



**University of
Sunderland**

Richardson, Andrew Grant (2010) Truth to Material: Moving from Software to Programming Code as a New Material for Digital Design Practice. Doctoral thesis, University of Sunderland.

Downloaded from: <http://sure.sunderland.ac.uk/3311/>

Usage guidelines

Please refer to the usage guidelines at <http://sure.sunderland.ac.uk/policies.html> or alternatively contact sure@sunderland.ac.uk.

Truth to Material: Moving from Software to
Programming Code as a New Material for
Digital Design Practice.

Andrew Grant Richardson

This thesis is submitted in partial fulfilment of the
requirements of the University of Sunderland for the
degree of Doctor of Philosophy.

December 2010

Abstract

This practice-led research project investigates the key characteristics of the use and process of programming code when applied to a creative design environment. The research is motivated by personal practice and a desire to move beyond the boundaries of software, and is set against a contemporary background of designers exploring code as a key part of their creative work.

The initial contextual study considers design practice in the context of contemporary digital technology, and identifies computational design as a distinct area, apart from software-centred design. Although not a formal term or grouping, the thesis highlights 'computational design' as an area of practice which has emerged out of dissatisfaction with the 'limitations' of software tools. The research establishes links between a range of contemporary design practitioners, whose work is motivated by a desire to understand and engage directly with the process and the 'material' of the computational environment. Using the Arts and Crafts movement as a case study, the contextual review discusses the ethos, process and material of software-centred and computational design alongside those of traditional design values. The research identifies the process and usage of computation as a distinct area of study for creative design which applies a traditional concern for the material and process of 'making' within the immaterial environment of the digital arena.

The identification of computation as a type of raw 'material' for creative practice provides the focus for the rest of the research. Based on the findings of the contextual review, the practice explores the detail of the process of 'making' using code, by creating two major pieces of computationally generated work, based on the botanical, decorative aesthetic of William Morris wallpaper prints. Each key stage of the work is outlined using the headings 'code', 'visuals' and 'process', providing a

detailed account of the developing process and relationship between the designer and the computational material. The study reveals that key to the use of computation is an understanding and development of structural and visual flexibility, which is inbuilt into the architecture of the work as part of the design process. The research identifies three core phases, or 'layers' within the process: 'concept', 'data structure' and 'data detail', each of which contribute important elements to the flexibility and fluidity of the structure and visuals. The research adds to the understanding of the process and practice of computational work within a creative context, increasing knowledge regarding the use and application of the formal elements of code within a creative design workflow.

Acknowledgments

I would like to thank my supervisors Professor Beryl Graham, Professor John Tait and Neil Ewins for their ongoing advice and support.

I would also like to thank the Arts Design and Media faculty at the University of Sunderland for their financial support. I would also like to express my gratitude to the staff in the Design department at the University of Sunderland, in particular Shirley Wheeler and Gurpreet Singh for actively supporting me through this process.

Final thanks go to my family, especially my wife and son, for the enormous amount of patience, forbearance and encouragement shown to me every step of the way.

TABLE OF CONTENTS

1	Introduction	15
1.1	Overview	15
1.1.1	<i>Research Question</i>	15
1.1.2	<i>Aims and Objectives</i>	15
1.1.3	<i>Background</i>	17
1.1.4	<i>Why is the Research Relevant?</i>	17
1.2	Methodology	19
1.2.1	<i>Practice-based Research</i>	19
1.2.2	<i>Features of this Methodology</i>	19
1.2.3	<i>Contextual Review</i>	20
1.2.4	<i>Practice</i>	21
1.3	Definitions and Scope	23
1.3.1	<i>Scope</i>	23
1.3.2	<i>Defining Terms</i>	29
1.4	Dissertation Outline	31
2	Contextual Review	33
2.1	Software-centred, Digital Design	36
2.1.1	<i>Introduction</i>	36
2.1.2	<i>Ethos</i>	37
2.1.3	<i>Materials and Technologies</i>	38
2.1.4	<i>Process and Skill</i>	40
2.1.5	<i>Object and Artefact</i>	43
2.1.6	<i>Summary</i>	46
2.2	Computational Material: Computational Design	46
2.2.1	<i>Ethos</i>	49
2.2.2	<i>Materials and Technologies</i>	54
2.2.3	<i>Process and Skill</i>	59
2.2.4	<i>Object and Artefacts</i>	63
2.2.5	<i>Summary</i>	66
2.3	Traditional Material: The Arts and Crafts Movement	67
2.3.1	<i>Ethos</i>	69
2.3.2	<i>Material and Technologies</i>	70
2.3.3	<i>Process and Skill</i>	72
2.3.4	<i>Object and Artefact</i>	75
2.3.5	<i>Summary</i>	76
2.4	Summary of Contextual Study	77
2.4.1	<i>Computation as Craft?</i>	80
2.4.2	<i>Contextual Review in Relation to Practical Project</i>	84
3	Colorcalm Project	88
3.1	Initial Line Drawings	91
3.1.1	<i>Code</i>	91
3.1.2	<i>Visuals</i>	92
3.1.3	<i>Process (manipulation / control / skill)</i>	94
3.2	Developing More Lines (branches)	96
3.2.1	<i>Code</i>	96

3.2.2	<i>Visuals</i>	98
3.2.3	<i>Process / Manipulation</i>	99
3.3	Variance and Difference: Inheritance	100
3.3.1	<i>Code</i>	101
3.3.2	<i>Visuals</i>	103
3.3.3	<i>Process / Manipulation</i>	104
3.4	Colour, Shape and Form	105
3.4.1	<i>Code</i>	105
3.4.2	<i>Visuals</i>	111
3.4.3	<i>Process / Manipulation</i>	112
3.5	Petals and Flowers	113
3.5.1	<i>Code</i>	114
3.5.2	<i>Visuals</i>	118
3.5.3	<i>Process / Manipulation</i>	119
3.6	Leaf Class and Final Pieces	121
3.6.1	<i>Code</i>	121
3.6.2	<i>Visuals</i>	122
3.6.3	<i>Manipulation</i>	124
3.7	Summary of Colorcalm: Dialogue between Form and Function	125
4	Moving Wallpaper Project	131
4.1	The Concept Stage	137
4.1.1	<i>The Concept Stage: Code</i>	137
4.1.2	<i>The Concept Stage: Visuals</i>	140
4.1.3	<i>The Concept Stage: Process</i>	141
4.2	The Branching Structure	142
4.2.1	<i>The Branching Structure: Code</i>	143
4.2.2	<i>The Branching Structure: Visuals</i>	145
4.2.3	<i>The Branching Structure: Process</i>	146
4.3	Visual and Behavioural Experimentation	147
4.3.1	<i>Visual and Behavioural Experimentation: Code</i>	147
4.3.2	<i>Visual and Behavioural Experimentation: Visuals</i>	149
4.3.3	<i>Visual and Behavioural Experimentation: Process</i>	153
4.4	Extending the Vocabulary	154
4.4.1	<i>Extending the Vocabulary: Code</i>	154
4.4.2	<i>Extending the Vocabulary: Visuals</i>	160
4.4.3	<i>Extending the Vocabulary: Process</i>	161
4.5	Translation and Rotation: Adding 3D Elements	163
4.5.1	<i>Translation and Rotation: Code</i>	164
4.5.2	<i>Translation and Rotation: Visuals</i>	166
4.5.3	<i>Translation and Rotation: Process</i>	167
4.6	Final Development and Variations	168
4.6.1	<i>Final Development and Variations: Code</i>	168
4.6.2	<i>Final Development and Variations: Visuals</i>	171
4.6.3	<i>Final Development Stage: Process</i>	174
4.6.4	<i>Summary</i>	175
5	Overview and Analysis	178
5.1	Introduction	178

5.2	Computational Process: A Unique Workflow	178
5.3	Computational Process: Detail	185
5.3.1	<i>Concept: Constructing the Material</i>	185
5.3.2	<i>Data Structure: Broad Architecture of the Work</i>	188
5.3.3	<i>Data Detail</i>	192
5.3.4	<i>Summary</i>	196
5.4	Visuals	198
5.4.1	<i>Colorcalm Visuals</i>	199
5.4.2	<i>Moving Wallpaper</i>	201
6	Conclusions	208
6.1	Characteristics of Context: Computation in the Design Landscape	209
6.2	Characteristics of Process	211
6.2.1	<i>A Creative Process of Making: Similar to Traditional Craft</i>	211
6.2.2	<i>A Distinct Workflow: Unlike Traditional Craft</i>	214
6.3	Characteristics of Material	218
6.3.1	<i>Concept</i>	219
6.3.2	<i>Data Structure</i>	220
6.3.3	<i>Data Detail</i>	222
6.4	Visuals in Relation to Data Structure	226
6.5	Reflections on Methodology	227
6.6	Future Research	229

List of Figures

Figure 2.1 Screenshot of 'Anemone' by Ben Fry accessed from: http://www.benfry.com/anemone/	64
Figure 2.2 Screenshot of 'Flowers' by Daniel Brown accessed from: http://www.play-create.com/	65
Figure 2.3 Screenshot of 'Oval x3' by Yugop Nakamura accessed from: http://www.yugop.com	65
Figure 2.4 Screenshot of 'ps3_1_praystation' by Joshua Davis accessed from: http://www.prystation.com/	66
Figure 2.5 Willam Morris' 'Willow' wallpaper design.....	86
Figure 2.6 William Morris' 'Jasmine' wallpaper design.....	86
Figure 3.1 An overview diagram of the Colorcalm project available at: http://www.random10.com/colorcalm_research/	90
Figure 3.2 Screenshot from 'lineDrawingSimple2' (2006) accessed from: http://www.random10.com/colorcalm_research/2setting_parameters/applets/lineDrawingSimple2	93
Figure 3.3 Screenshot from 'lineDrawingSimple3' (2006) accessed from: http://www.random10.com/colorcalm_research/2setting_parameters/applets/lineDrawingSimple3	93
Figure 3.4 Screenshot from 'lineDrawingSimple3c' (2006) accessed from: http://www.random10.com/colorcalm_research/2setting_parameters/applets/lineDrawingSimple3c	93
Figure 3.5 Screenshot from 'lineDrawingSimple3f' (2006) accessed from: http://www.random10.com/colorcalm_research/2setting_parameters/applets/lineDrawingSimple3f	94
Figure 3.6 Screenshot from 'lineDrawingSimple3fOOP3' (2006) from: http://www.random10.com/colorcalm_research/3oop_simple/applets/lineDrawingSimple3fOOP3	98
Figure 3.7 Screenshot from 'lineDrawingSimpleInhertance1' (2006) accessed from: http://www.random10.com/colorcalm_research/4inheritance/applets/lineDrawingSimpleInhertance1	103
Figure 3.8 Screenshot from 'lineDrawingSimpleInhertance2' (2006) accessed from: http://www.random10.com/colorcalm_research/4inheritance/applets/lineDrawingSimpleInhertance2	103
Figure 3.9 Screenshot from 'lineDrawingSimpleInhertance2' (2006) accessed from: http://www.random10.com/colorcalm_research/4inheritance/applets/lineDrawingSimpleInhertance2b	104
Figure 3.10 Screenshot from 'lineDrawingSimpleInhertance2c' (2006) accessed from: http://www.random10.com/colorcalm_research/4inheritance/applets/lineDrawingSimpleInhertance2c	104

Figure 3.11 Screenshot from 'cc1' (2006) accessed from: http://www.random10.com/colorcalm_research/5cc1_cc2/applets/cc1	106
Figure 3.12 Screenshot from 'cc2a' (2006) accessed from: http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc2a/	108
Figure 3.13 Screenshot from 'cc3' (2006) accessed from: http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc3/	108
Figure 3.14 Screenshot from 'cc4' (2006) accessed from: http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc4/	109
Figure 3.15 Screenshot from 'cc5c' (2006) accessed from: http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc5c/	109
Figure 3.16 Screenshot from 'cc6b' (2006) accessed from: http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc6b/	109
Figure 3.17 Screenshot from 'cc_basicVersion' (2006) accessed from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_basicVersion	110
Figure 3.18 Screenshot from 'cc_basicVersion2' (2006) accessed from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_basicVersion2	111
Figure 3.19 Screenshot from 'cc_march_03' (2006) accessed from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03	111
Figure 3.20 Screenshot from 'cc_march_04' (2006) accessed from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03	114
Figure 3.21 Screenshot from 'cc_march_05' (2006) accessed from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03	114
Figure 3.22 Screenshot of initial flower tests from 'cc_march_05_flower2b' (2006) accessed from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03	115
Figure 3.23 Screenshot from 'cc_march_05_flower3' (2006) from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03	115
Figure 3.24 Screenshot from 'cc_march05_flower3b' (2006) from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03	117
Figure 3.25 Screenshot from 'cc_aprilTest2' (2006) accessed from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03	118

Figure 3.26 Screenshot from 'cc_aprilTest5a' (2006) accessed from: http://www.random10.com/colorcalm_research/9ccApril_leafclass/applet/s/cc_aprilTest5a	123
Figure 3.27 Screenshot from 'cc_aprilTest5b' (2006) accessed from: http://www.random10.com/colorcalm_research/9ccApril_leafclass/applet/s/cc_aprilTest5b	123
Figure 3.28 Screenshots of final variations of Colorcalm work (2006) accessed from: http://www.random10.com/colorcalm_research/	123
Figure 4.1 An overview diagram of the Moving Wallpaper project available at http://www.random10.com/movingwallpaper_research/	136
Figure 4.2 An illustration of the relationship between the Line and Ball classes.....	138
Figure 4.3 A diagram illustrating the function of the Spring class in the.....	139
Figure 4.4 Screenshot from 'Jan08_plotpoints3' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/1_concept/jan08/	140
Figure 4.5 Screenshot from 'Jan08_plotpoints3_spring' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/1_concept/jan08spring/	141
Figure 4.6 A diagram illustrating the relationship between the Target and the Line objects in the Moving Wallpaper project.....	143
Figure 4.7 Screenshot from 'LinkingLinesTest2' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/2_branches/linkingLinesTest	145
Figure 4.8 A diagram illustrating the movement of the line away from the cursor in the Moving Wallpaper project.....	149
Figure 4.9 Screenshot from 'LinkingLinesTest3' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/linkingLinesTest3	151
Figure 4.10 Screenshot from 'LinkingLinesTest3_outlines' (2008) from: http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/LinkingLinesTest3_outlines	151
Figure 4.11 Screenshot from 'Line_Forces_Test_LRG' (2008) from: http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/Lines_Forces_Test_LRG	151
Figure 4.12 Screenshot from 'Line_Forces_27Feb' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/Lines_Forces_27Feb	151
Figure 4.13 screenshot from 'Line_Forces_27Feb_new' (2008) from: http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/Lines_Forces_27Feb_new	152
Figure 4.14 Screenshot from 'MovingWallpaper_March1_b' (2008) from: http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/MovingWallpaper_March1_b	152

Figure 4.15 Screenshot from 'MovingWallpaper_March1B' (2008) from: http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/MovingWallpaper_March1B	152
Figure 4.16 Screenshot from 'MovingWallpaper_March1E' (2008) from: http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/MovingWallpaper_March1E	152
Figure 4.17 Screenshots from 'MovingWallpaper_March12_BASIC' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/4_ExtendingVocab/March12_Basic	160
Figure 4.18 Screenshots from 'MovingWallpaper_April1_C' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/4_ExtendingVocab/april/	161
Figure 4.19 Screenshots from 'MovingWallpaper_March19b' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/4_ExtendingVocab/March19b	161
Figure 4.20 Screenshot from 'translateExample12C' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/5_Translation/translateExample12c	166
Figure 4.21 screenshot from 'Moving Wallpaper_June13_LRG2Color' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/5_Translation/MW_June13_LRG2color	167
Figure 4.22 Screenshots from 'Moving Wallpaper_June23_unfurl_shape3' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/5_Translation/MW_June23_unfurl_shape3	167
Figure 4.23 Screenshots from 'MW_June27_PlantTypes' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/6_FinalDevelopStages/MW_June27_PlantTypes	171
Figure 4.24 Screenshots from 'MW_July1_PlantShapes2' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/6_FinalDevelopStages/MW_July1_PlantShapes2	172
Figure 4.25 Screenshots from 'MW_July2' (2008) accessed from: http://random10.com/movingwallpaper_research/applets/6_FinalDevelopStages/MW_July2	172
Figure 4.26 Researcher's own plant and flower photographs used as visual reference	173
Figure 4.27 Screenshots of the final variations of Moving Wallpaper work accessed from: http://random10.com/movingwallpaper_research	174
Figure 5.1 A workflow diagram illustrating the network flow between different pieces of software	179

Figure 5.2 A workflow diagram illustrating the feedback loop of the craft-centred design process.....	180
Figure 5.3 A diagram illustrating the computational design workflow.....	182
Figure 5.4 An illustration of the concept element of the Colorcalm project	186
Figure 5.5 An illustration of concepts informing the Moving Wallpaper project	187
Figure 5.6 An illustration of the data structure of the Colorcalm project, and the relationship between classes	189
Figure 5.7 An illustration of the data structure of the Moving Wallpaper project, and the network of inter-related classes.....	191
Figure 5.8 An illustration of the 'flow' of data between classes in the Colorcalm project.....	193
Figure 5.9 An illustration of the 'flow' of data between classes in the Moving Wallpaper project.....	195
Figure 5.10 A visual comparison of stages of development from the Colorcalm project.....	200
Figure 5.11 A comparison between Moving Wallpaper visual and William Morris's Willow pattern image from: http://www.williammorristile.com/small_images/black_willow_sm.jpg	202

List of Tables

Table 2:1 A table outlining where computational design intersects with Arts and Crafts and software centred design.	78
Table 3:1 A list of key variables and functions from the initial line drawing stage: Colorcalm.....	92
Table 3:2 A list of calculations which alter line shape: Colorcalm.....	94
Table 3:3 A list of functions from the Line class: Colorcalm.....	97
Table 3:4 Examples of conditional statements used to alter the branching structure: Colorcalm.....	99
Table 3:5 Overview of Line class as the parent class: Colorcalm.....	101
Table 3:6 Overview of the Mainline class which extends the Line class: Colorcalm.....	102
Table 3:7 Overview of the Branchline class which extends Line class: Colorcalm.....	102
Table 3:8 A list and description of new variables added to the Branchline class: Colorcalm.....	107
Table 3:9 Overview of the Flower class: Colorcalm.....	114
Table 3:10 Description of visual attributes of the Flower class: Colorcalm.....	116
Table 3:11 Core variables of the Flower class: Colorcalm.....	116
Table 3:12 Description of variables added to control branching structure: Colorcalm.....	118
Table 3:13 Description of the key variables: Colorcalm.....	122
Table 4:1 Overview of the Ball class: Moving Wallpaper.....	138
Table 4:2 Overview of the Line class: Moving Wallpaper.....	139
Table 4:3 Overview of the Spring class: Moving Wallpaper.....	140
Table 4:4 Summary of initial classes: Moving Wallpaper.....	141
Table 4:5 Summary of characters allocated to affect growth and structure: Moving Wallpaper.....	144
Table 4:6 Example Strings and description of associated visual effect: Moving Wallpaper.....	144
Table 4:7 Overview of the relationship between visual elements and individual classes: Moving Wallpaper.....	145
Table 4:8 Overview of sine wave affecting shape: Moving Wallpaper.....	148
Table 4:9 Overview of visual changes affected by individual elements of code: Moving Wallpaper.....	150
Table 4:10 Overview of new characters used in Engine class: Moving Wallpaper.....	156
Table 4:11 Key variables of Attributes class: Moving Wallpaper.....	157
Table 4:12 Example of Attributes, Flower: Moving Wallpaper.....	157
Table 4:13 Example of Attributes, Branch: Moving Wallpaper.....	157
Table 4:14 Example of how 'sub' Attributes can be created: Moving Wallpaper.....	158

Table 4:15 Overview of variables used for 3D rotation of line: Moving Wallpaper.....	164
Table 4:16 Additional variables added to control visuals: Moving Wallpaper	169
Table 4:17 Overview of Plant class variables: Moving Wallpaper.....	170
Table 5:1 Summary of difference in code structure between Moving Wallpaper and Colorcalm projects.....	196
Table 5:2 Overall summary of visual and structural differences between Moving Wallpaper and Colorcalm projects.....	203
Table 6:1 Key mathematical functions used to manipulate variables.....	222
Table 6:2 Summary of the layers of code, associated flexibility and effect on the visual element of work.	224

1 Introduction

1.1 Overview

This research project is a practice based study into the use of programming, using *Processing* as the basis of work, to create and develop screen-based designs. The research will focus on the use of computation from the perspective of a designer, trying to establish a context and understanding of computation within the framework of creative, visual practice. This is not a technical, cultural or historical study of code, but a consideration of the formal aspects of code and programming language for creative practice.

1.1.1 Research Question

The overarching question which this research aims to answer is: what are the particular characteristics of the practice of computer programming when used in the context of creative design? In identifying these characteristics from the point of view of a designer, moving from the use of software into programming, the research aims to be of use and of interest to other practitioners and researchers in a similar position. The intended audience for the research is therefore designers who use, or wish to use, and understand computation, its role and practice in the context of creative work.

1.1.2 Aims and Objectives

The aim of the project is to look at the usage and practice of programming code, in order to identify key features and characteristics when applied to a creative design process.

It is the researcher's experience that the nature and usage of programming code in design offers a distinctive environment for creative design practice; a new form of design practice and a new material of expression. In order to gain a full understanding of the character and distinctness of programming for design, the research will consider programming code in relation to wider contexts. This will include a consideration of the following:

- The wider design tradition. i.e. the relationship between the computation for design and other modes of design practice.
- The features of the process and the workflow of the practice: i.e. the relationship between the maker and material.
- The specific syntax and detail of the programming language itself: i.e. the detail of the material.

This can be summarised as consideration of the *context*, the *process* and the *material* of programming for design. These three areas of concern can be summarized into the following questions / objectives:

- Where does creative-computation fit within the context of traditional and digital design landscape?
- What are the key features in the process, i.e. how can the relationship between the maker and the computational material be characterised?
- What are the key features of the material? How can the relationship between the material and the visuals be characterised?

1.1.3 Background

The researcher's background is in multimedia practice and this research has emerged from a professional interest in and use of code as part of creative practice which has developed over the past few years. Interest into multimedia and programming was instigated by experience of design-based multimedia software, e.g. Director, mTropolis, and Authorware. Subsequent growth in the use and capability of software such as Director and Flash, and their associated programming languages, lingo and actionScript, alongside the affordability and growth in multimedia technology, encouraged the researcher to explore more deeply into the understanding and application of basic programming elements for creative practice.

The researcher's progression in software usage reflects the wider development and growth of the multimedia design industry during the 1990s, as increased use and affordability of digital technologies heralded the digitization of the design process, and brought computer and design together into a single arena (Manovich, 2008, p.13).

1.1.4 Why is the Research Relevant?

The expansion of digital technology and increased proximity to programming within design makes this research both important and relevant to current practitioners. The development of digital design tools, and related programming languages as part of design, means that a greater understanding of, and critical enquiry into, programming as means of creating work is important.

Digital technology in the form of software has become a common part of design, however, the author's experience suggests that the practice of writing programming code as a design activity engenders a process and

experience unlike the that of working within the framework of software. As such, programming warrants individual attention and discussion regarding its role and place within the wider context of design. There is, therefore a requirement to take a broader perspective when considering the use of computation for design, to take a closer look at the approach, motivation and application of programming for design, which represents a growing means of creative design practice, and to investigate the use of code as the sole means and material for creating work.

Whilst there are other areas of research which do look at associations between computation and creative practice, for example: generative art; the cultural impact of code; software art; and the crossover between craft and digital software, specific investigation into the formal use of programming code for design is not widely documented by academic research. The formal use of computation in a design context is a subject around which there is very little previous critical research, evaluation or discussion. There is therefore a gap in research relating to the use of code within design practice. This research aims to fill this gap and provide a starting point for the discussion of the creative process of programming in design.

1.2 Methodology

1.2.1 Practice-based Research

This research concerns the practice and process of programming and so a practice-based research methodology was selected. A key feature of practice-based art and design research is that each project develops its own methodology, centred around the specific requirements and questions of the investigation. Unlike other areas of research, 'set models' and specific methods for art and design practice have not been defined. There are, however, some general characteristics of practice-based methodology that define overarching characteristics which are central to the use of practice as research.

A central characteristic to a practice-based methodology is the idea of reflective practice (Gray & Malins, 2004) a process defined by a cycle of evolving practice informed by analytical and reflective work. This process of informed, reflective work as defined by the cycle of working and reflecting (Shaw, 2007, p.161) can also be summarized as the "reflective" or the "engaged practitioner" (Marshall, 2008), in which the research is conducted as part of an ongoing engagement with programming as part of practice: a self-critical model of inquiry in which the investigation feeds into and develops a practical understanding of the subject.

