.g., Sketchpad, an innovative graphics program) have made no explicit use of psychology. He bemoans the fact that HCI does not use science, or that if it does the relation is haphazard. Yet, curiously, he overlooks that computer science is science too, and in fact underlies the major contributions he describes as driving innovation. While it seems to us quite right to try to promote psychological science and explore why it is in some sense under-rated or used haphazardly, it seems counter-productive to the wider purpose of HCI to overlook computational science. Carroll's more recent collection [4] sees HCI as something computer scientists need to be taught, as something quite other than computer science, rather than something that can draw on computer science *as well as* human sciences.

This externalist emphasis of the HCI field is routinely found in the standard HCI textbooks, of which most take externalist points of view—indeed, [5] suggests that teaching HCI should cover the computer science which standard HCI textbooks omit.

Barnard, May, Duke and Duce remind us of "syndesis," binding together systems that contain interacting subsystems, such as people and computers. They introduce the terms "Type 1 theory" and "Type 2 theory," referring to approaches that go *deeper* or that go *across* interaction respectively. They warn that we are not very good at establishing Type 2 connections, and this weakness may lead to "the fragmentation and demise of HCI as a coherent science" [1].

It seems that HCI needs terminology to discuss these issues. Our internalist/externalist distinction is analogous to the Lewis inner/outer HCI distinctions, but from the point of view of the computer rather than the human. Without repeating Lewis's arguments here, we too see the great potential of fruitful interplay between internalist and externalist perspectives.

Just as a brain-computer interaction (BCI) researcher would certainly wish to go deeper into the "inner HCI" than Lewis does, so also our "internalist" perspective has a rich internal

© Harold Thimbleby, Will Thimbleby, 2007

Published by the British Computer Society

Volume 2 Proceedings of the 21st BCS HCI Group  
Conference

HCI 2007, 3-7 September 2007, Lancaster University, UK  
Devina Ramduny-Ellis & Dorothy Rachovides (Editors)

structure—it isn't just "the computer" set against the wide range of standard HCI disciplines, anthropology, psychology, social science, economics, marketing, design; the internalist sees algorithms, complexity, information theory, proof, requirements, hardware, graphics, databases, and so forth ... a rich science contributing to HCI.

## 1.1 The Authors' Perspective

Both authors of this paper have an internalist background, and it is unashamedly from this perspective that this paper has been written. The paper has a twofold purpose: to name and introduce a useful distinction for HCI, and to stimulate debate on the balance—or the lack of balance—in HCI as practiced, and hence stimulate thinking on strategies for doing better.

We believe the internalist/externalist distinction allows a constructive discussion about the methodologies of HCI, without diminishing either internalist or externalist points of view. By naming the distinction, we suggest that there are different *and valid* views about how HCI, and particularly HCI research, can and should be done. Nevertheless, we believe internalist HCI tends to be under-valued by the more dominant externalist point of view, and this paper therefore makes an enthusiastic case for internalism.

HCI could not exist without programming computers, which is an internalist perspective, and also HCI could not exist without the human context and study, which is an externalist perspective. Singly, internalist and externalist perspectives are monocular and lack depth and perspective. Both are needed.

## 2. HOW WE GOT HERE

The HCI community's traditional emphasis of externalist perspectives to some extent eclipses internalist perspectives. Historically, existing externalist methodologies were ready when they were needed: there was and still is a very substantial resource of experimental psychology that was applied and works to a high standard. In contrast, it might be said that most early internalists did not know what they were doing; see below when we comment on the Therac-25.

A second, crucial, reason for the current emphasis on externalist methods in HCI is that external experimental methods can be used independently of the specifics of internalist details. Every HCI system has very different internals, and requires investment in specific programming and design; in contrast, the externalist methods (e.g., cognitive walkthrough, think aloud, eye tracking) work on all systems. Experimental designs, statistical methods and so on, can be applied to a word processor or to a graphics package with little modification. In contrast, a new contribution to HCI by an internalist might take years of work that has no other application. It is noteworthy that most externalist studies of programming in HCI design use trivial programs, because programming real user interfaces is too slow. Inevitably, few internalists contribute to mainstream HCI.

Perhaps the HCI community has changed too. As fewer internalists contribute at the same rate as externalists, the peer community becomes dominated by externalist values. If an internalist submitted a result to a conference or journal now, most referees calling themselves members of the HCI community would be externalists.