1.2.2 Features of this Methodology

Using the idea of the 'reflective practitioner', the methodology is characterized as an exploratory model of practice-based research, in which the process of using code is examined by practical investigation, underpinned by an initial contextual examination into the role of programming within a design context.

The initial contextual review provides a theoretical framework from which the practice can be explored, and the specific scope and the goals of the practice can be defined. Issues and themes developed from the initial contextual review illuminate, inform and direct the aesthetic aims and focus for the practice. The contextual review therefore provides a framework to inform the discussion relating to the ideas and practice of programming, and provides a platform for the interrogation of the practice-based elements of the work. The practice is instigated by a central theoretical discussion, which frames the specific nature of the practice and forms a key foundation to the rest of the work, generating a project which is informed by the interaction between theory and practice (Shaw, 2007, p.44).

1.2.3 Contextual Review

Unlike other established modes of design work, the history and context of design using programming has not been established. There is, therefore a need to establish a context for the practice, to understand how far the use of programming can be considered to be part of contemporary, software-driven digital design practice, and how much it connects with traditional elements of design. The aim of the review is to contextualize the work of programming as a means of 'making' within the landscape of design practice, bringing into focus key themes which relate to the broader relationship between the designer and the material for design. This is done in order to establish similarities and differences with programming when considered alongside both traditional and digital approaches to design.

Throughout the history of design, technological advancement and development have altered the outlook, the process, the material and the objects of design. Each development, from the printing press, through to modern digital techniques have brought new challenges, processes, and materials for the designer to work with. The key themes defining the ongoing changing relationship between the designer and his / her material therefore

reflect the challenges that each new development brings to the outlook (ethos), material, process and object of design. The research will use these headings of 'ethos' 'material' 'process' and 'object' to structure the contextual study, and in order to make direct comparisons between computational, traditional and software-driven design. The contextual review will focus upon the design philosophy and approach of the Arts and Crafts movement, contemporary software-driven design and computational design in order to identify the characteristics of other design practices. These areas have been selected because they highlight the issues of material, process, object and ethos which are key to understanding the context of computational design. The contextual review will therefore aim to situate programming for design against the landscape of traditional and digital design, placing it within the context of a wider discussion of making i.e. in relation to materials, ethos, process and object. This discussion will be used to frame, and define the direction of the practice.

1.2.4 Practice

A central aim of the research is to investigate the process of using programming code to identify key characteristics of process and therefore to examine the code as the material from which the digital object arises. The practice of using code is therefore a highly important element of the research. Using the contextual review as a framework for investigation the practical work will closely examine the detailed usage of programming language as a *process*, and as a material in order to establish the characteristics of *making* with code. The research will look at the detail of the formal aspects and instructions of code and associated visual output, considering the detail of the code (language and structure), the process and visuals.

Although artistic method can include an emphasis on chaos as a core element in artistic experimentation (Gray & Pirie, 1995) as a means of inquiry, the nature of programming, being a structured, systematic environment, together with the requirement to outline the details of the process, demands a considered, systematic mode of practice-led inquiry. Rather than employing an open-ended method of inquiry (e.g. one led by experimentation alone), the practice methodology for this project will be one which operates within set parameters and boundaries towards a specific aesthetic objective. The specific details of the aesthetic boundaries will be defined as a result of findings from the contextual review.

Hand-written code is used to develop 'experiments' and 'sketches'; a series of artworks, each building on the last providing a means to compare processes. Practice is characterized as a structured, methodological and reflective study into creative practice of code as a formalistic rather than cultural construct (Cramer, 2002). Concentrated attention is therefore be paid to the structural elements of the code: i.e. details of language, instructions, vocabulary and algorithm as a means of producing types of lines, shapes and forms with specific characteristics and behaviours. Each stage of the process is documented to reveal the stages of the practice and to show the relationship between the code and the visuals at each stage of the work. Documentation and observation of the detail for each stage form a core part of the method and will be recorded in the form of an online digital 'sketchbook'; a reflective digital journal of Processing 'sketches' (Gray, 2004, p.59 & p.113), building up to give a detailed visual account of the project work. Documenting and commenting upon these elements in this way enables a detailed picture of the usage and process to emerge, allowing observation, description and reflection of the dialogue between maker and material, code and visuals, to take place.

The decision to take a formal, structured approach and to use only hand-written code may seem to set unnecessary boundaries and limitations upon the artistic direction of the work. However these boundaries offer a degree of control which allows a close and considered look at the detail of the work and the process; providing a firm focus and framework from which the details of the process can be considered. This approach which is distinctive from the use of software, forms an important aspect of the research method allowing a close and considered systematic look at the formal aspects of the process of code. The overall emphasis of the research practice is therefore on process rather than on the innovation of the final object itself. The final object of the practice is significant as being part of the process, and part of the "interrogation of practice" (Shaw, 2007 p.166) within the context of using code. The new knowledge of this research will not be embodied by the final piece of work but "clustered around it" as a broader set of outcomes (Shaw, 2007, p.166).

1.3 Definitions and Scope

1.3.1 Scope

The emphasis of this project is on the investigation of the formal aspects of programming and specifically an investigation into the material, the process, and the context of code within the sphere of creative design practice. Although use of computation within art stretches back a long way (Gere, 2002), a wider commercial interest in, and use of, digital technology only relatively recently generated the means and motivation for a design-led usage of interaction technology. Exploration of new expressions of screen-based design, afforded by the growth and accessibility of computer technology (particularly the Apple Mac) and associated software, encouraged designers to explore the creative potential of the new media environment and to push the boundaries of digital design.

The boundaries of design sometimes cross into art: The 'Web Wizards' exhibition at the Design Museum (Nov 2001 - April 2002) provided a valuable insight into the experimental, exploratory nature of new designers working at the forefront of new media design during that time. The exhibition included work from prominent 'design stars' such as Joshua Davies, Yugo Nakamura, and Daniel Brown, and was promoted as follows:

As digital and interaction design evolve, a new generation of design stars is emerging. These are the designers who are not only mastering new technologies in their ongoing experiments, but whose innovations are inspiring other designers and setting the benchmark for design excellence on the web and other media.

(Design Museum, 2002)

The experimental nature of digital and interaction design blurred the boundaries between new digital design and art. The 'design' label used in the subtitle of the Web Wizards show is questioned:

The most remarkable aspect of Web Wizards is that despite being sub-titled 'Designers Who Define The Web' the only design project on show is Tomato's Connected Identity for Sony. The rest of the work is art...

(Macdonald, 2002)

Although the abstractions of Joshua Davis' Flash experimentation may have visual resonance with abstract art (Manovich, 2002a), it is important to note that the practitioners firmly root their work within the context of new media design. Notable computation designers including Daniel Brown and Yugo Nakamura for example site themselves and their practice within the context of design practice (Design Museum, 2006). This thesis acknowledges that

the discussion to the validity of work as art or design is complex and one which is bound to continue. It is not the role of this thesis to make judgments about this, it is enough to recognize that this usage of Flash programming to create new media work was cited as being design work by the practitioners themselves.

The use of computation within creative practice is not exclusive to the realm of design. Computation is used in different contexts across a range of creative digital practices including art and animation. It is necessary to clarify the scope of the project; to define and discuss areas of overlap and difference between areas of related creative practice, and to define the specific design-centred focus of this project. Specific attention will be paid to digital art, and animation. Each area will be briefly outlined and areas of similarity and difference highlighted, in order to provide a clearer understanding and definition of each, establishing and defining the scope of the research.

Artistic practice has often been at the forefront of experimentation with new types of media and technology, and the introduction of computational technology has encouraged a whole new sphere of artistic engagement with programming which is known as 'software art'. Software art is an umbrella term covering a wide range of creative activity centred around an artistic approach towards software, which recognizes its "aesthetics, poetics, and politics" (Goriunova, 2007). Software art is therefore not limited to a single specific practice, and can include artistic interaction with software either as a cultural construct or as a formalistic set of instructions (e.g. a programming language). Florian Cramer (2002) identifies these two aspects of software art labelling them "software formalism" and "software culturalism". Cramer suggests, however, that a definition of software art which is *exclusively* focused upon on the formal detail of the programming language limits its practice and understanding into an unnecessarily rigid set of parameters,

which exclude a range of concerns, themes and approaches regarding the cultural, social and political elements of software culturalism. This idea is continued by Lev Manovich (2003). In an essay relating to the 2003 Ars Electronica festival, he dismisses the idea that a systematic concern with the formal elements of code should be considered as art at all. He argues that a pre-occupation with the material and formal uses of code (e.g. the structure and syntax) limits the understanding of software art to process rather than true artistic concepts and "strategies". (Manovich, 2003).

Both of these theorists emphasize the idea that whilst a formalistic study of code may form some part of the practice and approach of software art, the true nature of software art practice must be inclusive of the broader cultural and political aspects of code, reflecting wider artistic concerns. Therefore whilst some overlap between areas of this research and software art can be identified, the specific focus of the research into the formal elements of code are too limiting to include the broader political, cultural and artistic concerns of software art. This research will therefore not include a discussion of software art, being more specifically focused on a design-centred formalistic review of the material qualities of code.

A second area of potential overlap concerns digital animation. Animation is traditionally described as the "process of creating a moving images by moving rapidly from one still image to the next" (Austin & Doust, 2007, p.180) However, despite the traditional emphasis on a sequence of still images, the development of digital animation tools and environments has meant that areas of crossover between animation and programming are growing. Computation has become an increasingly important part of digital modelling and animation software and is often used to simulate physical movement and environments, for example the movement of particles, wind, water, smoke, etc. (Austin & Doust, 2007 p.120). The role between creator and director of animation becomes blurred: computational parameters are

created, run through the computer, and the 'animator', acting as director and editor, lets the computer calculate the movement until the desired result is achieved. Lev Manovich (2008, p.182) describes this type of use of computer programming to create physical movement as a "hybrid between animation and computer simulation".

Despite areas of commonality between the use of computation and digital animation, there are key differences which distinguish creative computational work from animation. The final outcome of an animated piece of work (whether created via traditional, or digital processes) is a single, final rendered piece. Any code that may have been used as part of the process is no longer required after the final render. Computational work however, is not rendered into a single final finished animation. The code remains at the heart of the final piece of work and in fact the code *is* the final piece of work, which is run and viewed. Maintaining code as the focus, the central element of the work, means that the work is *generative*; it may look animated, i.e. time-based and moving but each rendering of the work produces different sequences and changes as the code renders different outputs, "like a computer graphic animation, except without a predetermined beginning or end" (Maeda, 1995).

Computation therefore may be used as a *part* of an animation, but has a different use and relationship with the final piece when used as the core element of the work. Direct emphasis on code as the sole generator and artefact of computational work alters the relationship with the core process and final object of the creative process, distinguishing it from that of the animated piece of work. So whilst areas of crossover and overlap between computation and animation can be identified, the nature and emphasis of programming as a practice is sufficiently different to that of animation to warrant being the specific focus of this study. Investigation into the relationship between programming and animation is a credible area of focus

for a different piece of research but will not be considered during this study, requiring, as it does, a different approach.

A final brief mention needs to be made to a third possible area of overlap which concerns the relationship between digital tools in the process of making physically 'crafted' objects. This is an area of discussion which is a frequent topic of debate, especially in the context of contemporary craft, as evidenced by the themes of some recent craft conferences, for example, *New Craft-Future Voices' International Conference* (Dundee University, 2007) and *Challenging Craft* (Gray's School of Art, 2004). However although some of the themes of this research, regarding the idea of crafting the digital object intersect with the themes of contemporary craft, the core focus of this project is upon the usage of computation, rather than other digital software tools, as the sole means of creative activity. Rather than taking a broad look at digital tools, including software, 3D modelling, etc. as the process of making, the specific focus of this project is that of using computation as part of the design process, and the creation of work via the act of writing computational code.

In summary, although the subject of this research will naturally overlap with themes, ideas or materials which relate to other creative fields of study (in particular art, animation and contemporary digital craft) none of these areas will be the focus for this research project. This research will instead focus upon an investigation into the formal elements of the material and process of computation within the context and tradition of design. The aim, therefore, is to create work which expresses an understanding of the formal elements of programming code, and which links it with an understanding and acknowledgement of a wider design tradition. The research intends to build upon a formalistic design approach, rather than a conceptual art-centred tradition, for the creation of computational graphics. In terms of defining the nature and type of computational work, John Maeda's (1995) identification of

time based or "reactive" graphics provides an interesting touchstone for the beginning of this work (i.e. screen based graphical work which has a generative time-based and / or 'reactive' quality to it, graphics which contain a kind of kinetic movement, and generative quality). The idea of time based graphics helps to establish a type of computational piece of design which is situated away from that of art or animation, and is a distinct type of generative, computational design.

1.3.2 Defining Terms

Some 'technical' vocabulary relating to the terms of the research needs clarification.

Computation: Computation in this instance refers to the creation and use of algorithms within a high-level programming language, used to generate logical instructions to be compiled and output as screen-based graphics. The level of programming in this research refers to that which is typically used by graphic designers or artists, and systems widely adopted within the creative community. The specific programme language used throughout this work is Processing (<http://processing.org>), a Java-based language, designed to allow artists and designers to programme images, animation and interactions.

Generative: Generative work involves the use of computation to create and run ever changing and evolving graphics. A computer programme is written and set running; as it runs shapes are generated and changed according to the data and code of the programme. The programmer has control over the programme, defining the overall structure and writing the initial code. However, the computer runs the code producing random, sometimes unexpected results from a set of variables which 'evolve' over time, as they are read to, and change each other. Unlike animation, which runs and

produces the same results each time, generative work changes each time it is run, as the data of the work is updated in 'real-time'. Cox, et al (2001), states that:

Crucial to generative media is that data is actually changed as the code runs... the power of code allows this [calculation] to happen in 'real-time', and the effects are largely unknown until execution.

Generative work is therefore an automated art process in which computer code is usually used as the means by which the automation is defined and controlled. Although generative work does not always involve machine or computer programmes to generate the piece, and can involve other sets of "natural language rules ... or other procedural inventions" (Galanter, 2005), in the context of this study the term will be used with specific reference to the use of computer code.

Object Orientated Programming: Object Orientated Programming (OOP) is a way of structuring code as a collection of individual objects which work together, send messages between each other and act upon each other, and is a common way of structuring code. Operations which manipulate the data of the object are stored as 'methods' inside a 'class' rather than being separated as they would be in procedural processes; a 'class' therefore defines the characteristics of an object (Crutzen & Kotkamp, 2006). Defining objects with their own behaviours which can act upon and interact with other objects is a core way of creating and thinking about the programming structure, and offers important and influential ways of thinking about and structuring code. Malcolm McCullough (2008, p.98) observes that:

Object-orientation is at the very root of one of the biggest advantages of digital media, namely the ability to operate on abstractions as if they were things.

Open Source: 'Open source' software allows the free viewing and modification of source code and is motivated by a philosophy and approach to the development of software which encourages a community approach to software development. The philosophy of open source towards the development of code encourages a community of programmers to share use and view source code, believing that source code is improved by the process of shared development (Krysa & Sedek, 2006).

Software-Centred Design: The term software-centred design is used in this research as a short-hand way of defining contemporary digital design practice which uses software as its primary means of production. The term is applied to creative visual design production methods which employ point-and-click digital software and used as a means to distinguish this from design which uses hand-written programming code as its primary means of creation ('computational design').

1.4 Dissertation Outline

The research document has the following structure: Chapter 2 discusses the contextual review element of the research used to establish the foundation and direction for the practical projects, looking in turn at the three main areas of design for review, comparison and discussion, i.e. digital, software-driven design (2.1), computational design (2.2) and traditional design (2.3). The final section of chapter 2 summarizes the comparison of these areas and identifies areas of connection and commonality between computation design and other realms of design practice.

Having established the direction and context for the computational practice, chapters 3 and 4 outline the specific processes of the two major projects, outlining and documenting each stage of the process to reveal the details of each project, the use of code, the nature of the visuals and the nature of the process. A comparison between both pieces of work, comparing overall processes is conducted in chapter 5, and summaries and conclusions are outlined in chapter 6.

2 Contextual Review

The contextual review forms an important foundation for the practical element of the research, which aims to situate computational design practice within the broader landscape of traditional and digital processes of design. Before discussing in detail the place of computational design in relation to its broader design context, it is first necessary to sketch out the current landscape within which this research is being undertaken.

In terms of related doctoral research, the specific focus of this thesis (the use and process of programming as part of creative design practice) is one which has largely been overlooked. Searching online databases of doctoral work (i.e. theses.com, eThos.bl.uk, ARTbibliographies Modern) targeting and searching specific institutions (e.g. MIT.library.edu) and searching related fields of research via conferences, conference papers and journals, revealed some doctoral research which relates to broader areas including the use of digital technology or software as part of creative design and craft-based work (e.g. Marshall, 2008), but no doctoral thesis which specifically matched this area of study. The closest completed work was the Masters thesis of Joanna Berzowska (1998), which is cited elsewhere in this document. In the wider field of academic grey literature, individual conference papers and academic journal articles discuss and examine ideas and examples of code usage as part of artistic, creative practice, and are cited in relevant sections of this document (e.g. Maeda, 1995 and Cramer, 2002). The fact that the specific focus of this study has largely been overlooked for doctoral work further strengthens the case for the originality of the thesis.

Despite the lack of formal doctoral academic research in this area, the use of programming as part of creative design practice and its relationship with software-generated design has been part of an ongoing discussion which has been explored through the work of a number of design practitioners.

Reviews and articles in professional design journals (e.g. *Creative Review*, *Computer Arts*) as well as individual design projects have highlighted an ongoing debate into the role, status and position of code as a legitimate means of digital design. This debate questions the perceived 'limitations' of software as a design tool, and highlights a shift in attitude away from software driven design, towards a design approach which encompasses traditional design values. Significant art and design exhibitions have recently raised the profile of this subject, most notably *Decode: Digital Design Sensations* at the Victoria and Albert Museum (2010) and *Code:Craft* (2010), part of the Lovebytes festival. Both of these shows brought into prominence the wider discussion regarding the legitimate use of code as part of art and design practice: exploring the nature of the relationship of code as part of creative work; outlining its uneasy relationship with software-driven design; and hinting at links between programming code and traditional design practice. The emergence, motivation and timing of these exhibitions reflects the current growth in interest and relevance of this subject and the broader discussion relating to the status of creating within the digital environment. It is against this backdrop that this research has been conducted.

This overview of the professional, contemporary backdrop against which this research has been conducted, serves to highlight the way in which, although ignored by formal doctoral research, the debate surrounding the use of code in a practice-based design context has been developing over a period of time. It also highlights the uneasy relationship between code and software-centred design, as artists and designers using code seek to 'free' themselves from the 'limitations' of the software environment, and make connections with forms of traditional design. This chapter will therefore discuss in more detail the relationship between software-centred and code-based design, as well as considering computational design within the context of traditional design practice. In order to reflect this wider perspective on the discussion, the ideals and approaches of traditional design, as typified by the Arts and Crafts

movement, will be introduced, allowing the research to make comparisons in relation to the context and understanding of traditional design practice.

Although this is not an art history thesis, and the Arts and Crafts movement represents only one moment in the traditional design landscape, it is important for the contextual review to consider the Arts and Crafts movement, as it provides a valuable means of encapsulating and articulating core values and ideals of traditional design practice. As well as providing a point of conceptual reference, the Arts and Crafts movement also provides a valuable focus for the practice based element of the research. The themes and aesthetics of the movement provide visual and conceptual direction for the practical project, inspiring both conceptual approach and organic aesthetic of the work. The contextual review will therefore introduce the Arts and Crafts movement as its particular 'case-study' for comparison, considering it alongside a discussion of software-centred and computational design. Each element of design: software-centred, computational and traditional design, will be discussed using the headings of 'ethos', 'material', 'process' and 'object'. These headings cover core elements regarding the relationship between maker and material, and are important in gaining an understanding of computational design as an engaged, creative process. The first area for discussion is software-centred, digital design.

2.1 Software-centred, Digital Design

Digital technologies have "an unassailable place in postmodern culture as the principle media of the post-industrial age"

(Crowley & Jobling, 1996. p.286)

2.1.1 Introduction

Central to this discussion is the idea regarding the nature of the relationship between designer and material, the relationship between designer and technology. Being a commercially led industry the nature of design has always had a close link with technology. Advancement in production methods and technology have had a direct effect on the nature of industry; as processes and materials have changed, so have the processes production and visual style of design. Graphic design therefore has natural, and close links with technology, and as such is a "key indicator" of technological advancement (Olding-Smee, 2002, p.7).

This section will discuss the ethos, material and technology of software-centred, digital design, outlining the landscape and use of computer software in relation to the field of design practice. The discussion will subsequently move on to focus on the specific, distinctive use of computation as part of design practice in the next section.

Although a theoretical discussion of, and approach to, graphic design is a relatively new area of study, and can therefore be problematic (Lupton, 1997, p.113), it is necessary to sketch an outline of the prevailing landscape of contemporary software-centred digital graphic design in order to allow identification of the key characteristics of digital design.

2.1.2 Ethos

Although not driven by a unifying conceptual or moral aim, the era of digital design was uniformly dominated, and unified, by the rise of software technology. The increased use and affordability of digital technology during the 1980s and 1990s had a profound, and lasting influence on the landscape process, environment and distribution of design: the Graphical User Interface (GUI) gave designers the ability to edit objects on screen, using direct point-and-click manipulation, affording a wider means of distribution, a digitized 'mass production', of the digitally designed object. Heralded as a "major revolution in graphic arts" (Heller, 2002, p.4), the digital revolution therefore marked a shift from the "tangible solid material" of ink, pen and paper, to the "intangible, 'floating world' material of the digital (ones and zeros)" (Betsky, 2000 cited by Poynor, 2003, p.113-114). The use and rise of "cultural software" (Manovich, 2008, p.11) i.e. software programmes for the creation and accessibility of media objects and environments, meant that software moved to the heart of all cultural and creative production:

During one decade a computer moved from being a culturally invisible technology to being the new engine of culture.

(Manovich, 2008, p10)

Software therefore unified the creative landscape, becoming the centre of creative practice and "the new engine of culture". Manovich (2008) neatly sums up the situation: media became "software-ized" (p.36). The development of software as a core part of creative and cultural practice within a single computing environment dominated and unified the approach and process across the creative spectrum, and brought about massive changes to the field of design. Although not universally liked, the influence and ubiquity of shift towards "cultural software" cannot be underestimated, and the impact of software on culture in general and on graphic design in particular cannot be overemphasized.

2.1.3 Materials and Technologies

The digital revolution of modern computing transformed the material of contemporary graphic design into a 'de-material' of conceptual space. The tools of design practice switched from being physical to digital, from tangible solid materials to the virtual "metamaterial" (Manovich, 2008) of pixels on a screen (Betsky, 2000 cited by Poynor 2003, p.113-114). The digital computer material as a 'metamaterial' unified all the design tools, as individual pieces of software, into a single digital universe.

In the process of the translation from physical and electronic media technologies to software, all individual techniques and tools that were previously unique to different media "met" within the same software environment

(Manovich, 2008, p.24)

Traditional media boundaries and specialisms were destroyed as the computer material consumed the all areas of creative design practice. Crowley & Jobling (1996, p.286) describe how the "dramatic effect" of computer technology was felt "not only on the formal languages of graphic design" but also "on the technocratic specialisms that define professional activity". Martin (1990 cited by Poynor 2003, p.108) agrees:

Perhaps the most profound implication for the future is that digital technology collapses all media into a single desktop tool speaking one digital language.

Designer April Greiman described the digital media as being a "new paradigm, a conceptual magic slate" (Poynor 2003, p.96), but how new is the new material? In many ways, the 'new' media of digital design can be seen as a reflection, a "remediation" (Bolter & Grusin, 2000), of the old media, re-

presenting the attributes of old media in a digitized format (e.g. the 'digital darkroom'), however, even if some of the 'surface' values of the objects, digital photographs and videos may have the same look as 'old' media, Manovich (2008, p.59) argues that by becoming part of the digital environment the underlying 'DNA' of the media is changed. Old, physical, media elements are transformed by the digitization process, to become part of an environment which is never finally defined, in a state of "permanent extendibility". In 'The Language of New Media', Manovich (2002b) outlines what he sees as the key characteristics of new media and outlines them as: Numerical Representation, Modularity, Automation, Variability and Transcoding.

This list highlights the way in which the process of digitization reduces the physical, cultural object into a set of data which can be reproduced, repeated and changed. In this way Manovich details what he sees as the core attributes of the new media. He outlines the way in which the media, reduced to algorithm, is subject to the computational process: modularity, automation and variability to create a new object with new meaning (transcoding). Transforming objects from the physical to the digital material therefore is a process of reduction, and variability in which all objects are represented as individual pieces of data within the same shared data environment. Graphical assets, text, video, images are reduced to a computational, numeric, representation, a piece of data or algorithm within a single data structure (Manovich, 2002b, p.196). In this way, every media object becomes simplified into a single digital 'unit' which can be altered and transformed via software manipulation. Graphical objects can be combined, copied, repeated, transformed, scaled and edited to form endless varieties of image and form. It is a flexible material: a digital universe in a state of 'permanent extendibility', with the potential to be transformed and changed into endless varieties of form.

2.1.4 Process and Skill

The digitization of all media objects (graphics, text, video, etc.) into a single digital universe of multiple software tools and operations had a profound impact on both the process and the skill-set of design. Specialist operations, and processes were merged into a single computer environment, and previous hard lines and physical barriers between different types of media production were eradicated. Creative media designers, previously separated by a range of specialist operations and activities all operate within the same digital environment in which the same design tools and software operations are common across design areas "contemporary designers use the same small set of software tools to design just about everything" (Manovich, 2008, p.141).

Compatibility between software, and the increasing ability to share individual graphics as data files across software, generated an increasing 'hybridity' of process and aesthetics. The same software "techniques and strategies" were used across the digital universe (Manovich, 2008) and design operations became limited to a narrow pool of software tools and a similar set of software operations. The shared digital environment encouraged a 'modular' design workflow, giving rise to the power of the ubiquitous 'import' and 'export' commands as a key part of the design workflow (Manovich, 2006). The ability to move graphics between software, to mix and match, image, text, and video; to position, reuse, combine and transform media within the same digital universe encouraged a culture of 'remix' to dominate the process and aesthetic of the digital work.

Software production environment allows designers to remix not only the content of different media, but also their fundamental techniques, working methods, and ways of representation and expression.

(Manovich, 2008, p.25)

Instead of working 'from scratch', the digital 'metamaterial' encouraged the designer to work as editor rather than as creator: reassembling, re-working and re-mixing existing visual elements into a collage of visual assets; a "remix" (Poynor, 2003) culture of copy-and-paste, in which "authentic creation" is replaced by selection from a menu (Manovich, 2002, p.120). The facility to copy-and-paste did not of course remove the ability to create from scratch, however dominance of the remix approach made this approach 'natural'; it 'legitimized' what Manovich (2002b, p.125) calls the "logic of selection", in which:

Pulling elements from databases and libraries becomes the default; creating them from scratch becomes an exception.

In addition to encouraging a copy-and-paste approach, the nature of the digital, 'no-rules' environment (Betsky, 2000, cited by Poynor 2003, p.114) encouraged an instant approach to the process of design. After the restrictions of the "straightjacket" of earlier design formalism, (King & Küsters, 2001, p.7) the 'no rules' digital world developed an environment which encouraged spontaneity and an intuitive approach to work, without the requirement for a pre-set grid. (Burns, 1992 p.39). Without the physical restraints, and grids of old processes, the new design machine encouraged the multi-layered aesthetics of deconstruction, a visual "spiral of excess" which "degenerated" into a "visual free-for-all" (King & Küsters, 2001, p.7).

This shift in emphasis from hand-generated techniques to the instant copy-and-paste culture of digital processes changed the process of design and challenged traditional ideas of skill and creativity within the design community. Despite the creative freedom afforded by the new software tools, eliminating need for laborious tasks, the ease and speed of digital processes were not always lauded as a positive influence.

The use of computers has rapidly become widespread across almost all professions and, in graphic design, all but eliminated the role of craft as a practical element of the production process.

(Olding-Smee, 2002, p.7)

Detractors claimed that digital software development and usage had a negative impact upon design: overemphasising technology for its own sake, focusing on the surface qualities of work, a "style over substance" approach to design (Crowley & Jobling, 1996, p.278). It was argued that the new technology undermined previously held notions of skill, reducing the role of designer to a technician selecting design from a menu of "set recipes" (Olding-Smee, 2002, p.7) of a "limited repertoire of fashionable [design] styles" (Aydin & Budak, 2005). In this way design skill and creativity was seen to be superseded by the technology itself: "any creativity is down to the software programmer" (Olding-Smee, 2002, p.7).

The dominance of software tools replacing the physical process of design can therefore be seen to challenge the status of 'skill'. Emphasis on skill (time, effort plus talent) can be seen to have been replaced by the techno-centric, point-and-click selection process of the GUI software interface. John Maeda (1999, pp.19-20), expresses this idea as follows:

Computers as they are used today have nothing to do with design skill... [the] form of disciplined approach toward understanding one's medium is what we traditionally associate with skill... due to the advent of the computer, mechanical skills have taken secondary importance to the skills required to use complex software tools.

It is interesting to note that designers have responded to these criticisms regarding the perceived shortcomings of the digital design environment in

opposite ways: some designers have reacted by encouraging a return to handmade processes and whilst other have attempted to re-apply skill and rules to the digital world via the use of computation and code (King & Küsters, 2001). It is this second response which forms the focus of this research.

2.1.5 Object and Artefact

As both the material and process of design shifted into the digital realm of the free floating metamaterial, so the aesthetic and nature of digital design object changed. Before considering the aesthetics of digital graphic design, it is useful to sketch out the wider cultural landscape into which the digital design process was born; a landscape dominated by the legacy and influence of postmodern thinking. Whilst it is not within the scope of this thesis to give a full and detailed account or discussion of postmodernism in all its facets, it is however enough to say that postmodernism, being characterized by diversity and reproduction (Crimp, 1990, p.53), and as an environment in which previously held certain, absolutist ('modernist') ideals are replaced with a culture of uncertainty and multiplicity (Watson, 1998), can be seen to go hand-in-hand with the digital object, and the digital aesthetic of this period. What was previously solid, certain and physical (modernist) was replaced by a process and style of design which reflected some of the key ideas and attitudes of 'postmodern' culture: the flexible, changeable, uncertainty of the digital design environment reflected the uncertainty and multiplicity of the postmodern culture into which it was developed.

The copy-and-paste, approach together with the ability of the metamedium to combine many different visual elements, encouraged the construction of images through the collection and combination of images and styles from a wide variety of sources. During the 1980s, groups of graphic designers,

including Katherine McCoy, April Greiman, Wolfgang Weingart, etc (King & Küsters, 2001) took the opportunity afforded by the new technology to experiment and explore a multi-layered aesthetic, to develop a 'digital' style. From this experimentation, a digital postmodern aesthetic emerged: A "free-for-all" (King & Küsters, 2001) of visual deconstruction and excess (Lupton, 1997) in which a risk-taking, experimental process of layering image and text (Heller, 2002) tested and stretched legibility to its limit (Kirschenbaum, 2005).

During the 1990s the "me too" generation of designers (Heller, 2002) continued to indulge in self-referential, layered, re-mixed visual experimentation. Visual parody, pastiche and quotation from a range of visual and cultural sources, e.g. Japanese popular culture, or science-fiction, were employed as playful visual devices. Newness was formed, not from creating but from "resampling and remixing" already existing styles (Poynor, 2003). Styles and genres were remixed recreated in amusing, playful or trivial ways as the aesthetic landscape of style over substance reflected the "flattening of values" of the postmodern culture (Jobling, 1996, p.276). Just how far graphic design was truly part of postmodernist theory is certainly a debatable point: Lupton (1997) suggests that the idea of deconstruction within graphic design is more about the style than a design 'movement', and Crowley & Jobling (1996) suggest that the links between postmodernism theory and graphic design are over-played; emphasising the fact that the link with postmodernism arises from natural characteristics of graphic design itself (e.g. an inherent emphasis on the reproducible image, and an aesthetic which naturally operates between 'high' and 'low' culture). However, whether or not graphic design can truly be aligned with postmodern theory, the digital aesthetic, and technology characteristic of graphic design during this era can be seen to reflect the ideas of visual deconstruction which coincide with the postmodern culture. Aydin and Budak (2005) describe the link between postmodernism and the digital design aesthetic as follows:

Digital technologies are a product of our Postmodern culture that produces styles automatically and juxtaposes most diverse fragments in its creations.

Whether truly postmodern or not, the digital universe, which provides the ability to combine, import and export different media, has made a large impact on the aesthetics. Where aesthetic distinctions between different areas of design were kept separate, each having its own aesthetic (animation, graphic, design, films, etc.), the shared nature of the computer environment encouraged the "aesthetics of hybridity" (Manovich, 2008) in which aesthetic styles between film, animation, graphics are shared and swapped until visual distinctions are no longer apparent.

When you look at pages featuring the works of a particular designer or a design studio, in most cases its impossible to identify the origins of the images unless you read the captions. Only then do you find that which image is a poster, which one is a still from a music video, and which one is magazine editorial.

(Manovich, 2008, p.140)

As well as changes in the aesthetics of the design object. The metamaterial of the digital environment also changed the nature of the digital object itself. As has been previously discussed, the digital environment allows objects to be represented in many different forms and single objects are not restricted to a single format. The graphical object as a piece of data can be reused and re-represented upon multiple platforms and in multiple ways. Digital objects are therefore characterized by the notion of multiplicity; an object which exists within and as part of the digital realm is therefore able to be used, re-used, re-applied and distributed in a range of different ways.

2.1.6 Summary

The work and environment of software-centred design is not focused around a moral ethical understanding of design but by the commercial development and use of design software packages.

The digital realm replaced the solid material of previous eras of design, changing it from the physical to the intangible. Software equivalents of real design environments (e.g. darkroom or printing press) altered the material from the solidity of ink on paper to the cut-and-paste world of pixels on a screen. The transition of design into a software driven environment altered both its process and the aesthetic. Previously well established boundaries between design disciplines were lost as all creative activity 'collapsed' into the single digital environment. Individual skills and processes were amalgamated into the point-and-click skill set of digital software and drop-down menus. The ability to copy and move graphical elements across the digital environment changed the process into a modular network of related software elements. The process and culture of selection, edit and remix replaced that of 'making from scratch' and this is reflected in the layered, aesthetic of digital graphics.

2.2 Computational Material: Computational Design

Graphic work ... rarely exceeds the mannerisms of its software environment.

(Reinfurt, 2005, p.7)

The introduction and use of computer technology altered the landscape of design. The rise of digital technology as a process and tool for digital design meant that software became the dominant tool for digital design, altering its process and aesthetics. A number of designers, however, questioned the

authority and dominance of the new digital tools and their impact upon creative expression. Marius Watz (2003), for example, expressed the view that "designers have become dependent on software, tools ... forcing the designer to adapt her work to the decisions and metaphors chosen by the programmer". Consequently a form of design practice developed which actively questioned the role of computer software in process and practice of graphic design, pushing the boundaries of design and technology, encouraging designers to become increasingly involved with the material of the computer. The desire to engage with, and develop a critical approach to the computer encouraged the development of a "new breed of graphic designer" keen to "crack open commercial software" (Womack, 2006). These designers actively involved themselves directly with the nature and material of digital technology by using programming as part of their practice; "confronting the computer on its own terms and in its own language" (Reinfurt, 2005, p.3). In his article 'Generation Flash', Lev Manovich (2002a) outlines the situation:

After GUI-based applications ... became commonplace, many computer artists continued to do their own programming ... it was taken for granted that even in the age of GUI-based applications a really serious artistic engagement with computers requires getting one's hands dirty in code.

This 'new breed' of designer represented a new mode of design inquiry which will be referred to within this discussion as 'computational design'. Computational design is typified by a hybrid practice and approach towards computer-based design, a creative melting pot in which the boundaries and definitions between design, art and programming become blurred. A practice which extends the role of design practitioner into the realm of technologist and programmer; combining a technical understanding of programming with a visual critical and experimental approach in "an attempt to provide

designers and artists with a new literacy in digital media". (Watz, 2003). Marius Watz (2005) summarizes this as follows:

Computational design is the discipline of applying computational approaches to design problems, whether related to presentation, analysis or aesthetic expressions.

The term computational design is not a formally recognized term or grouping; it is, however, a useful term used to encompass a range of new media designers, keen to explore the use of code as a means of design, as an alternative to the use of pre-written software. The term will be used in this study as short-hand for those interactive media designers who actively engage with programming as part of their own work and as the key element of their creative practice. Practitioners within the umbrella term of computational design cover a wide range of practice and types of design and artwork. Some practitioners use code to make their own software tools, (Van Rossum & Van Blokland, 2003), or to comment on, or parody, existing ones, for example 'Auto-Illustrator' by Ade Ward (2003). Other designers seek to explore new kinds of code-based visuals and aesthetics or use code as a means of generating illustration or design art making for example Joshua Davis (2009), Eric Natzche (2007). Some develop new forms and expressions of typography (Small, 2009) or interactive work for performance (Levin, 2009). The diversity and range of labels to describe practitioners (e.g. "digital designer and artist" (Nakamura, 2004), "information designer" (Wattenberg, 2004), "artist and technologist" (Davis, 2004) reflect the hybrid, experimental nature of the work, as well as a practice and involvement with computation which spans a wide range of visual art and design disciplines, e.g. typography, graphic design and drawing. Distinctions between art or design are not considered important, or even considered irrelevant by designers, who are rarely dogmatic about such labels: it is enough to recognize that this work naturally sits somewhere between the two. As Nakamura has said:

Whether or not something is viewed as design or art is just a result, so I'm not really concerned with it... its category doesn't matter... I think art and design are melding.

(Nakamura & Fitzpatrick, 2008, p.48)

Despite the diversity of practice, and although 'computational design' is not a formally recognized or unified group, the aim here is to highlight that there are common themes between this group of designers. Computational designers can be seen to share a common sense of ethos, based around a concern to use and write code as a central part of their practice.

2.2.1 Ethos

John Maeda has written extensively about the use of computer code in design, and the rationale and ethos behind it; his work and writing provide an important and influential point of reference in this study. As a researcher and educator, Maeda's thoughts and ideas have permeated through this genre of design, both via educational programmes at MIT (which has produced many graduates now eminent in this area: Golan Levin, Ben Fry, etc.) and from a wide sphere of influence as a designer. In this way Maeda provides an important touchstone from which many of these ideas have emerged and generated authority. Yugop Nakamura, for example, cites Maeda's ideas as a key influence for his work, and Maeda's writings provide an important starting point for the consideration of the ideas and ethos of computational design (Nakamura & Fitzpatrick, 2008).

The research will discuss three key themes relevant to the ethos of computation design: recognition of the distinctness and uniqueness of the medium; freedom of work; and the 'handmade' approach which embodies the core attribute of the 'computational' ethos.

i. Distinctness / uniqueness of medium

A fundamental point which provides a central pillar in Maeda's rationale is a recognition of the distinctness and the uniqueness of the computer environment as a distinct, reactive, medium, and the subsequent desire to design in a way which understands and harnesses the potential of the medium. Maeda, and others, questioned "software dependency" (Watz, 2003) and the "unquestioned hegemony" (Fiell, 2003, p.352) of pre-written software tools to replicate 'old' media in a new environment, in favour of engaging directly with the computer as programmable, reactive medium, with its own set of unique set potential capabilities. Maeda (2000) expressed the idea that the natural means of doing this is to work with programming code:

... when moving on to the world of digital expression, however, the most natural means is not pencil or paper, but, rather computation.

Maeda therefore emphasized a requirement for digital designers to use code as a fundamental means of working with the computer medium, as a way of encouraging a freedom of expression, unhindered by the deterministic ways of thinking set by ready made software tools. Computational designers therefore highlighted the necessity of programming as a means by which the true potential of the computer as a medium of expression could be realized, seeing it as a means to "reclaim computation as a personal medium of expression" (Fiell, 2003, p.352).

If the computer was a new, reactive medium for Maeda and others, code was the most direct way of working with and thinking about this new medium. In this way the decision to use code when working with the computer became an almost ethical or moral decision:

I do think it's essential that an artist-of-the-computer-medium be able to program in some way... it is the only way in which new computer experiences can be made.

(Levin et al, 2001, p.174)

Programming therefore represented a means which allows "direct manipulation" of the "true potential" (Maeda, 1999, p.21) of the computer media, tapping into the essence of the computer medium:

Programming puts you in touch with what software is really made of. That's the computer medium, not Photoshop files or raytraced animations, in my opinion.

(Levin, 2000)

ii. Freedom

The second feature of the computational ethos relates to the idea of freedom. The idea of using code, to 'speak the language' of the computer is seen to represent a kind creative and conceptual "liberation" (Levin, 2006) from the constraints of the medium, breaking the boundaries of commercial software (Simon, 2004) allowing designers to re-think their relationship with the computer. This idea of creative and conceptual freedom is fundamental to computational design, and distinguishes the use of code from a software-driven means of computer design. It is an idea which is central to Maeda's work and remains an important ideological rationale for computational designers which is reflected by the work and attitude of current computational designers. Jürg Lehni (2007) creator of Scriptographer describes "the ability to write programs ... as a freedom in the way computers are used", whilst designers of the LettError project describe the creative freedom enabled by writing code rather to move beyond the "tool horizon" (Middendorp, 2000) of pre-written software. There is, therefore, a

strong sense of the creative freedom associated with the process of designers writing their own code: "creative power comes from writing the code ... not from using it" (Crow, 2008).

Writing your own tools makes the ideas direct the development of software, rather than the other way round.

(Blokland, 2007, p.170)

iii. Making (handmaking and experimenting)

The third feature of the computational ethos relates to the way in which computational design embodies a fundamentally different ethos, or 'spirit' of engagement with the computer. Rather than think of themselves as users of the computer tools, computational designers emphasize their role as creators of the digital media; stressing the process of making and creating rather than editing and re-mixing. Computational design therefore represents a move from computer user to maker.

Computers are capable of a far greater number of things than any specific piece of software might lead us to believe... I'd like to offer a glimpse of what can be achieved when artists step away from Photoshop altogether and make their own software tools - with code.

(Levin, 2001, p.66)

This fundamental change of perspective changes interaction with the computer from tool to medium, and informs a culture of experimentation and making. The ethos of making, of manipulating the digital material, is reflected in attitudes towards the process of using programming: i.e. recognizing the value of working with programming to create work from scratch. Jared

Tarbell (Brown University, 2008) describes being intrigued by the idea of making from scratch:

... with software anything that can be imagined can be built...
Programming truly is a process of creating something from
nothing.

(Tarbell, 2007, p.158)

It is also reflected in the value placed on the final piece of work and the ability of programming to create individual, unique pieces of work which express the singular vision of the creator.

An important dimension is the quality and degree of craft that is applied to the creation of the work ... we'd like to believe that nobody else could have created it with the particular character and texture we did.

(Levin, 2007 p.512)

Interviews with, and the work of, computational designers re-iterates the idea of creating, the joy of discovering, pushing boundaries, creating from nothing, linking concept and practice. This ethos embodies an enthusiastic joy of the process of exploring, of crafting, the computational material.

This act of code writing itself is very important, regardless of what this code actually does at the end.

(Manovich, 2002a)

The use of programming becomes a way of hand-making, from scratch, unique objects, expressing the capabilities and the material, of the reactive environment. This is a type of honesty, embodying what Yugop Nakamura (2008) describes as the 'spirit of craft'.

2.2.2 Materials and Technologies

To talk about 'material' in this context, applying reference of a physical, traditional presence to the symbolic, abstract realm of computing is not, perhaps, an obvious step to take. The idea of applying terminology which reflects the traditional physical process to the abstract de-material of the computer is not, however, without its precedence. Malcolm McCullough's 'Abstracting Craft' (1998) is a study of how ideas of craftsmanship apply to making in the digital realm, and offers a broader view of computer practice within a craft ethos. This craft-centered understanding of computer practice provides an important point of reference for this discussion. McCullough sheds light on the wider practice of engaging with the computer on a craft-like level, and draws attention to the idea that the lack of physicality of the computer need not prevent notions of it having a 'material quality' being considered. He argues that the computer can be seen as a type of medium, i.e. something which mediates engagement and action for personal expression and form, and which therefore providing a range of means and tools for active, creative engagement and mental flow (McCullough, 1998, p.198) of craft-like work.

It is important to note that McCullough's definition of the computer as medium is wide ranging, and accounts for an entire class of 'affordances' which include tools and raw materials. Within the realm of computer material, McCullough defines software as a tool within the 'class' of the computer medium (McCullough, 1998). If the computer is a medium, a broad class, which includes software as the tools (a sub class), in light of the fact that this discussion concerns the specific use of code as a particular means of engaging with the computer medium, we therefore need to consider the status which can be applied to code. Code must be considered as a special case, different from software.

As has been discussed, practitioners of code-centred design make a clear distinction between the use of software, with its pre-written forms, and the

'directness' and 'freedom' afforded by the use of code. Directness of manipulation, directness of understanding of the computer are ideas which are seen as defining code as a different mode of working to that of software. In this way code may be thought of less as a tool and more as a raw material of the computer medium: i.e. something which begins in an unformed, 'raw', state, rather than a tool which has an in-built mental model, a specific means of engagement. For computational designers, code takes on the nature of a material which can be sculpted into forms which are more directly related to the thoughts and wishes of the maker, rather than being forced into modes of work 'dictated' to by the software tool. This idea is one which is certainly supported by practitioners of art and design-centred programmers.

It can be useful to think of each programming language as a material with unique affordances and constraints... in the same way that the different woods Pine and Oak "feel" and "look" different.

(Reas, 2003, p.175)

The term 'material' which conjures up notions of physicality to describe non-physical use of code, is one which may seem forced, but taking McCullough's view that virtuality may not be an obstacle to thinking in this way allows us to take a broader view of the subject and bring into consideration mental engagement and use of the code material. Discussing the use of the computer environment as a material for creative engagement, McCullough stresses the importance of mental models and mental engagement with the material. The idea of 'willing suspension of disbelief' is one which McCullough applies to the active engagement of computer tools in a craft-like creative practice.

This is a concept which could equally be applied to the idea of engaging with code as a raw material. The nature of the computer medium means that

notions of 'tool' and of 'materiality' are largely dependant upon the mental models constructed by the user (maker). When considering the idea of non physical code as kind of abstract, conceptual material for the computer environment, it is therefore important to consider the 'mental' attitudes of those designers who practice the use of computation "consider the computer as a medium of expression just as you would canvas or clay" (Flake,1998, p.11).

If code, its structure and syntax, can be considered to be a kind of computational material, having the ability to be manipulated and crafted by designers and artists into computational forms, then what are the characteristics of this material?

Code as Language

Computer code by its nature is a structured, rule-based, command-driven language, and forms the mechanism by which computational artefacts are created. As the language which commands and controls the 'machine' of the computer, programming code operates as a medium between human and computer, and functions in a way which reflects the 'mechanics' of the computer itself.

At its core a programme is a set of instructions that tells the computer precisely what to do... It is a sequence of formatted words and symbols that encodes ideas into a structure that can be interpreted by a machine. Every programming language is a collection of words and symbols (syntax) with a set of rules defining their use. Each language allows people to convert their ideas into code in different ways.

(Reas, 2004, p.44)

Programming code is the written notation from which work is generated. The language of code is the means by which concepts, as human instructions, are translated into machine commands to be executed by the computer. The language is abstract and conceptual, used to describe and define the detail of the data structure and logic.

Code is a notation of an internal structure that the computer is executing, expressing ideas, logic and decisions that operate as an extension of the author's intentions.

(Cox, et al., 2001)

The formal aspects of the language are governed by rules which determine the parameters within which the designer must operate, understand and work. The rules of syntax, commands and language can be self-generated or in-built into the system.

Abstract Language

At its heart, the expression of digital visual artefacts comes from the computational and conceptual description of the visuals. Visuals are 'described' via the commands and codes of the programming language. The language is not fluent in its description of objects and ideas, it is a functional language. The medium of computation is autographic, it can be read and written, whereas the GUI, point-and-click experience creates computer 'users'. The use of code allows people to become computer makers, (Cramer, 2003, p.100) with the ability to read and write the computational environment.

The ability to "read" a medium means you can access materials and tools created by others. The ability to "write" in a medium means you can generate materials and tools for

others. You must have both to be literate. In computer writing the tools you generate are processes... These processes that simulate and decide are the essence of software and they can only be fully understood through constructing them.

(Kay, 1989, cited by Reas, 2003, p.179)

In this way the use of language becomes a way of structuring and thinking about the computational material. "Beyond being a 'tool' merely 'involved' in shaping thoughts, computer language becomes a way of thinking" (Cramer, 2003, p.103).

The language is used to describe the mathematical logic of a process. An abstract language, code is a notation rather like a musical score, which remains a distinct entity from the artefact itself.

Logical / Numeric Language

A programming language reflects elements of the nature of the computer itself. The computer is a data processing machine and operates on the basis of logic and structure. The programming language describes, assigns and allocates the structure and movement of data within the framework of computer logic. The computational machine is essentially a numeric one, capable of vast calculations at enormous speeds. Computation and computing are therefore largely concerned with the input and output of number and data processing.