*ACM CHI*, the major international HCI conference, is primarily externalist. In contrast one of the major internalist conferences, *DSWIS* (Design, Specification and Verification of Interactive Systems) has only a hundredth of the participants. This reflects

a difference in the sizes of the communities. Thus, internalists face higher hurdles to participate in the development of the field. Then, as the externalists operate in a community dominated by externalists, it appears reasonable to *require* externalist criteria for contributing to that community: possibly even a hegemony—being defined as the emphasis of cultural beliefs, values, and practices to the dismissal and over-looking of others.

## 3. SAMPLE SYSTEMS

### 3.1 Therac-25

Horrible stories of bad HCI abound. The Therac-25 was a medical device that killed patients as a result of "operator" error (actually system design error). It is primarily an example of inadequate internalist HCI, an argument for better internalist HCI rather than fixing design problems with externalist HCI. Bad programming killed people.

Although the Therac-25 story is an extreme example, the case illustrates how important it is for user-centered design to react against sloppy programming practices—this paper is not arguing internalism is a panacea! Given that many programmers are not computer scientists, UCD *is necessary* to improve things.

One could argue that iterative design gained prominence to compensate for the difficulty of writing good software, particularly given the typical programmer skills available to industry.

### 3.2 Calculators

By considering logic programming, Runciman and HThimbleby introduced an analytic concept, *equal opportunity* [15]. HThimbleby used equal opportunity to constrain the design of a new user interface, choosing a calculator, as this is a well-researched artifact. Background research revealed how conventional calculators were badly designed, an internalist criticism of their poor technology [16]. Somehow this critical observation had escaped externalist research on calculator user interfaces.

We question the point of externalist research when it ignores the *intrinsic failure* of the technology; what point is iterative design or working with users when the conceptual problems of the user interface are so hard, complex and broken? HThimbleby made a technically improved calculator available to the community in 1986. However, it was not till 2004 that it had any externalist evaluation [2]. More recently, WThimbleby generalised the calculator, and made its user interface recognize handwriting [17,18,19]. This calculator has had a modest externalist evaluation [17].

The new calculator was developed entirely by internalist considerations. Specifically, it should do mathematics properly [19]. Few externalist considerations drove its design, yet it is very successful. The calculator was exhibited at Royal Society Summer Science Exhibition, 2005; at the exhibition, several thousand people used it. 90% of respondents said they really liked it or loved it. But despite the unusually large scale of the survey and feedback we gained no new ideas from users that would contribute to iterative design improvements.

Some feedback from users at the exhibition is listed below:

- "It visualizes the internal workings of abstract calculations, fun, as it is wonderful! Fun! Engaging and importantly visible!"—University Professor
- "Calculators seem clumsy and hard to use—the new method is genius!—when can I buy one in the shops (If I

had had one I would have done A level maths)”—A-Level Student

- “Engagement, excitement, interactivity, seamless, more visually appealing and easier to use!”—Teacher
- “I’ve never seen anything that’s brought a smile to my face while doing addition, but this has. For that reason alone, I want one!”—Artist

The point we would like to make is that an internalist design program has produced a good user interface, recognized as such by users. Yet by conventional externalist HCI criteria, the work would not be acceptable for publication.

### 3.3 Graphics Programs

The calculator is an example of an internalist HCI research program, spanning twenty years before it resulted in a user interface that attracted attention. In contrast WThimbleby conceived, designed and built a vector graphics editor within two years, as a purely internalist project.

The resulting program, Lineform, was fully formed on its initial release. No early focus on users, no empirical design, no iterative design [7] informed its development—though of course computer science and HCI principles did inform and direct its development.

The quality of the design was recognized by the award to WThimbleby of the 2005 Apple Student Design Award. Arguably, this shows the user interface design was better than of thousands of others (i.e., the number of competitors)—which, had they been realistically entered into the review, should have been excellent programs in their own right.

Lineform is sold by Freeverse Software and has been commercially successful. The program has been reviewed in commercial magazines and web sites. Its reception has been uniformly favorable.

Below are some sample quotes from reviews. They are included to support the claim that the HCI in Lineform is successful, regardless of its lack of externalist methodology. Like the facts we presented about the calculator, the evidence supports our view that HCI contributions can be good despite the lack of externalist practices.

- “Lineform from Freeverse Software claims to be the solution for modern drawing and illustration. It is. Winner of a 2006 Apple Design Award, Lineform is not only easy to use, but the interface design makes the application so intuitive, Mac users need no explanation to start illustrating.”—*CreativeMac* (Feb 2007)
- “It’s not often that you find a product you literally have to gush over ... but Lineform, for me at least, is that product.”—*AppleGazette* (Jan 2007)
- “Lineform has two other selling points. First, its speed: the program launches in a couple of seconds and shames Illustrator throughout in its responsiveness. Second, its ease of use. The simple interface alone makes it easier to find things.”—*MacUser* (Issue 22 Volume 22)

An internalist design program produced a very good user interface, recognized as excellent by the market and critical reviewers. Yet by conventional externalist HCI criteria, the work would not be acceptable for publication.