Computation operates on numbers, strings, and symbols by manipulating them at discrete time steps... each model has a well defined "program", "input" and "output".

(Flake, 1998, p 27)

Visual representation on screen is therefore a representation of numeric data. Descriptions of shapes, curves, colours, images and entire computer-generated scenes are fundamentally generated as sequences and patterns of numbers. The simple act of making a 'mark' is the product of data, used to set the line's location, colour values, width, etc. There is therefore a natural and close association between numeric data and graphical on-screen representations of images. Physical properties and behaviours are defined as algorithmic sequences and calculations. Detailed mathematical studies regarding number patterns and behaviours extend from geometric descriptions of shapes and lines into physics, biology (e.g. particle physics, rules of behaviour, rules of magnetism, fractals, etc.) and emergent behaviour (Flake, 1998). These areas of study provide the source of inspiration for numerous artistic works, in which behaviours and rules are mapped into the computer environment to create unique pieces of work. Each of these rules is a landscape in itself which commands a vast vista for learning.

2.2.3 Process and Skill

Working with code is therefore a process of understanding and writing the language and notation of the computer program. The practice of programming necessarily requires an engagement with the formal syntax and structure of the programming and is traditionally associated as technical practice; a process summed up by the title 'software engineering'. However the attitude of the designer who uses programming as material imbues the use of code with the status of creative activity which changes its status from engineering to making. The process of working with code as a creative practice is therefore a reflection of the ethos towards code which emphasizes the ideas of computation as the material of the computer environment. The process of computation therefore becomes an outworking of the attitudes and ideas of the material, a creative, experimental, process

of hand-making computational objects. This idea of treating technical abstract code as a creative material is evidenced by the language used to describe the process of working with code in this context. Data is described as something which can be "resculpted" (Gerhardt, & Jarman, 2007, p.392), and numeric algorithms, structures of the language are described as having physical properties: "Manipulating numbers is much like sculpting in clay or mixing paints" (Maeda, 1999, p.69).

Creating with code can be like working with a material, albeit a constantly morphing one. Algorithms have their own material qualities, one might feel liquid, another rigid... you can almost physically sense the dynamic qualities of an algorithm.

(Watz, 2006)

The process of programming therefore becomes an important way of thinking about, understanding, and using the digital material: concepts and ideas are developed through 'sketching' out programmes to explore creative possibilities. The process of coding combines technical and creative practice, generating a dialogue between the language, structure, limitations and rules of code, and the visuals from which an understanding is developed, as both the visual and programming 'rules' are learnt and practised. It is a process of exploring the limitations of the material, through which the computer may be said to reveal itself (McCullough, 1998), i.e. to be understood, both technically and creatively.

When I create art, I feel like I am in a conversation with the artwork... When I write programs, I have the opposite feeling that I am talking with a sympathetic and brilliant partner who helps me organize my thoughts and points out connections I hadn't seen myself.

(Wattenburg, 2007, p.162)

This process reflects a range of creative engagement with the material. Whilst some designers describe "finely crafting the semantics of each program" (Epilog Laser, 2009), others follow a practice of experimentation, exploration and uncertainty in which code is manipulated, pushed and pulled to breaking point as the designers explore the visual, creative, possibilities, looking for "interesting exceptions to the rules" (Simon, 2004, p.46) or unexpected results from randomness or "happy surprises" (Burton, 2007, p.263).

It's only in breaking things - in the anomalies - that I find the accidents that in the end become techniques.

(Davis, 2002, p.7)

The practice of programming becomes a valuable part of the creative process; an "aesthetic experience" (Knuth, cited in Krysa & Sedek, 2006, p.239) of working with, understanding and thinking about the material of code and the medium of the computer. "I tend to be more interested in process than product..." (Burton, 2007, p.263).

The practice of programming, in this context, takes on a more important role than as a means-to-an-end towards the finished piece. Instead it becomes the way in which a better and deeper understanding of the material is developed, and ideas formed. The act of programming is a means of manipulation, understanding and thinking.

A programming language is for thinking of programs, not for expressing programs you've already thought of. It should be a pencil, not a pen.

(Graham, 2003)

Using code becomes a way in which structure is created, limitations are understood and a means of developing an understanding within which the

creative practice can take place. The process of exploring the creative potential of the medium via the 'archaic' notational language of the material of code becomes key to the creative practice of computational design. The practice of coding is a process of understanding the computer material; understanding through doing.

The practice of code as a process of exploration and a means of developing understanding is part of a shared tradition of exploration and development. The newness of the new media has encouraged a community of designers to develop engage and learn through shared practice. A community of designers has developed that expresses, within its practice, an eagerness to share and develop fundamental theories of programming. Theoretical and practical understanding has been underpinned by an 'open source' ideology (Krysa & Sedek, 2006), a desire to encourage the development of code writing through openly sharing ideas and source code. Key figures in the development of the shared tradition of code for creative work include John Maeda, Joshua Davies, Ben Fry and Casey Reas. As an educator and designer John Maeda forged much of the initial exploration and experimentation between graphics and code. His early Design By Numbers (DBN) language was an initial attempt to bridge the gap between design and programming, which outlined an elegant description of the concepts of code as a graphical work centred around a functional programming language created especially for the design community. This early project inspired the continuation of this research via the Computation Aesthetics Group at MIT, where two students (Ben Fry and Casey Reas) developed the DBN idea into the Processing language. The Processing environment exists as a 'free to use' open source environment for 'computational designers' and fosters a lively community of artists designers and programmers (Fry & Reas, 2009). Many other designers and design communities foster the shared experience of learning and developing work: Jared Tarbell (www.levitated.net) and Joshua Davies (www.praystation.com) for example readily share ideas and

source code of much of their work, encouraging a shared knowledge and understanding of their code.

2.2.4 Object and Artefacts

The computational object is a dynamic visualization of its programming structure and data. It is therefore a fluid object, subject to the changes in data; either generated internally, or generated via external data or user input. The object is a dynamic visualization of the programming structure.

When computer programs execute, they are dynamic processes rather than static texts on the screen.

(Reas, 2003, p.176)

Joanna Berzowska describes the characteristics of the computational line and identifies key characteristics as "algorithmic appearance, dynamic change, and behaviour", (Berzowska, 1998, p.9). These characteristics can be summarized as 'algorithm', 'data' and 'behaviour' and mark out the key difference between the static, 'traditional' and the digital, computational object or line. Casey Reas (2003) outlines more specific characteristics of the computational form, and summarizes them as: dynamic form, gesture, behaviour, simulation, self-organization and adaption. All of these categories reinforce the idea of computational object as a dynamic data-based behaviour which is time-based and reactive. Although computational structures are linguistically and logically rigid, unlike traditional media, the computational form is fluid and changeable, and unlike time-based media, (e.g. animation) the forms are not predictable and endlessly repeatable. Computational forms are reactive, behavioural and fluid. Programming algorithms provide the structure and the parameters, defining the data structure for the fluid, computational object to emerge.

Computational forms are therefore highly influenced by natural, biological references, both in terms of behaviour and aesthetic. There is a natural link between physical, botanical properties and computational objects and behaviours. Computation and biological behaviour are closely linked (Flake, 1998), and so biology and organicness therefore influence the work of a great many computational designers. Organic concepts and behaviour (e.g. randomness, repetition pattern, dynamic form) are often used to link with natural forms and life with the computational object.

Flash artists are big on biological references. Abstract plants, minimalist creatures, or simply clouds of pixels dance in patterns which to a human eye signal "life".

(Manovich, 2002a)

Work produced by computational designers and artists reflect the concern with organic behaviour (Hodgin, 2010), data visualizations (fig. 2.1), and natural forms (fig. 2.2).



Figure 2.1 Screenshot of 'Anemone' by Ben Fry accessed from: <http://www.benfry.com/anemone/>

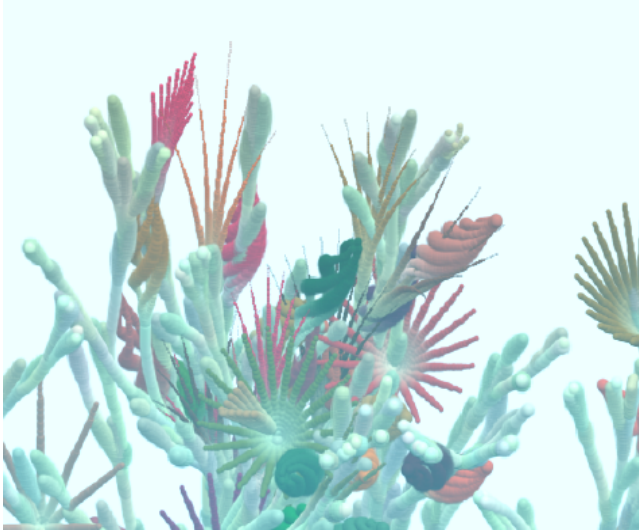


Figure 2.2 Screenshot of 'Flowers' by Daniel Brown accessed from: <http://www.play-create.com/>

Emphasis on the behaviour of the object leads to a simplification and abstraction of the aesthetic; shapes and forms are fluid, organic, and behavioural - they are often abstract, and visually 'light'. The examples of work by Yugo Nakamura (fig. 2.3) and Joshua Davis (fig. 2.4) typify this abstract 'light' aesthetic.

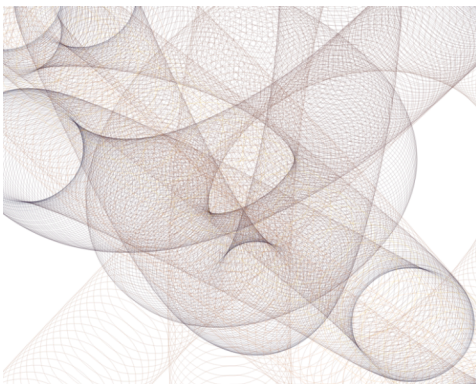


Figure 2.3 Screenshot of 'Oval x3' by Yugo Nakamura accessed from: <http://www.yugop.com>



Figure 2.4 Screenshot of 'ps3_1_praystation' by Joshua Davis accessed from: <http://www.prystation.com/>

Lev Manovich (2004) draws comparisons between the abstract, geometrical aesthetic of some programmed forms created by Flash artists with a modernist aesthetic, calling it a type of "soft modernism" (Manovich, 2002a).

2.2.5 Summary

This section of the work has provided a useful outline of some of the key features of computational design; identifying characteristics of ethos, material, process and object. The research has identified computational designers as a 'new breed' of designer, keen to explore the creative potential of code within the ethos of computational design the particular characteristics of 'distinctness', 'freedom' and 'making' have been identified. Although not a formal grouping, computational designers share an ethos which recognizes the uniqueness of the computation as a means of engaging the computer as a expressive medium, away from the limitations of pre-written software. Highlighting a sense of creative freedom offered by the process of making work 'from scratch', computational designers describe the process of writing and creating with code as others may with traditional,

physical material. The creative approach to the abstract, logical, rule-based language of code challenges traditional notions of the process of working with code as software 'engineering', suggesting an alternative view of the process as 'making'. The object of computational design, a visualization of its underlying data structure, is differentiated from other design work by its dynamic, generative and reactive nature. Computation objects possess a fluidity and behavioural quality often associated with natural organic environments and which contrast with the rule-based, logical structure from which they are created. Computational design can therefore be identified as an area which encompasses a distinct group of designers keen to challenge the 'limitations' of software, who are motivated by ideas which reflect the values of traditional design practice in the use and exploration of computation as a kind of material.

2.3 Traditional Material: The Arts and Crafts Movement

Having identified the ethos and approach of computational designers towards the process of using code as a kind of material, it is now necessary to include a detailed look at traditional design processes, using the Arts and Crafts movement as a particular case-study. The Arts and Crafts movement has been chosen because it encapsulates core values of traditional design. By considering the ethos, material, process and artefact of the Arts and Crafts processes, a broader perspective upon the discussion can be taken, allowing comparisons between traditional, digital and computational design work.

The Arts and Crafts movement of the late nineteenth century had at its heart the relationship between maker, designer and material. Characterized by strong moral views and strict adherence to a 'right' and 'wrong' idea of beauty, the Arts and Crafts movement made important contributions towards the understanding of design and craft. The main emphasis of the Arts and

Crafts movement centred around a resurgence of a craft-like sensibility, and the thoughts and ideals of this movement fed into the wider discussion relating to material and practice, which are important for this research. This section will begin with a general introduction to craft practice.

At the heart of a definition of any craft practice lies the idea of the applied, skilled understanding and mastery of material (McCullough, 1998, p.22). Regardless of medium, the craftsperson must demonstrate an understanding, a mastery, of their will upon their chosen material. The musician, painter, sculptor or writer must, in order to communicate effectively, understand the essence of their material, its structure, its parameters, its pliability. It is the demonstration of this understanding which has traditionally been equated as the great skill of the craftsman. McCullough (1998, p.201) outlines the idea of craft as personal knowledge plus practical intent: "where the medium is the basis for mastery: there we find craft".

The implication here is that this mastery is gained from a theoretical understanding ('textbook' knowledge), plus practical experience ('tactile' knowledge). The practice of craft is, therefore, not an entirely physical process, but encompasses both the physical and thought process of the creator upon material (McCullough, 1998, p.29). The practice of craft unifies head and hand, thought and action. Manipulating, 'crafting', the material is a balance between the physical forces of hand-work and invisible forces of intellect and understanding. To craft a material is to be involved in a reflective, thoughtful process, a constant dialogue between artist and material. This idea of craft being a balance between the visible and invisible forces, a unity between the material, environment and creator, is one which was central to the Arts and Crafts movement. The movement provides a valuable context for the evaluation of the relationship between designer, material and artefact. This discussion will not give full consideration to the

whole of the social and political elements of the movement, however, key ideas relevant to this discussion will be outlined and discussed.

2.3.1 Ethos

The Arts and Crafts movement was not only characterized by the style of the objects it designed and made, but also by the guiding principles which drove the movement and its members. Founded as a reaction against the industrialization of creative practice, the movement was driven forward by ideological principles which informed the nature and style of the work. Ideals and principles, particularly associated with John Ruskin and William Morris, were at the very heart of the practical work. Key to the principles of the movement was the idea of unity (Cumming, 1991, p.7). Unity between the human, environmental and material forces, between material, process and object was the foundation stone upon which the movement stood. This idea became influential in the wider development and understanding of the process of craft.

For Ruskin and Morris, the crafted object was never considered in isolation, its wider significance and value was calculated in terms of the surrounding forces that helped shape it. In this way, the process of craft transcended physical work and became an invisible moral and intellectual engagement with the material. The significance of craft had as much to do with the invisible approaches and attitudes of the craftsman as the beauty of the finished piece itself (Boe, 1977, p.107). As well as valuing aesthetics, the movement emphatically positioned the value of a piece of design according to its expression of underlying, 'invisible', forces involved in its creation. i.e. how it was created, by whom, for what purpose, and under what circumstances. For Ruskin and Morris, the objective of value was to strive for completeness, unity and harmony, within the work (Cumming, 1991, p.6). The views, and ideology of the Arts and Crafts movement are therefore

reflected in their attitude towards the process the material and the artefact of their work.

2.3.2 Material and Technologies

The use, choice, understanding and relationship with material is a central issue to an understanding of craft, an "act of appreciation" of the material (McCullough, 2008, p.203). The process and notion of craft can therefore be defined as a dialogue between the creator and material, this idea was a central theme of the works and writings of the Arts and Crafts movement.

Using 'nature' and 'history' as key foundation stones, the Arts and Crafts movement, typified by a desire to use and honour traditional approaches of design, involved themselves in traditional practices, working with the solid, stable, traditional material of craftsmen and artisans. The work therefore reflected an embracing of traditional craft practice, as an ideological stance against the machine and industrial process and practice. The selection and use of material was not neutral; the choice and use of material was highly significant, more than just an aesthetic selection, and a great amount of emphasis was placed on understanding, seeing and 'honouring' the intrinsic characteristics of the material. The writings of Morris in particular, emphasized the idea of the primacy of material.

Try to get the most out of your material, but always in such a way as honours it most. Not only should it be obvious what your material is, but something should be done with it which is specially natural to it, some thing that could not be done with any other.

(Morris, 1881, cited by Boe, 1977, p.69)

As a way of expressing the unity of the relationship between maker and material, each type of material had to be used in an appropriate fashion, so that the final piece of work would reflect the specific qualities of the material. Morris stressed this notion of unity saying that the inherent value of the material must be expressed in the piece of work. There must be unity between the object and its material.

You may conquer the obstinacy of your material and make it obey you as far as the needs of beauty go, ... be pleased with the victory of your skill, ... and you will know that your victory has been no barren one, but has produced a beautiful thing, which nothing but your struggle with difficulties could have brought forth.

(Morris, 1884)

Acknowledgement of the limitations of each type of material were seen as opportunities for the craftsman to demonstrate understanding of, and unity with, his material. Indeed the physical and structural limitations of the material were readily acknowledged, serving as rigid boundaries within which the artist was to work. 'Making the best of' or 'working with' the rules and constraints served as a test of the true craftsman.

As to the limitations that arise from the material we may be working in, we must remember that all material offers certain difficulties to be overcome, and certain facilities to be made the most of.

(Morris, 1882)

The maker was, therefore, never to disregard, break or bend, the rules of the structure of the material so as to make something 'unnatural'. The maker was, instead encouraged to work with and 'in sympathy' to, the nature and

character of the material. Caution was advised not to make too much of the skill of the craftsman over and above the characteristics of the material itself:

You must be the master of your material, but you must never be so much the master as to turn it surly, so to say...you are no longer an artist, but a juggler.

(Morris, 1882)

The choice of material was also considered important as a reflection of the natural world and the surrounding environment. Material chosen for work had to be appropriate to the surrounding landscape. In this way, use and understanding material became aligned with 'moral' values of goodness, beauty and truth and, impacted upon the value of the work.

2.3.3 Process and Skill

The Arts and Crafts movement, rather than looking toward the future for advancement of 'industrial design' was highly interested in the practitioners of traditional processes. Just as the material reflected the ideals of traditional crafts, so also the processes reflected those of traditional hand craftsmanship. History and tradition formed a strong ideological foundation for the use and understanding of craft processes. The strong sense of tradition and history which pervaded the Arts and Crafts movement encouraged notions of 'correct' practice, i.e. that the methods, and practices of manipulating any material was not left to individual decisions, but based on the unchanged, processes, practiced developed via the tradition of craft over generations. Morris spent a lot of time and detail outlining specific processes and how they should be applied to each specific material: there was little room for experimentation, clear judgements about correct and incorrect methods were outlined. Seeing the process as a part of this wider

historic tradition emphasises the ideas of learnt, and practiced skill and control, for the correct uses of material.

The special limitations of the material should be a pleasure to you, not a hindrance... it is the pleasure in understanding the capabilities of a special material, and using them for suggesting (not imitating) natural beauty and incident, that gives the *raison d'être* of decorative art.

(Morris, 1893)

Craft process was therefore to demonstrate practiced skill and understanding of material. As we have already discussed, the Arts and Crafts movement placed importance upon the mental attitude of the designer towards the material, (Boe, 1977, p.107), as one of the invisible forces of the practice. The skilled artist or craftsmen was to understand and to use the material in its appropriate fashion, and this understanding was to be practically demonstrated in the skilled mastery of material.

Emphasis on understanding the limitations and affordances of material, the Arts and Crafts movement therefore uses the process as a means of learning and exhibiting control over the material; a means by which both the skill of the artist and the intrinsic values, limitations and characteristics of the medium are exhibited. The process itself therefore becomes an important part of the development of understanding, moving from theoretical knowledge to practiced, skilled understanding. Each material has its own processes which must be employed to make the 'best kind' of form for each type of work. In this way the process is a practical outworking of the theoretical understanding; a unity between head and hand with a strong emphasis on skill and control. The notion of 'correct' practice was not, therefore, limited to correct methods of physically manipulating a material,

but extended to the correct ethos surrounding the practice: a morality which included the process as moral fulfilment of the practice itself.

Demonstrating a holistic, moral, approach to the processes and practice of craft, Morris therefore rejected the mechanical, 'dehumanizing', processes of industrial design in favour of human-centred processes and practices of traditional craft. Key to this were ideas of humanity and joy, and whereas the industrial model enforced a division of labour, in which workers were involved only in a single part of the process, part of the 'machine', the craft-centred process extolled ideas of individuality, wholeness, humanity and joy, in which the worker was involved in the entire process. 'Humanity' was expressed as joy in making, William Morris (1882) stated that the process and practice of craft should be "a joy for the maker and the user", a moral dimension of work which provides "happiness to both maker and user" (Boe, 1977, citing Morris, 1879, p.116). The processes and practices of industrial design were therefore rejected by the followers of the Arts and Crafts movement not only from an aesthetic standpoint, i.e. rejecting the quality of the resultant object, but also from a moral standpoint. Process and 'workmanship' for Morris was not merely a way of creating products but an end in itself. The human experience of joy and honour from labour was something which was lost by industrialised mechanical processes, and which, it was argued, could only be gained from traditional craft practice. Morris emphasised the idea that the process of work should reflect individual human endeavour and imperfection, rather than 'mechanical perfection'. The humanity of the object, i.e. what it represented in terms of human endeavour, and labour, became bound up with the moral and aesthetic value of a piece of work. Absolute geometrical perfection was dismissed and mistrusted in favour of the imperfection of the handmade object (Cumming, 1991, citing Ruskin, 1849, p.12). Perfection, said Ruskin, was the reserve of God alone (Cumming, 1991, p.14). The practice of craft can therefore be seen as a unification of the head and hand, the thought and actions of the creator upon

a given material (McCullough, 1998, p.29). The final object thus gained significance as a manifestation of the mental and physical engagement of the craftsman, a visualization of the balance between the three core themes of the movement: material, humanity and environment.

2.3.4 Object and Artefact

The resultant object of the Arts and Craft movement highlighted attitudes promoting the traditional, 'human' process and emphasis on the importance, choice and use of material. The designed object was more than the final piece of work and instead stood as a symbol of the wider social, environmental and moral forces that formed it; expressing the beauty of natural form, honour to material and reflecting the process and workmanship of artist. For Ruskin and Morris the crafted object was never to be considered in isolation, its wider significance and value was calculated in terms of the surrounding forces that helped shape it. The value of the crafted artefact had as much to do with the invisible approaches and attitudes of the craftsman as the beauty of the finished piece itself (Boe, 1977, p.107). The final object thus gained significance as a manifestation of the mental and physical engagement of the craftsman; a product of the balance between the three key themes of material, humanity and environment.

The finished crafted object bears the visible hand marks of the craft process and the less explicit 'marks' of the mental process of its creator, thus becoming a unification of the physical and mental processes that formed it: a result of both the physical (visible) and 'mental' (invisible) forces acting upon the unformed material. Through seeing and touching, the viewer may be able to glimpse the practiced mental and physical skill involved in its creation, and the skilled understanding of the artist. The crafted artefact reflected ideas of pleasure; pleasure from work and pleasure from use:

"Sound ornament was, for both [Ruskin and Morris] ... the sign and guarantee of the spiritual dimension of human work" (Fuller, 1988, p.127).

Another facet of the Arts and Crafts movement ethos was its environmental concern, which expressed itself in use of material which has 'sympathy' for its surroundings, as well as an emphasis placed on nature as inspiration for work. The Arts and Crafts movement emphasised the idea that the final object should demonstrate links with environment both in terms of the material and form of the object. Ruskin gives clear examples of the importance of unity between object and environment. In the 'Nature of the Gothic', Ruskin (1977 [1853], p.11) speaks of the "brotherhood between cathedral and Alp". Ruskin's emphasis on the natural form is highlighted by Boe (1979, citing Ruskin, 1849, p.88) in which Ruskin is quoted to espouse the 'nobility' of natural form, even to the point that "forms which are not taken from natural objects must be ugly".

2.3.5 Summary

Informed by a strong moral and ethical approach to craft and traditional practice; the Arts and Crafts movement can be summarized by ideas which highlight the unity within the process, the primacy of material and the value of workmanship, reacting against the 'mechanical perfection' seen to be at odds with nature. The movement placed significance on the process of making as a highly valued expression of unity between worker, material and environment. Placing emphasis on the skilled understanding and use of material, the movement encouraged an approach to practice in which the maker was to 'honour' his material; i.e. to create work which makes the best expression of the chosen material without 'twisting' it beyond its natural means. Importance was placed on the final designed artefact, not as an object considered in isolation, but as an expression of true workmanship. Although, on reflection, the movement can be criticised for its dogmatic

moral and absolutist approach to 'good' and 'bad' design, i.e. design that expressed aesthetic and moral values, the Arts and Crafts movement has become an important foundation for later design movements, and an important case-study for this thesis. The approach towards design which emphasises the importance of the process, skilled use and understanding of the material, and unity between process, material and object are all core elements of the Arts and Crafts movement which form its design legacy and influence.

2.4 Summary of Contextual Study

Having highlighted the different approaches, and processes including both digital and traditional design, the research is now able to consider where computational design 'sits' with reference to the ethos, materials, processes and objects of both digital and arts and crafts design. By taking an overarching view of the contextual study, key themes can be seen to emerge which highlight areas of overlap between computation and the areas of digital and traditional design.

The shared digital environment which both software-based design and computational design inhabit, give these areas some natural and obvious areas of commonality. Closer examination, however, of the 'invisible' ideas, attitudes, motivations and approaches towards the process and material of design reveal areas of commonality between computation and craft; a commonality of ethos which influences and directs subsequent approaches towards the process and material of work. The following diagram offers a summary of the shared themes of each element of design which have emerged from this section of the study. The lighter shaded areas highlight the key distinctive elements of each area, whilst areas of commonality between computation, arts and crafts and digital design are darkly shaded.

Arts and Crafts	> <	Computation	> <	Software-centred
traditional material physical process	primacy of material handmade ethos culture of making original object uncertainty and randomness	logical rule syntax	digital environment fluid object	no clear ethos technology driven re-mix aesthetic import/ export process

Table 2:1 A table outlining where computational design intersects with Arts and Crafts and software centred design.

The table highlights some key areas of commonality and difference in relation to computational and software-driven design, and offers some notable intersections between the Arts and Crafts movement and computational areas. Whilst both software-centred and computational design share characteristics which reflect the common characteristics of the virtual digital universe, the logic and syntax of the rule-based structure of the computational environment marks an area of clear distinction from the processes and ideals of the re-mix world of software-driven design.

Despite obvious differences between the material and physical process of the Arts and Crafts movement and computational design, the diagram highlights interesting thematic intersections between these two areas. A common ethos based on the primacy of material, the value of 'hand-making', and a emphasis on uncertainty within the process gives rise to a shared set of values which informs both processes and transcends physical differences between these areas.

Core attitudes and ideas of the Arts and Crafts movement which place great emphasis on the significance and importance of the material and the process of work are reflected by the mental engagement of computational designers, and their keenness to use, understand and be 'true' to their computational material. The desire to move beyond the perceived limitations and

boundaries of point-and-click software, towards a process of handmade, individual tools is driven by the idea of the potential of the computer as an new medium in its own right. Dissatisfaction with the limitations of digital media can be seen to echo the reactionary attitude of William Morris against mechanization. David Reinfurt (2005, p.2) observes that Muriel Cooper's "critical relationship" towards the tools and materials and the expression of her desire to confront the "mundanity" of software tools "sounds as if it were written by William Morris in 1872". In the introduction to *Design By Numbers* (Maeda 1999) a similar connection is made toward the rigorous approach of John Maeda his work.

The most important part of Maeda's production ... is not the final object, but rather the process. In his work, the process is the core that informs the final outcome. Maeda's fundamental idea is that to successfully design with a computer, one has to design, or at least understand, the program one uses.

(Antonelli, 1999, p.10)

Computational designers highlight a desire to use, understand and explore the potential of the computer; to hack, play and experiment with code to generate individual, 'handmade', pieces of work, emphasizing an ethos in which the process of creating forms a vital part of understanding and bending the rules of code. The process of working with code can therefore be seen as a means of understanding and 'mastering' the computer as a medium. Just as the ethos of the Arts and Crafts movement informed their work, so the recognition that the computer can be approached as a kind of material reflects and informs the process and approach of computational designers work. Attitudes expressed explicitly and implicitly through the work of computational designers such as Marius Watz, Daniel Brown and Casey Reas emphasise the formal aspects of code as a type of material, and their

need to master the medium in order to be fluent and expressive with it. Computational design can therefore be seen to represent an interesting combination of attitudes from Arts and Crafts and digital design.

The inter-relationship of computational design, digital design and Arts and Crafts highlights the way in which, despite the shared environment of software and computation, the approach and attitude of computational design towards the process, practice and material of computing 'as material' highlights important linkages with the those of the Arts and Crafts movement. Although computational and digital design share a common environment in which digital objects are produced, the core ideas of skill, process, practice, understanding and 'hand making' are characteristics which underpin and motivate computational work, and distinguish it from software-centred design.

2.4.1 Computation as Craft?

The consideration of computation in the light of the Arts and Crafts movement opens up interesting potential new ways of understanding the computational environment. Do the links with ideas regarding the approach ethos to the traditional approach of the Arts and Crafts mean that practice of computation can be considered more broadly alongside traditional craft practice? Does the practice of computation place emphasize the process of engaging with material in a way in which is not reflected by software-centred design practice? The lack of the physical or tactile elements of computational work presents some obvious problems to this idea, however in recent times, and in the light of new technologies, the literature of traditional craft has been extended to include non-physical, digital objects and processes.

In her PhD research, Jane Harris (2000) explores the use of digital technology in relation to practice of textile design. Her thesis explores the linkages between digital environment and traditional making, establishing

links which move the idea of craft into the digital arena, challenging traditional roles of craft 'making'. Links are made between the digital and physical process which emphasize the ideas of craft as process of challenging and "understanding the strengths and weakness of material" (p.66), whether digital or physical and extending the process of craft as a sense of "enquiry into the potential of chosen media" (p.73). Her work highlights the close mental approaches between digital and traditional processes.

The two processes are "physically" very different experiences, but mentally there are similarities.

(Harris, 2000, p.85)

Similarly Hillary Carlisle (2007) explores the practice of programming as craft within context of textile design, specifically the use of C++ programming to create non repeating patterns and exploring the use of code as a "new craft practice".

The examination of craft within the broader context of digital technology is also the subject of broad debate and discussion within the craft community. The Challenging Craft (2004) conference run by Gray's School of Art presented a number of papers were which proposed conceptual notions of craft when applied to the digital realm. An example being Gilbert Riedelbauch 'Craft and New Technologies' (2004), a paper which explores the creative integration of rapid-prototyping into the realm of craft practice.

Although the focus of this study is on the specific use of computation, rather than other digital software processes, viewing craft within this broader context provides a useful background to the discussion regarding the use of computation in relationship with creative craft practice. Being able to take a more conceptual view of craft introduces a broader dimension to the discussion, enabling the process of computation to be more firmly linked with

craft, allowing ideas of craftsmanship to be applied to the process of computation.

Using an understanding of craft which emphasises a practiced investigation, skilled understanding and mental engagement of a material whether physical or digital (Harris, 2000) brings computation more clearly in line with craft, allowing greater comparisons between computation and craft to be made. The emphasis on process, material, understanding and handmade objects of the computational design can therefore be aligned to a broader, conceptual understanding of craft practice. An overlap becomes identifiable between the ideas of computational designers who take a real interest in their medium, with the practiced investigation of material of the craft process. This is a strong theme which runs through the work and ideas of computational design and designers. Several designers make these connections clear when commenting upon their work. David Crow (2008) outlines a 'new' interest in craft highlighted by the exploratory nature of designers within the realm of new digital material. The designer Yugo Nakamura outlines the influence of traditional craft upon his own work using code highlighting ideas of the 'spirit of craft' in the work (Nakamura and Fitzpatrick, 2008). Whilst Marius Watz reinforces the craft approach natural to the computational process:

... I think many artists (and designers) who work with digital media are rediscovering the issue of craft, both in terms of attention to detail and as a way to get closer to the medium. That is certainly true for artists working with computational media

(Watz, 2006)

Jared Tarbell outlines his own engagement with the process of programming as a creative and conceptual "deep trance" (Tarbell, 2007, p.158), which

engenders the concept of 'flow' cited by McCullough (1998, p.198) as a craft-like engagement with a material which offers a "sense of continuum of possibilities". This idea of engagement with the process of computation as 'flow' is reinforced by McLean and Wiggins' (2010, p.9) own observations: "Speaking anecdotally, programmers report losing hours as they get 'in the flow' when writing software".

The idea of computation as craft has also recently found expression via a number of high-profile curated exhibitions most notably the British Council's My World: New Subjectivity in Design (2006), the Lovebytes 'Code:Craft' show (2010), and the Victoria and Albert exhibition Decode: Digital Design Sensations (2010). Each of these shows make and explore links between the ideas of materials and making in the context of digital and computational work, featuring the work of well established names from the computational design arena: Daniel Brown, Golan Levin, Casey Reas. The following text taken from the My World exhibition features the work of Daniel Brown and makes the link between his digital, computational work and craft:

Digital technology is one of the most intriguing and unexplained places we find a new kind of craft. It is clear that craft in many senses – ingenuity, brilliant technical manipulation, deliberate and unique personal expression – does exist. The practitioners that manipulate pixels with the most affecting results the are the ones that have craft.

(British Council Arts, 2005)

The idea of, and attitude towards, material is clearly important to both craft and computational design, and provides the foundational corner stone which both practices share, namely the key themes of making, skill, mastery, understanding of material and the 'handmade'. The computational ideas of

'handmade' play, process and understanding can therefore be seen to reflect the ideals of craft.

2.4.2 Contextual Review in Relation to Practical Project

The focus of this research is on the investigation of the practice of using code as part of creative design, viewed from the perspective of a practitioner moving into the realm of computation for the first time. The focus and investigation of the work therefore centres around the process and development of practical work. Although the linear nature of this research document suggests otherwise, the development of practice was not started directly after the contextual review, but was begun and developed alongside. Computational sketches, tests and experiments were developed during the contextual study and research phase. Developed understanding and ideas from the contextual review therefore gradually informed the development of initial tests and experiments as the practice was begun. Initial phases of the first project (the Colorcam work, chapter 3) reflect initial attempts at use and understanding of programming. This early stage of work was influenced by early work and research reflected by the contextual study, and in particular the work of John Maeda, and a visual aesthetic of computational drawing based on ideas of recursion. As the contextual review developed more shape and form, clearer links and ideas, which related the practice of computation in relation to the Arts and Crafts movement emerged. This linkage was developed during contextual work, and focused the early simple line drawing experiments towards a defined goal, which connected the practice and contextual research.

The inclusion of the Arts and Crafts movement as a specific 'case study' in the contextual review provides a wider lens through which the context of computational design can be more clearly seen. Links and comparisons between computational and traditional design provide both an important

basis for the practical element of the project, and an aesthetic and conceptual focus for the work. The thematic overlap between traditional and computational design practice highlights commonality between ideas which emphasize the significance of material and value of process. Both traditional and computational design practice can be seen to place significance upon the primacy of 'material', either as tangible, physical material, or as the computational 'material' of code. Core values of traditional Arts and Crafts design is reflected by computational designers expressing a keenness to understand, use and be 'true' to their computational material. This intersection between traditional and computational design, the shared emphasis on the value of process and respect for material, therefore provides important conceptual direction for the practice. Conceptual inspiration and direction for the practical project comes from a personal desire to use and understand the 'essence' of the computational material, to move beyond the software short-cuts and 'effects' and apply an Arts and Crafts attitude which emphasizes the importance of process and material within the work.

The practical project work will therefore be created by direct use and manipulation of code, removing the 'layers' of software usually encountered between the user and the material. Starting with the unformed raw material of code, shapes, objects, lines, forms and colours will be sculpted and generated via individual algorithms and instructions written to shape and sculpt the on-screen visuals. All visual elements will therefore be created 'from scratch' by handwriting the code. 'Processing' will be used as the computational environment for creating the work: a language specifically designed to allow for direct manipulation of code for generating graphics. The requirement to 'hand make' each visual element of the work underlines the idea of working to attain a fundamental understanding of the 'essence', the core characteristics, of the code. The simplicity and directness of the process highlights both the concept at the heart of the practical work and the

value of *process* as a means of using and understanding the 'material'. Emphasis on understanding through making reinforces the conceptual focus of the work, linking it conceptually and aesthetically with values of traditional design, embodied in the Arts and Crafts movement.

Aesthetically, the decorative, botanically inspired work of the Arts and Crafts movement, and in particular the repeating, organic, flowing shape and pattern of Morris' wallpaper design, also provides an important reference and influence for the computational shape form and pattern.



Figure 2.5 William Morris' 'Willow' wallpaper design



Figure 2.6 William Morris' 'Jasmine' wallpaper design

The flowing organic lines, leaves, flowers and stems of Morris' 'Willow' (fig. 2.5) and 'Jasmine' (fig. 2.6) wallpaper designs, in particular, provide a key visual reference for the computational work, as it explores conceptual and practical links with traditional practice. Taking visual influence from William Morris' wallpaper design and applying it to computational work, the practical

project work is summarized by the title 'computational wallpaper'; a title which highlights the overall focus of the practice i.e. one in which organic, fluid lines, shapes and forms of stem, leaf and flower are created as a series of computationally generated pieces of work. The intention of the computational wallpaper work is therefore to re-apply elements of the Arts and Crafts aesthetic, and specifically Morris' 'Jasmine' and 'Willow' designs, into the computational environment. The development of computationally generated, botanical shapes and forms will be used as a means of investigating and understanding the formal aspects of programming code, its syntax and structure.

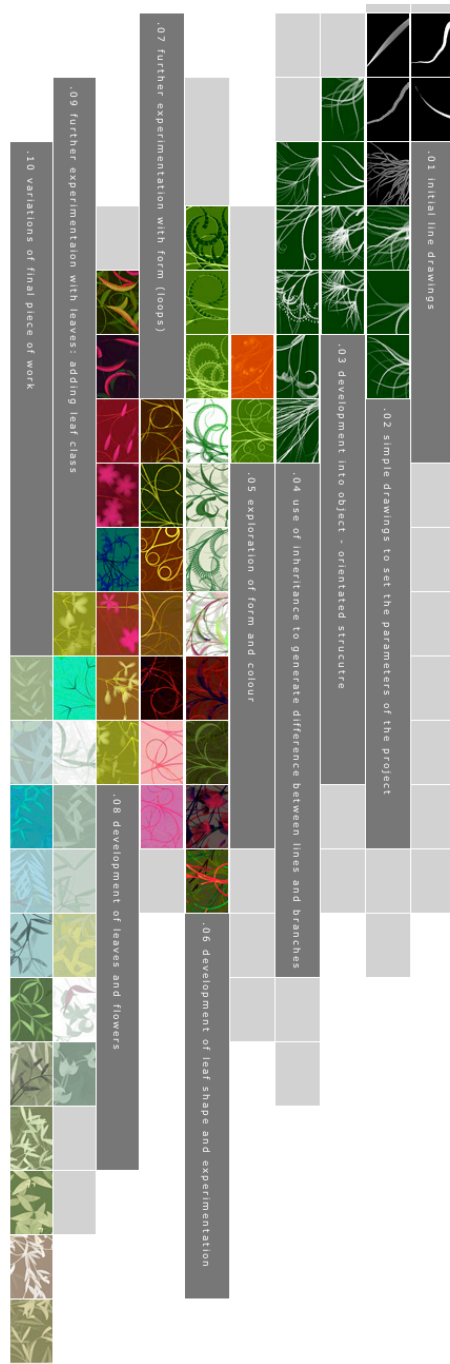
The computational wallpaper project work is therefore a series of computationally generated, screen-based digital decorative shapes and forms. Hand-written programming code is used as the single source 'material' and means for creating the digitally generative pieces. The application of the Morris wallpaper aesthetic into the computational environment; combining the computational hand-made ethos and process of programming with the aesthetic of the Arts and Crafts wallpaper, provides a way in which resonance between code and making can be developed and explored, both conceptually and visually. The computational wallpaper element of the research is divided into two core projects: the 'Colorcalm' and the 'Moving Wallpaper' project. These projects are unified by the same 'computational wallpaper' idea, and provide a framework from which a broad comparison of computational processes can be made. The stages of development for both pieces of project work are mapped out by project diagrams (fig. 3.1 and fig. 4.1) which give an overview of each project. The structure of these diagrams illustrate the way in which projects move from experiments towards final pieces of work.

3 Colorcalm Project

As has been outlined in the previous chapter (2.4.2) the overall intention of the practical project is to create computationally generated shapes and forms, conceptually and aesthetically inspired by the wallpaper design of William Morris. The Colorcalm project represents the first body of work developed as a computationally creative project and is followed by the Moving Wallpaper work, which builds on and develops the work of the Colorcalm project.

The Colorcalm project is defined and unified by the idea of 'computational wallpaper', a concept which runs through both projects, creating work which explores core values of the computational environment, and which seeks to reflect the concept and the aesthetic of the Arts and Crafts movement. Emphasizing the importance of *process* and *material*, flowing botanical shapes and forms, inspired by Morris' 'Willow' pattern and 'Jasmine' designs, are 'crafted' from hand-written code as a process of gaining personal understanding of the computational material. The intention of the Colorcalm work is to reflect the importance of material and process in creating computationally 'crafted' work. This project therefore explores the fundamental elements, the building blocks, of programming code, which define the basic abstract logical structure of the computational material. Individual algorithms, variables and mathematical functions are used as a means of creating and sculpting a computational system to generate shapes, curves and flowing botanical forms which resonate with the shape and pattern of the William Morris aesthetic. Each step of the process is documented to show clear development and thinking, as the impact of each element of computational material is examined and considered at each stage. The systematic nature of the documentation emphasizes the significance and importance of process to the project, as the work explores and expresses core values of computation as material.

Figure 3.1 (next page) shows a visual overview of the development of the Colorcalm project in the form of a screenshot from the web page, available at http://www.random10.com/colorcalm_research. The page provides examples and source code of each piece of work when individual thumbnails are clicked. The web site and source documents are also available on the CD included as appendix 4. The performance of these java applets the will vary according to the browser and operating system upon which they are run. Outlines of the key elements of source code for each of the stages of the work are available in appendix 1.



This diagram is a representation of the stages of work in the colorcalm project. Each development or phase of the project is respresented along each line. The left-to-right span of the diagram is a representation where each element sits with regard to engagement with the computer as machine or as material.

Figure 3.1 An overview diagram of the Colorcalm project available at: http://www.random10.com/colorcalm_research/

3.1 Initial Line Drawings

The first phase of the Colorcalm project can be characterized as the initial 'mark making' exercise. It is the phase in which the first lines are created determining and defining, in computational terms, the fundamental conceptual foundation of the project. Before a line can be created initial decisions have to be made regarding the nature and the concept of a 'line'. Key questions have to be answered which establish the scope and the direction of the work; how the line will be drawn, what key data elements will define the nature and type of line. This phase is therefore the first attempt at answering these questions in order to establish a direction and context for the rest of the work. It is a key phase in which basic material properties are explored and expressed.

3.1.1 Code

The computational set up which creates the marks is defined as being an iterative process of a moving single mark across the page. A single line is plotted at a point on the screen and its position is then re-calculated using a simple algorithm and re-plotted. As the object moves across the screen so it leaves behind a trail giving the illusion of a continuous line being drawn.

Simple numeric variables are used to control the direction of each small point which affects the overall shape of the line, its width and its length. The key data elements in this piece of work are the `timer`, `angle`, `radius`, `x` and `y` variables. Each of these variables form direct associations between the computational concept and the visuals. The core variables, and their impact on the visuals of the line can be summarized as follows:

Variables	Description
x, y	the location of the line on the screen. Values for x and y are the result of calculations using the other variables (width, radius etc.). The values are 'reset' when a 'new' line begins drawing.
timer	a timer variable is used to allow control of the intervals at which the line is drawn. It is a incremented value which marks the passing frames, and used to dictate the speed of the drawing machine. The timer is also used, in some initial tests in the calculation which re-positions the point.
angle	a variable which is used to recalculate the next position of the line, the manipulation of this value is key to altering the shape of the line.
radius	a variable to determine the distance moved by the line in each frame, which affects the speed of growth: A large radius value will re-draw a large section of the line each frame and give the impression of a fast drawing.
w	a value to set the width of the line for each frame. Altering this value during the drawing allows the line to get thicker.
inc	a value to determine the amount by which the angle value is to be recalculated (incremented).
Functions	
lineDraw ()	a function which contains the algorithm to calculate, plot, and re-draw the line.
reset ()	a function to reset all the variable values to begin redrawing the line.

Table 3:1 A list of key variables and functions from the initial line drawing stage: Colorcalm

3.1.2 Visuals

The work at this stage is characterized visually as very simple repetitive marks and lines on a page. A small line moves across the screen leaving behind a 'trail' which forms the entire line (fig. 3.2). This process is repeated to create a number of different lines (fig. 3.3). The effect of the line-trail, combined with intermittently layering of a transparent colour over the top of the screen allows the movement of a single object to give the impression of many lines being drawn on top of one another giving a multi-layered appearance to the work (fig. 3.4). Once the piece is set up and running,

there is little variation in the 'quality' of the lines. Variations of this work develop and explore some of the basic visual qualities of the line such as width, shape, and growth speed of the repeated line (fig. 3.5). Lines look simple and 'mechanical', reflecting the simple computational system which created them.

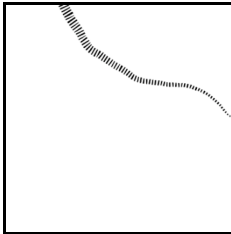


Figure 3.2 Screenshot from 'lineDrawingSimple2' (2006) accessed from: http://www.random10.com/colorcalm_research/2setting_parameters/applets/lineDrawingSimple2

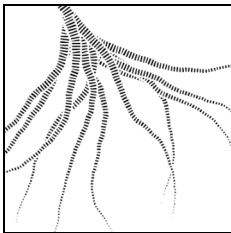


Figure 3.3 Screenshot from 'lineDrawingSimple3' (2006) accessed from: http://www.random10.com/colorcalm_research/2setting_parameters/applets/lineDrawingSimple3

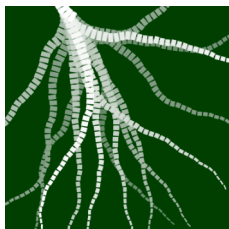


Figure 3.4 Screenshot from 'lineDrawingSimple3c' (2006) accessed from: http://www.random10.com/colorcalm_research/2setting_parameters/applets/lineDrawingSimple3c

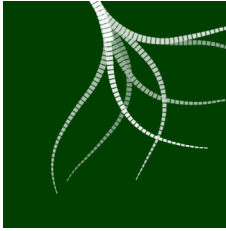


Figure 3.5 Screenshot from 'lineDrawingSimple3f' (2006) accessed from: http://www.random10.com/colorcalm_research/2setting_parameters/applets/lineDrawingSimple3f

3.1.3 Process (manipulation / control / skill)

The structure of the code allows the shape, width, length and speed of the line to be modified within the limited parameters of the programme. Direct manipulation of the key characteristics of a basic line is afforded via the allocation of individual variable values. Manipulating simple aspects of the number values (e.g. angle or inc) alters the qualities of the line. Changes within the lineDraw() function, relating to the way the angle is recalculated, allows simple basic connections between numeric manipulation and visual output to be made. Different calculations of the angle increment value (inc) yield different line shapes and alter the quality of the line:

Changes to 'inc' value	Type of line
<code>inc= inc + 1;</code>	curve.
<code>inc= random (-1, 1) ;</code>	random jaggy line.
<code>inc += random (-1, 1);</code>	line with random curves.
<code>inc = sine (angle) * 4;</code>	flowing sine wave.

Table 3:2 A list of calculations which alter line shape: Colorcalm

Line weight is manipulated by setting the initial width of the line and altering the amount by which the width is decreased. Larger values of decrement yield shorter lines. For example:

```
w = w - 0.4;
```

Line speed is controlled by frequency of the call of the `lineDraw()` method. For example the following code from the 'lineDrawSimple1' sketch calls the function to draw the line every two frames.

```
if (timer % 2 == 0) { lineDraw ( ) } timer ++;
```

Simple associations between logical comparisons and numeric values are made to affect the nature and type of work. For example the following code makes a decision regarding the minimum width of the line.

```
if (w < 2) { reset ( ) }
```

In addition to the way in which altering main variables makes changes to the nature of the line drawing, there is also an inter-connectedness of visual elements which may not seem to be otherwise connected. For example the structure of the code means that the width of the line helps to determine the line length: i.e. the line continues to be drawn and gets longer, until its width reaches a minimum value. Line speed can also be affected by the `radius` value which determines the size of each line 'segment'. Drawing large segments during each frame gives the appearance of a faster moving line. These basic associations between number and visuals establish an immediacy and directness in the development and use of computational material. Changes are easily made as the number values are altered allowing simple visual qualities to be explored. There is a simple directness between the code and the visuals allowing easy experimentation with individual number values which affect the visual qualities of the line. The limited parameters established at this point, however limit the visual range and expression of the work.