### 3.4 Google

On any measure Google is an extremely successful user interface, with a value to users that exceeds most conventional user interfaces studied in HCI. Google is in fact just a text field

with a substantial algorithm behind it [12]: its user interface is successful because it has a good internalist design. First, the internalist algorithm *then* the user interface. *Once* Google works it *then* makes sense to evaluate it and refine it from an externalist point of view: what services do users want given that Google works, and how can they be made better? However, the original, key HCI innovation was internalist.

Few of the services Google now offers would have made any sense to users or anyone else until after the basic algorithm worked, and had been demonstrated working well. Although externalism is now essential to Google, it was not how it started.

## 4. SAMPLE ISSUES

### 4.1 Anecdotes

If Jo is using a system, and this is reported in a research contribution, then an externalist wishes to know in what way Jo is typical of the population and to what extent, if at all, the particular interaction is typical. Jo may be idiosyncratic; the experimenter may have misdirected Jo. If we wish, ultimately, to design better interfaces for anybody other than Jo, we need reliable, generalizable knowledge. Statistics is a good way to characterize reliable generalization, and a one-off experiment with a unique individual would be hard-pressed to be reliable.

From an internalist perspective things look very different. Internal arguments are independent of the user. For example, computability could show that certain tasks are impossible. Not just for Jo, but for *anybody*—impossible for the whole human population, martians, dogs and bacteria. One hardly needs to recruit conventional experimental methods to make such claims reliable. This is not an anecdotal claim, but an analytic claim.

The confusion of these two methodologies undermines communication. It is our experience that internalist papers submitted to journals and conferences have been rejected because the referees have interpreted our analytic descriptions as “anecdotal.”

The desire that contributions to HCI must include sufficient (and valid) externalist content before they are acceptable, increases the burden on the internalist researcher. Few researchers are able to span the internal/external bridge; different skills, different theory, different methods are required. Moreover, in the way of things, externalist work can only follow after internal work—or simulate it (e.g., with paper prototyping, which has no internalist content). Perhaps this is *the* gulf of HCI? An internalist has to do twice as much work?

### 4.2 Reproducibility

The systems mentioned in this paper are fully working systems and can be downloaded by interested researchers ([www.freeverse.com/lineform](http://www.freeverse.com/lineform) for the graphics program, [www.cs.swansea.ac.uk/calculators](http://www.cs.swansea.ac.uk/calculators) for the calculator, and [labs.google.com](http://labs.google.com) for an API). From an internalist perspective, the research these systems embody is reproducible. That is, the claims we make about the quality and design can readily be checked by any interested researchers; because the claims are user independent.

From the perspective of the present paper, of emphasizing internalist HCI, it seems a great advantage that exactly what we have contributed—the underlying science, the programs, and so forth—are completely available to any researchers who wish to build on or critique our work. This level of reproducibility is very rarely the case with externalist HCI research.

### 4.3 Opposition or complementarity?

At the BCS HCI 1995 conference, what we would now call an internalist/externalist debate was presented by an externalist in a keynote, metaphorically, as an actual war: “Which trench are you shooting from?” [6], illustrated with pictures of carnage. Another keynote at the same conference [8] suggested that “in a nutshell ... what I see is a need to get away from the computer at centre stage, and a need for methods of description that make themselves useful ...” If it’s a war, consider [21], which starts off, “If you want to make software developers squirm...” and sets out to create the impression that developers don’t know what they are doing. Some don’t, no doubt, but most have a hard enough job getting systems to work at all, and they should not be blamed for problems that arise through poor management expectations and requirements that *nobody* understood until their systems were working.

Landauer’s *The Trouble with Computers* [11] blames programmers for being “arrogant” (p173)—not designing for users, testing, evaluating, and so on. Programmers have “fantasies” he says. Yet he also mentions that Stu Card “a leading expert in HCI” was “confident” that a new word processor would be “vastly” better—but was proved wrong. Thus he makes rhetorical distinctions whose effects are to discredit the internalist perspective in HCI: internalists are “arrogant” whereas equally wrong externalists are “leading.”

We surely need more balanced views, particular as both internalist and externalist share the same goals for the user. A first step in being more balanced is to name the imbalance.

Grudin, one-time editor of the *ACM Transactions on Computer-Human Interaction*, presented a mature view of the diversity of the HCI community [9], based on his experience as editor and final arbiter between conflicting referee and author points of view. A non-partisan view is [20], which argues how easy it is for differences to escalate to unconstructive conflict.