3.2 Developing More Lines (branches)

The next phase of the Colorcalm work alters and develops the computational system and structure in order to accommodate a greater number and variety of lines, expanding the visual range and 'vocabulary' of the work. A core element of the project is to create work which is visually rich and expressive, allowing the material the ability to generate a wide range of shape and forms. The capability of the material for visual expressiveness and a greater range of visual work therefore begins to be explored.

The main change in this stage relates to the introduction of an object-orientated (OO) structure into the code. The idea is that the object orientated structure can be used to create a robust, generic line 'class' which can accommodate a greater range of visual vocabulary and expression for the rest of the work. The concept of a single line is 'abstracted' into a `Line` 'class'¹ which enables a greater number and variety of lines to be created. The `Line` forms the conceptual and visual foundation for the rest of the development of the project, defining the key parameters and characteristics for all subsequent lines and determining the key elements and principles to be developed.

3.2.1 Code

The use and abstraction of the line into a `Line` class is an important conceptual step which establishes the fundamental properties of a 'line', and which is used as the basis for all other lines (see appendix 1.2). The main functions of the `Line` are summarized as follows:

¹ A 'class' can be described as the computational concept for an element in the programme - a computational template from which many similar objects can be created.

Line Class	
Function	Description
<code>Line ()</code>	a constructor function to set up the variables - the basic properties of each individual line.
<code>init ()</code>	a method to (re)initialize variables, to restart the line.
<code>run ()</code>	the main method keeps drawing the line or resets the line to start again, by calling either <code>plotpoint ()</code> or <code>init ()</code> functions.
<code>plotpoint()</code>	translates the stage to position <code>xpos</code> , <code>ypos</code> / calculates new <code>xpos</code> , <code>ypos</code> (based on angle and radius), draws a rectangle at point (0, 0).
<code>incrementShape ()</code>	a calculation to change the <code>x</code> , <code>y</code> , width and angle of the piece. The angle is recalculated according to a sine wave - make the line more link a wave.

Table 3:3 A list of functions from the Line class: Colorcalm

The core variables which allow the shape of each individual line to be manipulated remain the same as those outlined in stage one (table 3.1). Additional variables are included into the `Line` to control some of the structural and logical elements of the new data structure and to help develop the branching structure.

New branches are added via the addition of a 'recursive' element defined within the body of the line class. Recursion is a way in which a computational object creates a new instance of itself by a self-generating function. In this case the `Line` contains a recursive function which generates a new `Line` (as a 'branch'). The recursive function is called and branches are generated by a conditional statement which checks the number of segments which have been created. New variables are added to count the total number of segments for a line (`segNum`) and the current number of segments drawn by the line (`segCount`) and used to trigger the recursion:

```
if (segNum - segCount == 60) {  
    // new Line ( )  
}
```

The nature of this recursive process means that every line therefore inherits the same branching structure. Each line contains the functionality to add new lines (branches), and does so in the same way each time. This means that all the lines have the same structure and appearance.

3.2.2 Visuals

The visuals at this stage in the process have developed only a little from the previous stage. In terms of visual quality, each individual line maintains the same visual attributes as the previous examples. The key visual development is that of the branching structure which is a reflection of the changes made to include recursive functionality. The branching structure allows more lines to be drawn and affords the possibility of lines to generate other lines, however the range of visual expression is limited; most of the lines drawn are essentially the same. There are more lines generated, however more lines generate an increasing amount of similarity rather than difference (fig. 3.6); individual branches repeat the same quality and curve of line. Although the visuals are beginning to show some basic plant-like branching structure, they are still rather mechanical and lack the subtle variance and organic 'randomness' of botanical forms, which give Morris' 'Jasmine' and 'Willow' designs their visual liveliness.



Figure 3.6 Screenshot from 'lineDrawingSimple3fOOP3' (2006) from:
http://www.random10.com/colorcalm_research/3oop_simple/applets/lineDrawingSimple3fOOP3

3.2.3 Process / Manipulation

Manipulating the form of each individual line remains fundamentally the same as in the previous stage; the core attributes of `width`, `angle`, `x`, `y` and `radius` remain unchanged, and the 'quality' of each individual line is manipulated by altering the algorithm to re-calculate the angle.

In addition to the ability to manipulate individual lines, the addition of the recursive functionality extends the possibility of altering and manipulating the overall 'structure' of the plant shape. Manipulating structure means finding and changing the conditional statement used to govern when new branches are added, which is determined according to how many 'segments' have been drawn. Altering the conditional statement causes branching to occur at different times:

Conditional statement	Branch frequency
<code>if (segCount == 40)</code>	branch after 40 segments.
<code>if (segCount % 10 == 0)</code>	branch every 10 segments.
<code>if (segCount == segNum)</code>	branch at the end of the line.
<code>if (segCount == segNum - 20)</code>	branch 20 segments from the end of the line.

Table 3:4 Examples of conditional statements used to alter the branching structure:

Colorcalm

The number and angle of new branches can be changed and the plant structure, generated by inter-related lines, is starting to take shape. Although the manipulation of each single line shape remains the same, greater ability to manipulate the structure has been developed.

The development of the OOP structure means that the initial concept defining the key characteristics of a single line has become more firmly established, applying as it does to all of the lines and branches that are created. Having established the initial concept, the `Line` class begins to 'solidify' as the parameters and rules governing the material of the code start

to be put into place. The concept of the class forms the bedrock for the project as the rest of the work follows the path set by these initial conceptual and structural decisions. The specific range of visual vocabulary is now more clearly set and the core material qualities of the work is becoming well defined. The OOP structure has added an extra layer to the computational material of the work which allows the manipulation of the overall structure as well as individual visual elements within the work. However, although more lines can now be produced, the rules that govern each of the lines remain essentially static; more lines can be created but are created with more repetition and similarity. A greater degree of organic flexibility needs to be created within the material of the work and the visual vocabulary needs to be widened.

3.3 Variance and Difference: Inheritance

A key focus for this project is to generate computational work which reflects the concept and the aesthetic of the Arts and Crafts movement. The idea of the computational wallpaper is to generate work which uses the flowing botanic shapes and forms of William Morris' wallpaper design (e.g. 'Jasmine' and 'Willow' patterns) and re-apply them within the computational environment. The visual liveliness of these botanical patterns form the inspiration for the visual development of the Colorcalm project, as it seeks to computationally generate a range of moving stem, branch and leaf shapes. The need to develop a broad 'visual vocabulary' of line (which includes change, variation and randomness) is a key idea, affecting both the visual and computational, 'structural' element of work. The ability to introduce 'flexibility' into the project applies to both the visuals and the data structure from which the visuals are generated. The requirement to create a controlled, functional computational structure, which contains enough flexibility to generate work and which possesses a visual fluidity ('organicness'), is therefore key consideration for this entire project. 'Rigid'

stable programming structures which define the attributes of a line need to be set against the requirements which allow each line to have its own set of qualities and attributes. Flexibility, variance and difference therefore become key considerations for this stage of the work. Having already created a structured drawing programme, greater consideration of the visual quality and range of lines is required. This next phase of the project develops the basic class structure in order to accommodate a greater degree of variance and 'flexibility' into the structure, whilst maintaining the core essence of the line and the ability to control and differentiate between different types of lines.

3.3.1 Code

The basic line drawing programme is extended to allow similarity and variance by use of OOP based 'inheritance'. The `Line` class, is extended, creating new 'child' classes which inherit the core attributes of the line, whilst adding their own unique attributes. A summary of the class and inheritance structure is outlined in the following tables. The `Line` class is the 'parent' class, used as a template for other sub-classes.

Line Class: (Parent Class)	
Variables	Description
x	x location of line.
y	y location of line.
ang	angle of line (direction).
r	radius: distance between points on line.
w	width of line.
Functions	
run ()	keeps drawing or resets the line.
plotpoint()	draws a single point of the line in its new position.
incrementShape ()	re-position the location of the point.

Table 3:5 Overview of Line class as the parent class: Colorcalm

The Mainline class extends Line, but includes its some of its own methods:

Mainline extends Line	
Functions	Description
init ()	resets variables to begin drawing Line again.
run ()	adds a call to init () and calls makeBranch ().
makeBranch()	begins a new Line object.
incrementShape ()	extends parent function by including specific re-calculations for the shape (angle) of this line.

Table 3:6 Overview of the Mainline class which extends the Line class: Colorcalm

The Branchline class extends the Line and adds the following:

Branchline extends Line	
Variables	Description
distort	a number to specify the direction of the branch (+1 or -1).
go	a setting to make sure the branch does not start re-drawing.
leafLength	a variable to control length of leaf shape.
leafAngle	a variable to control angle of leaf segments.
Functions	
plotpoint()	new variables and calculations added to allow the ability to draw 'leaf' shapes after a given number of segments.
incrementShape()	extends parent function by including specific re-calculations for the shape (angle) of this line.

Table 3:7 Overview of the Branchline class which extends Line class: Colorcalm

The Line class describes the basic structure of the work, i.e. the plotPoint() and incrementShape() methods. Each of the subsequent class (i.e. Mainline and BranchLine) inherit the basic structure of the line class and add more specific elements and characteristics. Each type of line is therefore fundamentally the same but can differ in some key points. The Mainline includes a method to create continuous branching lines, whilst

the BranchLine contains an algorithm to create a different kind of curve and also creates other sub lines as 'leaves'. Subsequent sketches develop this central structure which remains a fixed element as more experimentation with the detail of the work (e.g. colour, shape, leaf structure) takes place.

3.3.2 Visuals

The branch lines are now computationally separate from the main line, each can have its own behaviour and characteristics and simple changes can be made. Branches can have their own shape, width and length. Early experiments at adding a 'leaf' structure is included. Visually the work begins to look a little more organic: a greater range of flowing lines are starting to emerge that have the potential to be developed into a wider range of organic shapes and forms. Development of the data structure has added a greater visual range and flexibility to the work. The material is beginning to show the capacity to develop a wider range of visuals.



Figure 3.7 Screenshot from 'lineDrawingSimpleInheritance1' (2006) accessed from: http://www.random10.com/colorcalm_research/4inheritance/applets/lineDrawingSimpleInheritance1



Figure 3.8 Screenshot from 'lineDrawingSimpleInheritance2' (2006) accessed from: http://www.random10.com/colorcalm_research/4inheritance/applets/lineDrawingSimpleInheritance2



Figure 3.9 Screenshot from 'lineDrawingSimpleInheritance2' (2006) accessed from: http://www.random10.com/colorcalm_research/4inheritance/applets/lineDrawingSimpleInheritance2b



Figure 3.10 Screenshot from 'lineDrawingSimpleInheritance2c' (2006) accessed from: http://www.random10.com/colorcalm_research/4inheritance/applets/lineDrawingSimpleInheritance2c

3.3.3 Process / Manipulation

This stage in the process is one of structural and data development in which the computational concept begins to widen to allow a greater degree of visual exploration and experimentation. Although detail of how each individual line shape is defined remains unchanged from the earliest experiments (using the same basic numeric manipulation), the development of the inheritance structure extends the amount by which the code and the branching structure can be manipulated. Manipulating and changing the visual structure involves altering the logic regarding the frequency of branches. Details of the shape and form of each line is changed from within the `incrementShape()` method of each class. The fact that each `Branchline` and `Mainline` contains its own versions of the `incrementShape()` method makes the process of change a little more complex. The core characteristics of the material are essentially the same as

those of the initial pieces of work. However structural change has added a further 'layer' of complexity to the computational material, not only providing greater opportunities to develop individual elements of the work, but also creating a more complex material structure. The addition of inter-related classes and variables begins to firmly define the data and material structure around which the rest of the project will be built. The inter-relationships between individual variable elements become increasingly involved, and the structure more 'rigidly' defined. As the variety of form grows, so the ability to directly manipulate key elements of the work becomes less direct.

3.4 Colour, Shape and Form

Having developed the OOP inheritance structure which defines the key concepts of the programme, the next stage of development focuses on the parameters which have already been created, exploring the visual vocabulary of the current system. The project moves from *creating* the material structure to *manipulating* the material structure. Having developed a framework which allows for a variety of shapes and forms, this phase attempts to manipulate the detail of the material to express a wider variety of organic, botanical elements. Manipulation of specific data within the structure allows visual details i.e. the colour, shape and form to be altered.

3.4.1 Code

Computational developments relating to this stage of the work concern the inclusion of a greater number of variables together with the manipulation of existing variables and algorithms which modify three key visual parameters of the work: colour, leaf shape and line form.

Colour

Control over the colour values is afforded by the introduction and use of variables to set and change the red, green and blue values. Variables are initially used to set the fill, background and line colour but once set, the values remain unchanged (fig. 3.11).



Figure 3.11 Screenshot from 'cc1' (2006) accessed from:
http://www.random10.com/colorcalm_research/5cc1_cc2/applets/cc1

During the later sketches colour values are modified so that the colour of each line changes during the lifecycle of the drawing. Individual red, green and blue values are allocated to the `Line` class; the variables are declared and initialized in the `Line()` 'constructor' method, or at the beginning of the sketch, and allocated as the stroke colour for each line as it is drawn.

```
tr = 95 ; // red
tg = 133; // green
tb = 44 ; // blue
stroke (tr, tg, tb);
```

As values are incremented, within the `incrementShape()` method, the amount of red and green of each line segment is constantly altered as the sketch is run. The following examples alter the amount of green (`tg`) and red (`tr`) by a randomly generated value between -20 and 20.

```
tg -= int (random (-20, 20));
tr += int (random (-20, 20));
```

Limitations are put on the number values to prevent them going too far off the scale. Logical 'if' statements used to control the minimum and maximum values of green and red are added.

```
if (tg < -200) { tg ==-200;}  
if (tr < -200) { tr ==-200;}  
if (tg >456) { tg=456;}  
if (tr>456 ) {tr=456;}
```

Leaf Shapes

A leaf shape is generated by modification to the width of the end part of a branch line, using a sine wave calculation to alter its width and give it the recognizable curvature of a leaf form. Modifications to the `Branchline.plotPoint()` method includes an algorithm to generate a 'leaf shape' to the end segments of the branch line. A new `leafSegment()` method is included within the `Branchline` class. New variables are added within `Branchline` to accommodate this extra calculation.

Variables	Description
leafLength	max length of leaf segment.
leafAngle	angle of leaf from branch.
xEnd	x endpoint of leaf.
yEnd	y endpoint of leaf.

Table 3:8 A list and description of new variables added to the `Branchline` class: *Colorcalm*

Along each segment of the branch a series of lines are drawn to create the leaf shape. The length of each of the lines is determined by a sine curve calculation, in which the width of the branch line is used to determine the size of the leaf.

```
float lineLength = sin (radians (counter))*w*8:
```

This `lineLength` value is used to calculate a point at a set angle from the branch:

```
xEnd = cos (radians (leafAngle)) * lineLength/n+ xPoint;  
yEnd = sin (radians (leafAngle)) * lineLength/n+ yPoint;
```

The width of each leaf is kept relative to the width of the branch line it is attached to, so as the width of the line gradually tapers out so does the leaf shape. The mathematical sine curve calculation ensures that each leaf maintains an even, constant shape. Variations in the amount and type of lines created to form the overall leaf shape yield a variety of results:



Figure 3.12 Screenshot from 'cc2a' (2006) accessed from:
http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc2a/



Figure 3.13 Screenshot from 'cc3' (2006) accessed from:
http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc3/



Figure 3.14 Screenshot from 'cc4' (2006) accessed from:
http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc4/

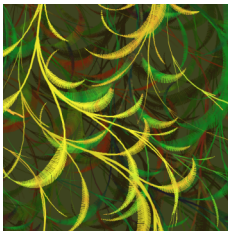


Figure 3.15 Screenshot from 'cc5c' (2006) accessed from:
http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc5c/

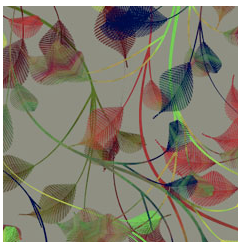


Figure 3.16 Screenshot from 'cc6b' (2006) accessed from:
http://www.random10.com/colorcalm_research/6cc2_cc7_leaves/applets/cc6b/

Line Form

The shape of each Branchline and Mainline is controlled within the `incrementShape()` method, using the same re-calculation of angle as established from the initial stages of development: sine wave, random and increment calculations (table 3.2). However, the addition of new variables used to modify the details of the sine wave calculation produce a different kind of looping line:

```
if (ratio < 80 ) { inc = 1; }; // minimum ratio
if (ratio > 220) { inc = -1 ; } //maximum ratio
ratio += inc;
frequency = 0.05;
magnitude = frequency * ratio;
timer += frequency;
angle += sin (timer)*magnitude;
```

The key factor here is that the `magnitude` value used to create the sine curve is altered to a changing value rather than a static one; sliding between a minimum and maximum value. As the value of `magnitude` gets 'bounced' between a minimum and maximum value, so the shape and curvature of the curve alters. Generating the code for this simple looking line proved to be more complex than had been first imagined. In addition, a `ratio` value is made into a variable value which is randomized and used to give each new line its own curvature setting, producing shallow or tight loops.

Manipulation of numbers via mathematical calculation is the means by which the essence of the shape and the aesthetic of the line is controlled. Changing the detail of the sine curve calculation and manipulating the `magnitude` and `ratio` values alters the shape, form and quality of the line.

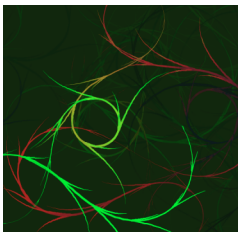


Figure 3.17 Screenshot from 'cc_basicVersion' (2006) accessed from:
http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_basicVersion



Figure 3.18 Screenshot from 'cc_basicVersion2' (2006) accessed from:
http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_basicVersion2

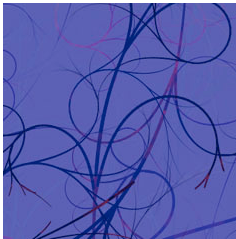


Figure 3.19 Screenshot from 'cc_march_03' (2006) accessed from:
http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03

3.4.2 Visuals

This stage of the process is characterized by visual experimentation and development. Details of the code are manipulated to change the visual elements of the line; colour, leaf shape and line form. The computational ability to control the details of line shape, colour, leaf shape and branching structure make these later pieces of work develop visual complexity, whilst remaining clearly linked to the conceptual core established at the start of the work. The structure has added more details which can change and control the visual pattern of the work. It is noticeable however that whilst development of the visuals reflects the development of detail within the computational structure, the overall combination of all the changes together does not necessarily make for a coherent or visually successful piece of work. There is no immediate correlation between the 'success' of the work with regard to computational functionality and its visual success. An example

of this may be seen in the use of colour. The introduction of variables within the structure allows colour elements to be changed and manipulated, however the changes to red and green values are made in a rather overstated and visually crude manner.

The work at this point therefore reflects the added complexity and functionality within the computational structure, but careful selection of these functions has not necessarily been used. This may be due to the fact that the work is being judged and assessed from a programming standpoint, and its success is bound up in the pleasure of 'getting the script to work'. The work is still attempting to include all functions possible which may be used with greater effect later on. Although some experimentation with form and visuals is being made, judgments regarding the 'structural' and functional aspects of the work are uppermost.

3.4.3 Process / Manipulation

This stage of the Colorcalm work is characterized by mathematical manipulation of the detail of the visuals. Colours, shapes and forms are represented by numeric values and algorithms within the line and branch drawing system. Calculations using sine, random and incrementation are used as the means by which these values are altered and changed. Key to developing the shape of the curve is the manipulation of the sine curve calculation. Rather than using a consistent sine calculation to re-calculate the angle, which would yield a regular unchanging curved line, the calculation was changed so that the magnitude setting was altered giving less consistent results, creating a curved line that changed and contained variance (e.g. fig. 3.19). Using the same sine curve and altering the magnitude calculation also produces different 'types' of leaf shapes (e.g. example see fig. 3.12, fig. 3.13, and fig. 3.14).

Much of the Colorcalm work involves allocation of numeric variable values to specific line attributes and the subsequent manipulation of these values through simple mathematical calculations. Combinations of algorithms, e.g. randomness, incrementation or other simple numeric transformations, are used to change the specific visual attributes. Although the computational structure has been developed throughout the process, the work remains true to the core computational ideas developed in the initial line drawing sketches i.e. assigning number values to represent line position, width and colour. The intrinsic 'values' of the computational material remain the same.

This stage of the work emphasizes the manipulation rather than the creation of the computational structure. The introduction of variables and individual data elements allows a direct means of experimentation with core elements of the programme which directly affect the aesthetics. The mathematical modification of individual values alters the way or rate of change and directly affects the on-screen visuals. Even slight changes can significantly affect the colour, shape, speed and type of movement. This represents an experimental part of the process which introduces a more 'playful' element to the work.

3.5 Petals and Flowers

Having defined the structural detail of the work and having created the ability to generate and manipulate line shape, colour and the branching properties of the drawing, this phase of the work attempts to expand the visual vocabulary of the lines by adding leaf and petal shapes. This is done in order to develop visuals which have a closer resonance with the organic, flowing, patterns Morris used in his wallpaper design. The aim is to begin to move the work closer towards the aesthetic pattern of the Arts and Crafts movement and in particular to reflect the elegant complexity and 'depth' of the leaf and branching structure of the 'Jasmine' and 'Willow' pattern designs.

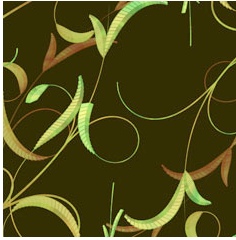


Figure 3.20 Screenshot from 'cc_march_04' (2006) accessed from:
http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03



Figure 3.21 Screenshot from 'cc_march_05' (2006) accessed from:
http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03

3.5.1 Code

To accommodate the idea of the flowers and petals a new `Flower` class is added to the project. The `Flower` class, is similar to `Line` and `Mainline` and is essentially another 'line drawing' class which has its own characteristics.

Flower Class	
Functions	Description
<code>Flower (x, y, radius, angle)</code>	constructor method of the flower class.
<code>drawLine ()</code>	a function used to draw a set number of lines, one after the other, each with its own length and angle. The line length and angle is incremented very slightly each time and so the overall result is of a single 'petal' shape.

Table 3:9 Overview of the Flower class: Colorcalm

For each single petal a series of seventy lines are drawn one next to the other. Each line is given a new angle and line length. The angle is incremented slightly, and the line length is calculated using a sine curve to create the petal shape:

```
angle += radians (60);  
drawAngle += radians (30);  
radius = sin (radians (5*angle/2))* fSize;
```

The number of lines, the incrementation of angle and the calculation for the length of each line determines the shape of the petal:

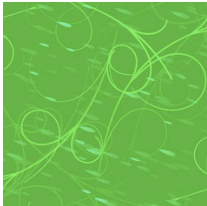


Figure 3.22 Screenshot of initial flower tests from 'cc_march_05_flower2b' (2006) accessed from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03

After creating a single petal the next step generates a complete flower by repeating the petals a number of times. The flower concept is constructed around the concept of simple use of computational logic: a 'for loop'. A for loop is used to draw a number of petals at the same time, the completed flower is therefore drawn in one go.

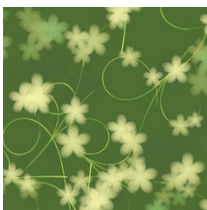


Figure 3.23 Screenshot from 'cc_march_05_flower3' (2006) from: http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03

Core visual elements of the Flower are outlined as follows

Attributes of Flower	Description
Shape	The for loop creates the flower all in one go.
Colour	Colour of the flower is dictated by the (universal) changes in colour values affecting the rest of the sketch (tr, tg, tb).
Appearance	A number of flowers are drawn over the top of one another, to give each flower a different size, and different visual quality. Transparency applied to the flowers to give a softness.

Table 3:10 Description of visual attributes of the Flower class: *Colorcalm*

Specific variables which define attributes of the Flower are outlined as follows:

Variables	Description
fSizeMin	starting size of flower - this is incremented to re-draw the flower until reaches fSize.
fSize	overall (max) size of flower.
radius	length of petal.
More variables added later (cc_aprilTest2)	
petalCount	number of petals on flower.
rotation	amount of rotation between each line.
increaseAngle	amount of rotation between each petal.

Table 3:11 Core variables of the Flower class: *Colorcalm*

Once the core functionality of the `Flower` is established additional variables are used to generate more variance and randomness between each flower. Random numbers and sizes of petals are added to the branch by assigning a random value to a new `petalCount` variable.

```

for (int i=0; i<=int (random (1, 5)); i++) {
  petalCount++;
  myPetal [petalCount] = new Petal (xpos, ypos, x, 72*i+1);
}

```



Figure 3.24 Screenshot from 'cc_march05_flower3b' (2006) from:
http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03

In addition to flowers and petals, further complexity is added by developing the detail of the branching structure, i.e. branches with sub-branches. This is done to create a greater number of 'levels' within the structure, to develop the growth of the computational plant and to bring closer to the organic structures of Morris' 'Jasmine' and 'Willow' pattern designs. In order to achieve greater 'depth' the to branching structure, the `makeBranch()` method is given additional parameters ('arguments'). The `Branch()` constructor method extends the number of variables to accommodate the possibility of more detail: providing the ability to manipulate its curvature and branching frequency. For example a branch frequency (`bf`) variable is added to control the how often branches occur:

```

if (segCount % bf == 0) { makeBranch ( ) }

```

New variables which control the branching structure.

Variable:	Description
distort	controlling direction of the branch (+1 or -1).
depth	a number to control the level of recursion for sub-branches.
ratio	number used in the calculation of the curve.
bf	branch frequency a number to allocate the occurrence of new branches.

Table 3:12 Description of variables added to control branching structure: Colorcalm

The constructor methods for `mainLine` and `branchLine` are also developed to accommodate new variables.

```
Mainline (x, y, ang, r , w, bf)  
BranchLine (x, y, ang, r, w, distort, depth, ratio, bf) ;
```

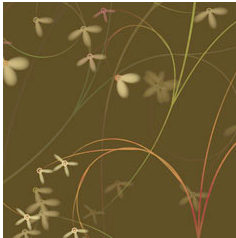


Figure 3.25 Screenshot from 'cc_aprilTest2' (2006) accessed from:
http://www.random10.com/colorcalm_research/7ccMarch_loops/applets/cc_march_03

3.5.2 Visuals

Visually the work at this stage continues to explore the parameters of the branch system already created. The work still lacks the subtlety, especially in terms of colour, as the usage of red, green and blue often clash producing garish results. There is a more visual organic variance of the shapes and patterns of the forms but this is very controlled.

There is a visual and computational 'simplicity' about the work; flowers, leaves and lines all apply the same basic concepts of programming (numeric variables, loops, iteration) and mathematics (sine and trigonometry) to produce the visual results. Although the structure is becoming more complex, the work represents an exploration of the basic 'building blocks' of computational material, specifically the relationship between number and visual, in an effort to understand and apply the basic rules of computation and calculation towards an elegant, visual outcome.

3.5.3 Process / Manipulation

The class structure of the line drawing system is now coming up against its limits. The structural qualities of the original work are proving to be increasingly difficult to manipulate as a wider range of forms are developed within the set parameters and limitations of the computational structure.

Adding a greater number of variables allows closer control over the types and the structure of the forms which can be created, however it also adds another layer of complexity. As the structure becomes increasingly complex so does the ability to make real and direct changes. The simplicity and directness of the early sketches is starting to get lost as the data and the structure becomes more layered. The core line-drawing data structure works for simple shapes and marks but is struggling to be extended into other shapes and forms (i.e. petals, flowers, and leaves). The amount of lines required for the drawing to create a complete petal or flower shape makes the processing speed of the program slow, especially when lots of petals are drawn altogether. The initial concept of iterative line drawings is being stretched almost to breaking point. Conceptually the structure struggles to accommodate the drawing of petals and flowers. Petals are conceived and drawn as a series of individual lines and not as a unified flower shape. The work is visually unified but is structurally disconnected, there is little

structural linking of lines and elements, even though this is what appears to be happening on the screen. The petal and flower ideas are removed and discontinued from the work.

It is important to note that the petal and flower ideas are discontinued for computational rather than aesthetic reasons. The petals and flowers do not work well within the computational concept and structure which has already been established, and prove to be too far removed from the original idea. In this way it is possible to see how the computational material determines the type of work produced, influencing the nature and direction of the project. The work is being produced in a way which is 'true' to its material, and the intrinsic computational 'honesty' is maintained. Work that is *computationally* inappropriate is discontinued. Judgments about the suitability of an idea are not based solely on aesthetic reasons but are also based around how it works alongside the materiality of the code.

3.6 Leaf Class and Final Pieces

In the final stage of the work the material structure of the code employs the initial core line drawing concept but does so as part of a computational structure which is growing increasingly rigid, almost immovable. Having developed and added classes and structural elements throughout the process right from the initial line drawings, scope for developing the structure at this stage of the work is limited. Although a final `Leaf` class is added to develop and define the leaf-like structures and forms, much of the rest of the work takes the form of exploring and manipulating numerical details of the structure in order to arrive at final pieces which create the best possible decorative wallpaper-like shapes and forms.

3.6.1 Code

A `Leaf` class is added alongside the `MainLine` and the `Branchline` as yet a further extrapolation of the `Line` class. Many of the same variables and functions used in the `Mainline` and `Branchline` classes are used again, allowing for individual changes which create the distinctions between the different types of lines.

New variables have also been added to the main project replacing 'hard coded' numbers with the intention of allowing greater access to, and control of, more of the visual details of the sketch. The final pieces of work use a range and combination of a wide range of simple variables, created and added over the course of the project to define and control the all the key visual elements and attributes of the programme. A summary of the key variables is listed as follows:

Variable	Description
count	counter used as timer.
branchNum	number of branches created in the plant.
petalCount	number of petals.
values affecting colour	
redFraction	amount by which red is adjusted.
blueFraction	amount by which blue is adjusted.
greenLine	amount by which green is adjusted.
redLine	amount of red in line colour.
blueLine	amount of blue in line colour.
bgColour	background colour.
values affecting the branch properties	
b_mr = 20;	branchMin ratio.
b_mxr = 200;	branchMax ratio.
b_bf	branchFrequency for sub branches.
b_len	branch length.
b_ang	branch angle.
values affecting the leaf properties	
l_f = 20	leaf frequency.
l_srt	leaf start - when leaves begin on the branch.
l_w	leaf width.
l_s	leaf size.
l_len	leaf length: amount by which the length is multiplied.

Table 3:13 Description of the key variables: *Colorcalm*

3.6.2 Visuals

Closer attention to the detail of variable values yields results which display more subtly within the work as the visuals begin to reflect the 'Arts and Crafts' aesthetic of the Morris wallpaper more clearly.



Figure 3.26 Screenshot from 'cc_aprilTest5a' (2006) accessed from:
http://www.random10.com/colorcalm_research/9ccApril_leafclass/applets/cc_aprilTest5a

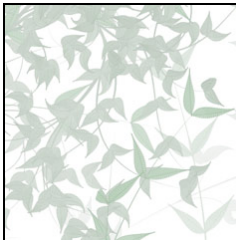
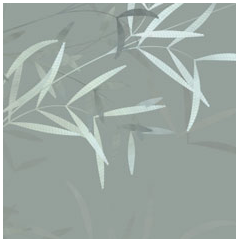


Figure 3.27 Screenshot from 'cc_aprilTest5b' (2006) accessed from:
http://www.random10.com/colorcalm_research/9ccApril_leafclass/applets/cc_aprilTest5b

Final variations of the Colorcalm work:



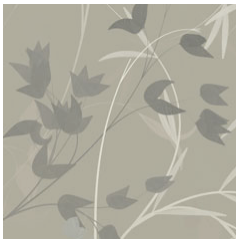
cc_May4b



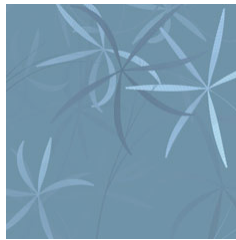
cc_May5



cc_May7



cc_May7b



cc_May8



cc_May8b

Figure 3.28 Screenshots of final variations of Colorcalm work (2006) accessed from:
http://www.random10.com/colorcalm_research/

3.6.3 Manipulation

Having created and established the structure, which is now almost immovable, close attention is paid to the specific number values of the variables. Which value, for example, works best as the `ratio` value, which colour values and combinations work more successfully and more 'harmoniously' together etc. More attention is also paid to the parameters of the variable values, i.e. understanding the upper and lower-most values pertaining to colour, angle, number of branches and leaves. Attention is paid to specific variable values in order to hone the control of the visuals to achieve results which reflect those intended. Slight tweaks or changes to a single numeric value, or algorithm alters the overall visual flow of the work. A long, iterative process of change and testing is employed as aesthetic decisions about each detail of the work are made, and several variations of the work are produced. Greater number of variables also, however, leads to greater number of problems when it comes to trying to change or alter individual visual elements, especially when the structure is becoming increasingly complex with lots of similar-yet-different classes. This stage of the process is reaching the point in which the structure 'creaks and wobbles', it becomes almost unmanageable: finding single values can be difficult and one value may have a complex knock-on effect with unexpected results.

3.7 Summary of Colorcalm: Dialogue between Form and Function

The Colorcalm work represents the researcher's first attempts at using programming code to produce creative, visual work and therefore reflects an initial 'learning curve' of understanding. It is the initial attempt to apply some of the Arts and Crafts values and aesthetics to the work, emphasizing the hand-written process of programming as a direct means of testing and exploring the basic 'material' qualities of computation. Initiated by the desire to create work using only simple, 'hand-written' code, in an environment stripped of software menu options and short cuts. The idea of 'remaining true' and honest to the material is something which remains a clear motivation throughout the work, informing the choice and direction of the project throughout the development process. Even as the computational material grows, changes and becomes more 'rigid', the desire to 'remain true' to the material structure, not twist or stretch it into forms to which it is not suited, is something which informs the work throughout.

Emphasis placed upon 'handwriting' code as process places a requirement on the researcher to learn and apply the technically correct syntactical and grammatical aspects of the programming language - the core 'values' of the material. A trial-and-error learning process defined much of the early engagement with the 'material', providing a valuable means of understanding its core characteristics and values, laying an important foundation for later development and experimentation. Although physical manipulation of computational material is not possible, direct manipulation of code is achieved by 'manipulation' of individual data elements within the computational structure. Experimentation with simple variable, numeric, logical and structural elements of the programming environment is used to gain an understanding of the basic materiality of the computational environment. Control of the material is therefore achieved by numeric

manipulation of variables and algorithms which define specific visual attributes of the work.

The overall process shows development from a simple to a more complex structure, as the desire to create a material environment which is both computationally sound and which contains enough 'flexibility' to generate a wide variety of visual shapes and forms becomes increasingly important. As the project progresses, the focus moves from understanding the core *technical* qualities and requirements of the material, towards developing an *aesthetic* understanding of the material. By concentrating on reproducing a computational version of the flowing lines and forms, inspired by Morris wallpaper pattern design, the work moves from the *development* of the computational material towards the *manipulation* of the computational material. Interplay between the structure of code and its aesthetics, between the technical ('rigid') rules of the code and the fluidity of the visuals forms a core part in the formation and development of the work. This reflects the wider dialogue of the traditional design environment, a dialogue in which the key elements of code visual and process become intertwined, and in which understanding of the values of the material are displayed and applied. This dialogue between maker and computational material reflects the values of William Morris whereby the material values and limitations are to be understood and worked with. The following offers a summary of the Colorcalm project work using the headings of 'code', 'visuals' and 'process' as outline headings for further discussion.

i. Code

Immediate attention is placed on the direct connection between the use of simple variable values and the visual development of the line. Initial tests and experiments centre around simple 'mark-making', testing and establishing connections between both variables and algorithms with line quality, shape colour and width, etc. Experiments with simple computational

ideas, particularly recursion are undertaken to explore the structural and generative nature of the material. Ideas and tests with object oriented principles using simple classes are developed in an attempt to widen the visual and structural scope of the work.

As the structure becomes increasingly complex, so development of the work becomes increasingly reliant upon development of the detail of the work, i.e. the individual variables, properties and shape algorithms. An increasing number of variables are added to manipulate specific elements for each of the different line types (colour, speed, number or leaves, shape of leaves, frequency of leaves, shape, etc.).

ii. Process

During early developmental stages of the work, the general dynamic between form and function remains relatively fluid: the functional elements are established, and aesthetic elements can easily be experimented with and explored. The overall structure is easily manipulated and individual properties modified and altered. As the work progresses however, and the data structure is more rigorously defined, the relationship between structure and individual parameters becomes increasingly rigid. Increased computational complexity, the addition of more functions and classes, etc. develop an increasingly 'solidified' structure. Computational rules dominate the work as greater amounts of data are added. More lines and leaves are produced, but the rules that govern each of the lines become increasingly specific and complex. Manipulation and change of the data becomes increasingly difficult as the material becomes more defined.

By the end of the work the shape and format, the material, of the code feels almost solid; the structure is set, and altering the direction of the work is almost impossible without beginning again. Individual variables and functions enable lines to be changed and manipulated but the data structure is so rigid

and unwieldy that making real changes to the work, to develop it functionally or aesthetically, is increasingly difficult. This relationship between the functional and aesthetic elements changes over time as the function becomes more established and the opportunities for flexible development of the work become less and less. As complexity of the visuals increases, the fluidity and flexibility of the work gradually decreases. This is perhaps most evident in the attempt to include flower and petal forms into the work. Although these shapes started to 'work' on a visual level, the lack of real connection with the underlying data structure means that creating these flower forms is achieved against the structure of the rest of the line drawing application. The material of the data structure is judged to have been stretched too far in creating the flower shapes and they are not used. Adding more elements into the data concept involves 'pushing' and 'pulling' the structure beyond the realms for which it was originally designed; the addition of visual complexity also adds structural complexity and instability. The programmed material i.e. the structure, is not robust or flexible enough to accommodate greater demands of more a more complex form and the elegance of the visuals is not reflected in the cumbersome structure of the code. Development of work here is not just a matter of getting good visual outputs, there is an equally strong emphasis placed on creating work which is successful on a structural basis, work which has both visual and structural integrity. Attaining the balance between mastery and twisting of a material (Morris, 1882) is a fine one. Developing the data structure almost has a feeling of developing the fabric, and the structure of the material itself (Watz, 2006).

iii. Visuals

The Colorcalm project demonstrates a connection between the nature of the data structure and of the visuals, and shows a clear connection between the way in which simple number values can be used to represent visual attributes of a piece of work. The simple development change and growth of

the lines and forms reflects a craft-like simplicity to the aesthetic of the work which in turn reflects a simple and direct connection between individual variable values and the work on screen. The shapes and forms are not overly complex and are direct and simple manifestations of the 'material', i.e. the numeric values from which they are created. Data values define a limited range of visual elements: shape, width and colour; the simplicity of these may be seen are directly represented by the simplicity of the aesthetics.

The code connects with the visuals not just on the 'low' level in which a variable represents a single visual attribute of the shape (e.g. width of line or colour) but on a higher level in which the entire data structure, the application of objects, classes and methods and the interplay between them, impacts upon the quality and direction of the work itself. Increased structural complexity and 'rigidity' is reflected by the visual development of the work: although the individual shapes and forms maintain some degree of elegance and simplicity, the work culminates in a series of pieces which are visually 'rigid'. The leaf and plant shapes are generated by a generative computational process but have the appearance of work which may be animated along more traditional lines. The on screen shapes and forms are 'static' and demonstrate little of the dynamic, or behavioural qualities of the computational environment, a reflection of the increased complexity and rigidity of the data structure.

iv. Conclusion

The Colorcalm project can be defined as an interplay between the conflicting opposites of 'function' and 'aesthetics'; the functionality of code and the aesthetics of the visuals. The project highlights how important it is to view the process of creating work as an ongoing dialogue between contrasting elements of the computational environment; a harmonious interplay between the structural, functional concerns of the material and the visual, aesthetic qualities of the work. The emphasis here is not purely on form or function, on

knowledge or feeling but a dialogue and harmony between the two. Experience of undertaking the Colorcalm project suggests that successful work which demonstrates elegance and subtlety in its sympathetic use of the material is more likely to arise out of the development of a useable elegant, flexible structure around which the detail of a project can be developed. Craft-like programming work arises not only from gaining an understanding for the detailed elements (numbers and variables) of a project, but from establishing an elegant structure which allows work to develop in a fluid and flexible manner. Elegance of programming can therefore be revealed through the elegance and fluidity of the structure which allows for greater harmony between form and function of the work. Overarching systems and concepts (e.g. data structure) must be applied to the particulars and details (e.g. data detail and variables) of the work. The next project will pay more attention to these over-arching data structures.

The ongoing tension between function and form provides a framework thorough which many other 'conflicts' of the process may be viewed. The relationship between these contrasting aspects of the process reflects other tensions between opposing viewpoints. Each aspect of the process represents a series of related, wider concerns, and the discourse reflects wider tensions within the process. Issues of control and randomness, elegance of form and of structure, value judgments of success, form and function, humanity and automation, technology and craft are all reflected by the broad dialogue encompassed by machine and material. The culture and climate of creative-programming is therefore typified by the conflict between each opposing ethos. Viewing the process in terms of a dialogue between opposites helps the understanding of the tensions related to all these other similar issues.

4 Moving Wallpaper Project

The overall aims of the computational wallpaper project, outlined in detail at the beginning of chapter 3, define the overarching parameters, ideas and intent for both the Colorcalm and the Moving Wallpaper projects. These can be summarized as the intention to re-apply concepts and aesthetics inspired by traditional design values, and more particularly the Arts and Crafts movement, into the computational environment. Emphasis on understanding the intrinsic material values of computation, and applying them to the creation of botanically inspired organic forms, provides the context and motivation for both projects which resonate with the values and aesthetics of the Arts and Crafts movement. Elements of the Arts and Crafts aesthetic, specifically William Morris' 'Jasmine' and 'Willow' designs, have therefore been used to provide specific aesthetic direction for the work, and used as a basis for understanding and documenting the formal, material aspects of programming code, syntax and structure.

The Colorcalm project has outlined the first attempt at producing computationally generated, Morris-inspired lines, forms, shapes and patterns, and marked the first stage of research as a designer moving into programming. The work explored and investigated the fundamental qualities of the computational material, using basic programming details (e.g. variables, logic and algorithms) to set and generate flowing organic plant shapes and forms. The project therefore demonstrated how the application and manipulation of individual numeric details can be used to set and generate paths of movement and colour, shape and form.

Although the Colorcalm project work has successfully created a range of computationally generated botanical shapes and forms, highlighting the value of numeric detail as a key part of computational work, the conceptual

and aesthetic attributes of the project still require further consideration and development. The final pieces of the Colorcalm work lack the visual and structural 'flexibility'; a key attribute of the fluid, reactive computational environment. Plants and shapes grow across the screen as the programme generates and draws them, but once created, they act as 'static' visuals without any reactive or behavioural qualities which allow them to move, develop or change. The Colorcalm work creates a series of patterns which lack integrity with reactive characteristics of computational material, and as such miss a key conceptual element of the traditional design values which inspire the project.

The idea that drawings on screen possess a kind of reactive organic potentiality is an important consideration for the project. Maeda's term 'reactive' graphics i.e. graphics which have the ability to be changed or to move, is key. One of the fundamental features of computational, screen-based objects is that they are not static, fixed in time like traditional media, nor are they linear and animated like 'traditional' time-based media, but possess behavioural, reactive qualities. As has been discussed (2.2.4) the computational object is algorithmic and behavioural, subject to the generative changes of the code from which it is created. The potential always exists to change or modify the computational image according to input elements or internal / external data. The 'behavioural' element is a key, distinct aspect of computational, screen-based work. It is important therefore, that the line is not a static mark on the screen, but that it expresses the reactive element of the computational (screen) environment. A lack of this behavioural quality in the Colorcalm work therefore means that its conceptual integrity to the reactive, computational material is undermined and the work lacks 'truth' to its reactive, behavioural material.

Lack of behavioural fluidity is also mirrored by a lack of visual variance, randomness and the limited types of plant forms that the Colorcalm project

can produce. The aesthetic aim of the work is to generate visuals which reflect conceptual links with traditional design, and create visual links with the organic qualities of structure and form of Morris' wallpaper work. Although the Colorcalm pieces allow colour and shape to be modified, the facility for developing a wide visual vocabulary of plant shape and form has not been fully realized. The final visuals of the Colorcalm work offer a limited range of branch structures and leaf shapes, with little variation of colour within each plant, and no facility for generating petal or flower shapes. The lack of both visual and 'behavioural' fluidity within the work is a product of its underlying rigid computational structure. Emphasis on creating and manipulating numeric detail within the project resulted in a material which became increasingly complex and rigid, making visual and behavioural variance difficult. The Colorcalm structure lacks the flexibility to reflect the reactive behavioural attributes of the computational environment or the organic visual variety of the leaf, flower and branching structure of the Morris aesthetic.

Having explored the basic numerical detail of the material with limited conceptual and aesthetic success, the next phase of the project, the Moving Wallpaper work, intends to reflect more clearly an understanding of, and 'truth' towards, the behavioural and flexible elements of the computational environment and aesthetic. The aim of the Moving Wallpaper work is therefore to create pieces which are both behaviourally and aesthetically flexible; which can move and react to the user, allowing a greater more varied range of plant leaf and flower forms to be created. The visual concept of the project has moved from the idea of a line as being a straightforward visualization of simple numeric data (variable values) towards the concept of the idea of the computational line as a generative, organic, line with behavioural and "reactive" (Maeda, 1995) qualities. Greater structural and visual flexibility will therefore be developed via a new data structure which affords a greater degree of fluidity, creating a wider range of visual outcomes

each which have the potential to be more 'reactive', i.e. have the potential to be manipulated, by external forces (e.g. mouse cursor). The work aims to develop a closer understanding and greater use of the core concepts of the reactive computational design material, demonstrating a wider visual vocabulary to reflect the organic variance of Morris's wallpaper aesthetic. Both aesthetic and behavioural elements of the work will try to more accurately reflect a greater range of organic plant, leaf and flower forms, using direct observation from nature as inspiration. By creating greater visual and structural flexibility, (concentrating on the naturalistic, behavioural elements of the line) this project aims to produce work which expresses 'truth' to the behavioural qualities of the 'reactive' computational material developing closer conceptual links with the ideas and aesthetics of the Arts and Crafts movement. Developing an organic, behavioural quality to the lines, shapes and forms also helps to reinforce the ideals of the work and ideas of the Arts and Crafts movement which cited nature as the ultimate inspiration and model for creative activity (2.3.4) against which all design work should be judged.

It is worth noting here that although the work seeks to develop a behavioural element of the line which 'reacts' to the viewer, interaction as a subject is not something which is within the scope of the research and will not be explored in-depth as part of this thesis.

The following chapter provides an outline of the development of the Moving Wallpaper project. The project is divided into six main stages of development, from concept to the final visuals and variations. The descriptions include references to the online example files for each stage of the work. Figure 4.1 (next page) shows a visual overview of the project which is included on the CD in appendix 4 and online as a web page available at http://www.random10.com/movingwallpaper_research. The performance of these java applets will vary according to the browser and

operating system upon which they are run. The CD also contains the Moving Wallpaper project work as a series of Macintosh formatted 'stand-alone' applications these have been included due to inconsistencies with the pieces when viewed in a browser. Outline elements of code for each stage of the work is available in appendix 2.

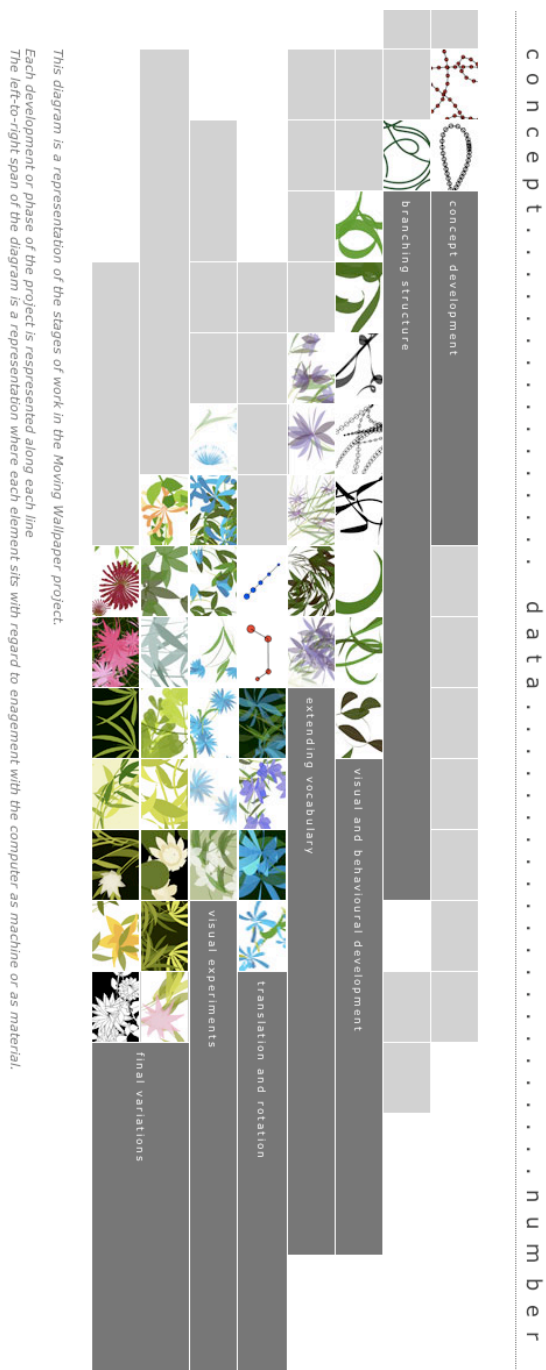


Figure 4.1 An overview diagram of the Moving Wallpaper project available at http://www.random10.com/movingwallpaper_research/

4.1 The Concept Stage

The first stage of the Moving Wallpaper work defines and establishes the core structural elements and concepts of the project. As with the Colorcalm work, a concept has to be established which determines how a line is created and drawn. Whereas the previous piece of work defines a line according to the movement of a single object leaving a trail across a screen, the concept for this work is centred around the idea of the computational line as a "reactive" (Maeda, 1995) line; an object which 'exists' on screen, and can move, react and interact. Just as the line is 'conceived' in a different way to that of the Colorcalm project, it is also defined in a different way, as a series of individual objects generated as the mouse moves across the screen.