## 5. CONCLUSIONS

This paper has proposed a distinction between externalist and internalist approaches to HCI. The distinction helps clarify the nature of HCI research and practice, as well as preferred approaches within the HCI research community.

This paper described a selection of very different products of internalist HCI. None have been developed through or supported research that would have met conventional externalist HCI criteria, indeed none followed any recommended externalist HCI development cycles—yet all are successful. Of course the systems beg a wide range of externalist questions, but the fact that one can now do externalist work does not mean it was necessary to do it for the overall work to form a valid contribution to HCI.

Our purpose is not to dismiss externalist approaches, but to recognize that an internalist approach to HCI can be very effective and lead to good user interface design. Internalist design and research can be valid without any externalist evaluation.

Given that the computer science community argues that design should start with a mathematically rigorous specification, and then refine to implementation—almost the opposite of the externalist HCI view of design—there are new questions to be asked. Can internalist approaches lead to quality HCI, and if so, to what extent and under what assumptions? This paper has shown that internalist HCI can. We need to see more internally-driven HCI, and we need to explore when and why it is successful.

## 6. REFERENCES

- [1] Barnard, P., May, J., Duke, D. & Duce, D., systems, Interactions, and Macrotheory, *ACM Transactions on Computer-Human Interaction*, 7(2):222–262, 2000.
- [2] Cairns, P., Thimbleby, H. & Wali, S., Evaluating a Novel Calculator Interface, *Proceedings BCS HCI Conference*, 2:9–12, 2004.
- [3] Carroll, J. M., Introduction: The Kittle House Manifesto, *Designing Interaction*, J. M. Carroll, ed., pp1–16, Cambridge University Press, 1991.
- [4] Carroll, J. M., ed., *HCI Models Theories and Frameworks*, Morgan Kaufmann, 2003.
- [5] Cockburn, A. & Bell, T., Extending HCI in the Computer Science Curriculum, ACM International Conference Proceeding Series, 3, *Proceedings of the 3rd Australasian conference on Computer Science Education*, 113–120, 1998.
- [6] Gasen, J. B., Support for HCI Educators: A View from the Trenches, *Proceedings BCS HCI Conference*, 21–36, 1995.
- [7] Gould, J. D. & Lewis, C., Designing for usability: key principles and what designers think, *Communications of the ACM*, 28(3):300–311, 1985.
- [8] Green, T. R. G., Looking Through HCI, *Proceedings BCS HCI Conference*, 21–36, 1995.
- [9] Grudin, J., “Crossing the Divide,” *ACM Transactions on Computer-Human Interaction*, 11(1):1–25, 2004.
- [10] Kidder, T., *The Soul Of A New Machine*, Back Bay Books, 2000.
- [11] Landauer, T., *The Trouble with Computers*, MIT Press, 1995.
- [12] Langville, A. N., Meyer, C. D., *Google’s PageRank and Beyond*, Princeton, 2006.
- [13] Lewis, C., Inner and Outer Theory in HCI, in *Designing Interaction*, J. M. Carroll, ed., pp154–161, Cambridge University Press, 1991.
- [14] Nye, D. E., *Technology Matters*, MIT Press, 2006.
- [15] Thimbleby, H. & Runciman, C., Equal Opportunity Interactive Systems, *International Journal of Man-Machine Studies*, 25(4):439–451, 1986.
- [16] Thimbleby, H. Calculators are Needlessly Bad, *International Journal of Human-Computer Studies*, 52(6):1031–1069, 2000.
- [17] Thimbleby, W., A novel pen-based calculator and its evaluation, *Proceedings ACM Nordic Conference on Human-Computer interaction*, 445–448, 2004.
- [18] Thimbleby, W. & Thimbleby, H., A Novel Gesture-Based Calculator and Its Design Principles, *Proceedings BCS HCI Conference*, 2:27–32, 2005.
- [19] Thimbleby, H. & Thimbleby, W., Mathematical Mathematical User Interfaces, *DSVIS 2007*, in press.
- [20] Thimbleby, H., Supporting Diverse HCI Research, *Proceedings BCS HCI Conference*, 2:125–128, 2004.
- [21] Udell, J., Capturing user experience closes the feedback loop, *InfoWorld*, [www.infoworld.com/article/04/06/04/23FEuser\\_1.html](http://www.infoworld.com/article/04/06/04/23FEuser_1.html) 2004.

*Acknowledgements.* Thanks to Ann Blandford, Richard Harper and Matt Jones.