4.1.1 The Concept Stage: Code

The Moving Wallpaper project takes a strongly Object Orientated (OOP) approach to the work and this is reflected by the concept of a line as a series of separate, but connected, individual behaviours and functions. Rather than attempting to create and manage the entire functionality and behavioural qualities of the line as a single class, the work abstracts different facets of the line into a series of separate building blocks of 'classes'. The basic line is defined by combining two classes, `Line` and `Ball`. The `Line` class creates holds and manages a group of `Ball` objects together, allowing functions to be applied to an entire group of balls. The `Ball` class manages each specific single ball as an individual object, defining the detail, e.g. the location and functionality, for each.

As the mouse moves across the screen a trail of individual balls is created, which are managed by an array within the `Line` class.

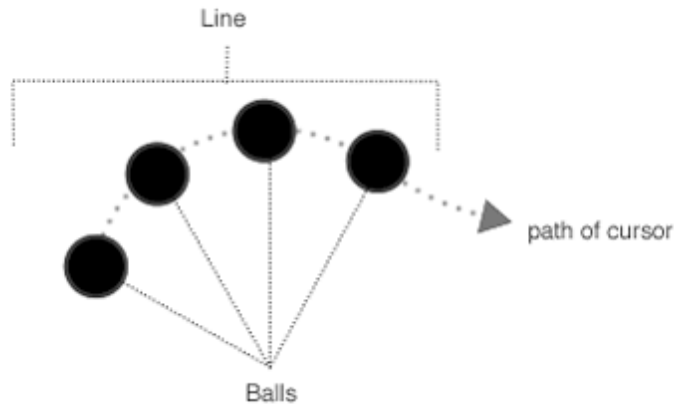


Figure 4.2 An illustration of the relationship between the Line and Ball classes in the Moving Wallpaper project.

This differentiation between the whole (Line) and the individual elements (Ball) immediately establishes a flexible way of thinking about and operating upon a single line. The following tables summarize each of the basic classes.

Ball Class	
Variables	Description
loc	location of ball.
vel	velocity of ball.
w	width.
Functions	
setLoc()	calculate location of ball.
drawBall()	draw ball at set location.
drawLine()	draw line between current and previous ball.

Table 4:1 Overview of the Ball class: Moving Wallpaper

Line Class	
Variables	Description
ballCount	number of balls created in the line.
ballArray	a list of all balls in the line.
Functions	
updateTarget ()	updates the position of the target which the line follows (e.g. the mouse).
drawBalls ()	draw all of the balls in the line.
addBalls ()	create new balls as the line lengthens.
makeSpringy ()	apply tension and spring between balls.

Table 4:2 Overview of the Line class: Moving Wallpaper

A Spring class is added to the Line and Ball class system to add to the visual and behavioural dynamic of the line. The Spring class calculates the tension, friction and 'springiness' between any two given objects, updating their location according to the values of stiffness, damping and mass variables.

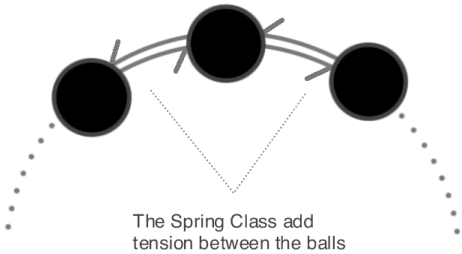


Figure 4.3 A diagram illustrating the function of the Spring class in the Moving Wallpaper project.

A new function, `makeAllBallsSpringy ()`, is added to the Line class to allow a Spring behaviour to be introduced to each of the balls.

Spring Class	
Variables	Description
stiffness	a value which sets the tightness of the spring.
mass	a value which affects amount of spring.
damping	rate of slowdown.
springLength	distance of spring.
Functions	
updateSpring ()	a function which applies the spring functionality.

Table 4:3 Overview of the Spring class: Moving Wallpaper

4.1.2 The Concept Stage: Visuals

The visuals at this stage of the work are functional and basic. However it is not the visual but the behavioural qualities of this simple 'chain of balls' which are most interesting. The addition of the `Spring` class and functionality adds a behavioural quality reflecting the idea of the computational line as a 'living', 'reactive' organic, object. The addition of this 'fluid' visual quality adds an extra, almost tactile, quality to the line.

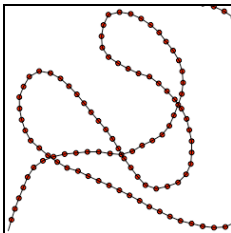


Figure 4.4 Screenshot from 'Jan08_plotpoints3' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/1_concept/jan08/

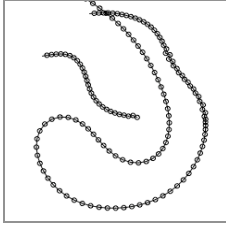


Figure 4.5 Screenshot from 'Jan08_plotpoints3_spring' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/1_concept/jan08spring/

4.1.3 The Concept Stage: Process

A key point in the creation and manipulation of this work is the inter-relationship between classes and the 'flow' of data between them. Each class encapsulates an individual concept of the line, which can be summarized as follows:

Class Name	Description
Ball	individual unit of a line.
Line	whole line, a collection of ball objects.
Spring	force acting between each line segment.

Table 4:4 Summary of initial classes: Moving Wallpaper

When each of these elements is put in relationship they all contribute to the overall definition of the line. The 'flow' of data between each of these separate classes begins to create a flexible and strong structure providing the ability to identify and make specific changes to individual elements and characteristics of the line. Once the basic line-drawing trail is created, the addition of extra behavioural qualities, (e.g. `dragMe()` and `makeAllBallsSpringy()` functions) can be added relatively simply. Developing a structure for the line, based around separate individual classes of behaviour and functionality (`Line`, `Ball`, `Spring`), opens up the idea that the line can be based on much broader computational concepts which includes the possibility of adding more behavioural qualities.

4.2 The Branching Structure

The basic structural building blocks (`Line`, `Ball`, `Spring`) have been put in place. This stage of the project expands the idea from the single line towards a self-generating, plant-like, branching structure of Morris wallpaper pattern. This includes the development of two new elements of the project:

- A behaviour to replace the use of the mouse as the means by which the lines are drawn and created.
- An extension to the logic of the structure to allow new lines and new branches to be created as off-shoots of the main lines.

The first of these elements is solved by the introduction of a new (`Target`) object whose movement replaces that of the mouse (see appendix 2.2). The second issue of the branching structure requires wider research and development. One of the most difficult elements encountered in the *Colorcalm* work was that of creating and handling the branching structure and growth. Different types of organic elements (e.g. branch, leaf or stem) were developed via a series of rather mechanical, difficult to manipulate, logical conditional statements. For example:

```
if (timer % 50 == 0 ) { // draw branch }
```

Research into computational systems of organic behaviour, growth and regeneration uncovered the concept of the L-system (Flake, 1998, p.77). The concept of the L-system is one which closely matches the requirement for computational simplicity, flexibility and integrity in this project.

There is not enough space here to give a detailed explanation of the L-System, but the core concept is one which encompasses a neat, flexible, simple, computationally organic concept. The L-system is a rule based system in which individual letters (or 'characters') are made to represent a

single specific function defined as part of the system. As the programme is run each character is read, interpreted, and used to trigger its associated function. A series of individual characters (or a 'string') can therefore be used to encapsulate an entire sequence of instructions. In this way the logic of a branching structure is represented not by a list of conditional statements but by a set of data strings. This represents a more concise and elegant solution using a minimum amount of data for maximum value.

4.2.1 The Branching Structure: Code

Mouse movement previously used to control the line drawing is replaced by a new class, the `Target` class. The `Target` object is a single moving object whose path and movement are dictated by simple iterative calculations of angle and position based on a sine curve, in a similar way to the `Colorcalm` project. The movement and the path of the `Target` object replaces mouse movement in leading the `Line` and defining its path, shape, direction, and growth. The 'length' of the `Target`, i.e. how long it is on the screen, and its movement therefore define the length, shape and form of the `Line`.

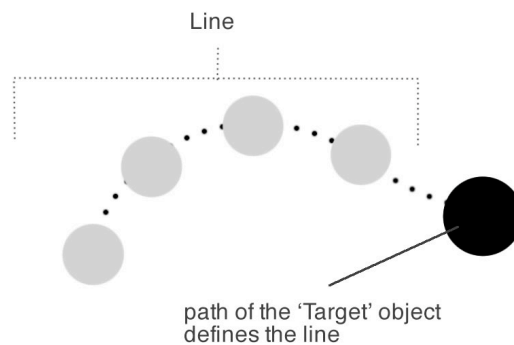


Figure 4.6 A diagram illustrating the relationship between the `Target` and the `Line` objects in the `Moving Wallpaper` project.

Whilst the path and shape of the `Target` class are handled by its own internal function and variables, the logic of the branching system is controlled by string data. Each `Target` contains a string, i.e. a set of characters, or letters put together. Each letter in the string is accessed one at a time and interpreted to influence the `Target` object. Each character in the string represents a specific function which is defined and set within the new `Engine` class. This rule based system allows branching structures to be created and is based on the concept of the L-system, as outlined previously. Each time the system encounters a 'B' character as part of the string, a new branch is formed. The individual functionality of each specific letter ('characters') defined within the `Engine` class, used to affect the growth structure of the work, is outlined here:

Character	Description of Function	Engine Class code
F	adds more to the timer of the Target (grow more).	<code>t.timer += 100;</code>
B	creates a new Target instance (branch).	<code>t.branch ()</code>
>	returns back to the start of the String (loop).	<code>t.timer = 0;</code>
*	stops growth (end and remove the Target).	<code>t.remove ()</code>

Table 4:5 Summary of characters allocated to affect growth and structure: Moving Wallpaper

The result is that different strings are able to *describe* different characteristics of the line and different types of branching structure. For example:

Example String	Description of Line
"FFFFBFFFF"	a long line, lots of growth with one branch.
"FBFBFBFBFB"	shorter line with 4 branches set at equal intervals.
"FFFFFFBFFFBFFBFB"	long line with 3 branches set at different intervals.

Table 4:6 Example Strings and description of associated visual effect: Moving Wallpaper.

4.2.2 The Branching Structure: Visuals

The line structure is now one of inter-connected branches. Each line is rendered as a chain of interconnected balls. At this stage, the emphasis is on the behaviour of the line, the development of the branching structure and the ability to maintain inter-connectedness between the lines and 'branch' lines; rather than on the aesthetic qualities of the line.



Figure 4.7 Screenshot from 'LinkingLinesTest2' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/2_branches/linkingLinesTest

There is already a clear connection between the key elements of the visuals, (its shape, structure and behaviour) and the class structure. The relationship between the class and the visual attributes of the lines can be summarized as follows:

Visual Element	Class	Description
Shape	Target	Defined by movement of Target object.
Structure (branches)	Engine	Handled by Engine, interpretation of String data.
Behaviour	Spring	Forces controlled by Spring class.

Table 4:7 Overview of the relationship between visual elements and individual classes:
Moving Wallpaper.

4.2.3 The Branching Structure: Process

The development of the class structure and addition of the `Target` and `Engine` classes, continues to broaden the overall structure and the concept of the work. Maintaining a focus on the use and flow of data and the interconnectedness of separate classes allows a fluid, flexible type of material and structure to develop. The use of string data to encapsulate branching structure logic is one which supports a flexible system in which complex logic can be simply described. The use of characters to describe the structure of the line adds an interesting element of grammar to the work, which can be literally used to describe the nature of the structure.

Important themes for this work are starting to emerge. Notions of *simplicity* (finding the simplest, most elegant solution), *integrity* (creating a solution which works 'in harmony' with the rest of the data elements) and *flexibility*, (broadening and extending the concept and data structure in order to create the widest range of visual outcomes) are all key to the development of this project. A core aim and intention for this piece of work is emerging: to develop a computationally elegant, flexible and fluid structure, one which is able to create a wide variety of organic forms within the parameters of its data structure.

4.3 Visual and Behavioural Experimentation

Having established the foundational elements of the concept, data structure and behaviour, the next stage of the work continues to explore visual and behavioural qualities of individual lines and the overall growth and structure of the work.

4.3.1 Visual and Behavioural Experimentation: Code

No new classes are added during this phase of work, functionality is developed within the existing classes, and emphasis is placed upon exploring the potential of the current structure. Initial experiments develop the shape and width of the line. A `calcBallWidth()` method is added to the Line class and sine calculations are used to calculate the width of each ball within the line in order to generate soft gentle forms for the overall line shape.

```
width = 180 / ballcount;  
key variables : ballcount and magnitude
```

Subsequent adjustments to the `calcBallWidth()` calculation alters the nature and the quality of each line. Altering the core sine calculation produces a range of visual results. For example, the difference between a 180 degree sine curve and 90 degree sine curve can be clearly illustrated:

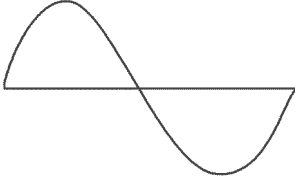


Code	Shape
<code>width = 360 / ballCount</code>	
<code>width = 180 / ballCount</code>	
<code>width = 90 / ballCount</code>	

Table 4:8 Overview of sine wave affecting shape: Moving Wallpaper.

Setting and changing the magnitude (`maxWidth`) value within the body of the sine calculation alters the maximum breadth of the line.

Changing the width value, or the sine curve ending value is a simple way to alter the visual quality of *all* lines when the programme is run. Creating variety within the same system, (i.e. defining and controlling specific lines with their own width of sine value) is, however, more difficult and requires changes and additions to the computational logic. Conditional statements are used to create difference and variety between 'branch' and 'stem' lines. Differentiation between lines (i.e. establishing whether it is a 'stem' or a 'branch') is created by comparing the `parentLinePos` value of the each new Line:

```
if (parentLinePos == 0 ) {
    maxWidth = 2 // the line is a 'stem'
}
```

Conditional statements are also used to control and determine the growth of new, 'stem', lines which generate entire new 'plants'. In order to allow the drawing programme to continue generating new plants, a new method, `newLine()` is added to the `Line` class. A conditional statement is added in

to the class to control the timing of the new line when an internal timer reaches a given number:

```
if (time == 100) { addLine ( ) }
```

As well as altering the visual appearance of the line additional methods are included to the `Line` class as ways to alter the behavioural elements of the line. A `calcForces()` method is created to re-calculate the position of each `Ball` in relation to its proximity to the mouse. This method is used to calculate the distance and angle between each ball and the cursor on screen: It uses these values to repel each the ball away from the mouse, giving the line a 'reactive' quality. This can be illustrated as follows:

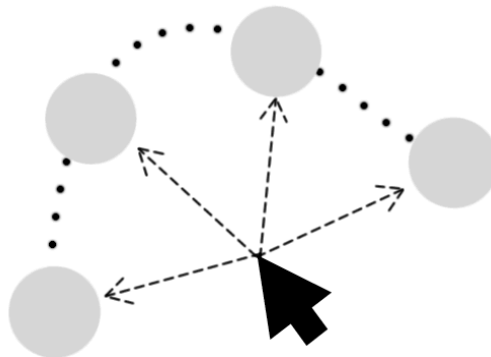


Figure 4.8 A diagram illustrating the movement of the line away from the cursor in the Moving Wallpaper project.

4.3.2 Visual and Behavioural Experimentation: Visuals

Emphasis during this stage of the project is on visual and behavioural experimentation; modifying the existing data structure in order to sculpt change, add nuance and develop the controlled randomness of the line. The visual aspect of the work begins to take shape as basic organic forms,

shapes and curves are created. The ability to change single variable values is the first means of adding variation and control to the lines shape and width, and produces pleasing results as the sine curve is used to generate smooth elegant shapes and lines. The scope for modifying the specific attributes of the lines can be summarized as follows:

Visual Change	Description of Code
change ending of line	modify the <code>calcBallWidth ()</code> formula (change 180 to 90).
adding growth and decay	modify <code>calcBallWidth ()</code> method.
infinite growth	call <code>newLine ()</code> based on timer or <code>lineWidth</code> .
define widths of lines (leaves or stems)	use <code>parentLinePos</code> variable to distinguish between different types of lines.

Table 4:9 Overview of visual changes affected by individual elements of code: Moving Wallpaper

The behavioural development of the work adds a simple 'reactive' quality to the lines. In keeping with the notion of the 'living line' theme, each line reacts to the movement of the mouse, creating an effect in which the lines (stems, branches and leaves) appear to sway as the mouse moves over them. The addition of this simple, subtle interaction, together with the tension between each `Ball`, generates a springy, reactive behaviour and a more visually fluid piece of work. The overall effect is therefore that of a 'reactive', living, flexible line; a concept which works in harmony with the overall theme and structure of the work.

The following are examples of experiments with different renderings of the reactive lines:



Figure 4.9 Screenshot from 'LinkingLinesTest3' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/linkingLinesTest3

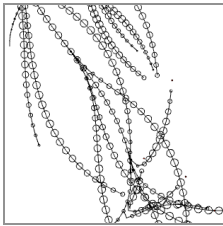


Figure 4.10 Screenshot from 'LinkingLinesTest3_outlines' (2008) from:
http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/LinkingLinesTest3_outlines

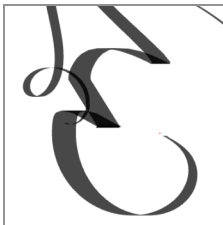


Figure 4.11 Screenshot from 'Line_Forces_Test_LRG' (2008) from:
http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/Lines_Forces_Test_LRG

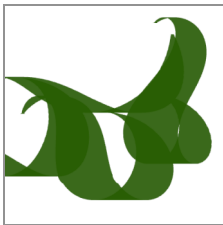


Figure 4.12 Screenshot from 'Line_Forces_27Feb' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/Lines_Forces_27Feb

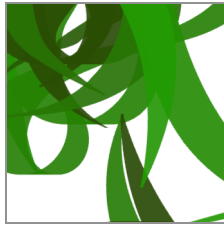


Figure 4.13 screenshot from 'Line_Forces_27Feb_new' (2008) from:
http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/Lines_Forces_27Feb_new



Figure 4.14 Screenshot from 'MovingWallpaper_March1_b' (2008) from:
http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/MovingWallpaper_March1_b

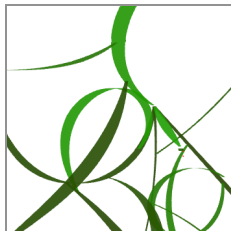


Figure 4.15 Screenshot from 'MovingWallpaper_March1B' (2008) from:
http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/MovingWallpaper_March1B



Figure 4.16 Screenshot from 'MovingWallpaper_March1E' (2008) from:
http://www.random10.com/movingwallpaper_research/applets/3_visualExperiments/MovingWallpaper_March1E

The mouse as trigger for movement may in future versions of the work be substituted for other means of interaction e.g. the movement of a figure moving past the work. The application the class-based system means that the core functionality is created and can be extended to include other elements and items.

4.3.3 Visual and Behavioural Experimentation: Process

Changing and modifying individual variables within the structure affects the detail and the fine tuning of the work. Single variables are created to represent a specific visual aspect of the line (e.g. `maxWidth`). There is a direct correlation between the value of the variable and the specific detail of the visual. Modifications made to the value of the variable naturally affect elements of the line. Assigning one variable to one visual element creates a clear, simple and easily identifiable connection. Variables define the properties of individual visual details, their affect can be easily seen and understood. There are, however, issues and problems relating to the use of single variables. Access to variables is difficult as they are often 'buried' within the code. Clarity of what the variables do is often obscured, thus making change or building elements to change these data elements tricky.

Calculations and variables, which affect the shape and form of individual lines of the Moving Wallpaper piece of work, are very similar to those used in the Colorcalm project. Both use similar sine-based calculations to describe shapes. The significant difference between the projects is the conceptual, structural and data elements of the work. Seeing the similarities between the projects allows the significance of the differences to be observed. The impact of the changes in structure and data may be observed.

The use of conditional statements to add variation to the work, (e.g. for determining the timings of new plants, as well as the widths of leaves and

branches) offer a limited means for adding variance and nuance. The use of the conditional statements to control line width, for example, offer a choice with little scope for subtlety and nuance, and makes accommodating other types of lines difficult. The condition, which defines the line width according to the value of an 'obscure' `parentWidth` variable, is not *logical* and has little consistency with the core data system. Similarly, the conditional statements which control the re-generative nature of the work, offer only limited control of this element of the code, making clumsy associations between the width of the line and the growth of a new plant. Reliance on timers, internal clocks and counters is a rather unsubtle, 'mechanical' solution which lacks nuance and flexibility.

4.4 Extending the Vocabulary

A fundamental concern of the work is its ability to express and create a wide range of decorative, organically inspired, shapes and forms. However having developed core elements of behaviour and structure, the project at this stage only allows for the creation of a narrow range of plant shapes and forms. This phase of the project, therefore, aims to develop the visual vocabulary of the work. Detailed examples of code for this section can be found in appendix 2.4

4.4.1 Extending the Vocabulary: Code

Differentiating between the visual properties of each line, a problem for the Colorcalm system, is handled by the introduction of a new class. The `Attribute` class is created to group lots of key data elements, i.e. properties containing details of the visible attributes of a line, e.g. colour, width, string data, into a single set of 'attributes'. This a class with virtually no functionality and is used primarily as a container; a means of 'data storage',

providing the ability to condense lots of different data variables into a single instance. Each new `Attribute` instance is given a name which describes the type of line it is to represent e.g. 'Stem', 'Flower' 'Branch'.

```
Attribute = new Attribute (width, string) ;  
Branch = new Attributes (5, branchString);  
Stem = new Attributes (1, plantString);  
Flower = new Attributes (10, flowerString);
```

In this way a unique set of attributes can be assigned to 'branch', stem and leaf lines. An `Attributes` object is included in each new `Target` object and is included in the `Target ()` constructor method:

```
Target (atts, loc, angle, parent, depth)
```

When a new line is created, a specific set of `Attributes` are attributed to it.

```
addBranch (Attributes a) { }
```

The `Engine` and the `Target.addBranch ()` methods are altered to incorporate the name of the `Attribute` to be added to the new line.

```
B = t.addBranch (Branch);  
f = t.addBranch (Flower);  
L = t.addBranch (Leaf);
```

Assigning individual letters to represent specific functions and instructions is another concise way of formatting sequences of rules and conditions which may otherwise be difficult to express. The concept of the L-system in which single characters represent functions and rules defined within the programme, has to this point been used to accommodate only a narrow range of letters and functions. The ability to extend the range and scope of the `Engine` class offers potential to extend the visual and computational

vocabulary of the work. Many more elements of the line structure (leaf, branch, stem) can be defined and greater degree of control, flexibility, subtly and nuance can be introduced to the rule based structure. The `Engine` class is extended to accommodate additional vocabulary and functionality. Additions to this class can be summarized as follows:

Letter	Description	Sample Code
P	create a new Plant	<code>t.addPlant ()</code>
+	increment angle	<code>t.angle += leafAngle</code>
-	decrement angle	<code>t.angle -= leafAngle</code>
L	add new leaf	<code>t.addBranch (Leaf)</code>
f	add new flower	<code>t.addBranch (Flower)</code>

Table 4:10 Overview of new characters used in Engine class: Moving Wallpaper

In this way a string of otherwise meaningless letters becomes an expression of the nature and structure of the plant; a succinct set of commands open to subtle change. The development of the string and the `Engine` class generates a more 'fluid' structure than the conditional statements used in the `Colorcalm` project, a natural development of the data structure.

Further modifications and additions made to the `Attribute` class enable greater visual detail to be included as part of the definition for each line. Colour data elements are added allowing sets of colour values to be attributed to each line type.

```
Attributes = new Attributes (width, String, colour1, colour2);
Branch = new Attributes (5, branchString, color (255, 0, 0),
color (125,10, 12));
```

Defining two colours rather than one means that each line type is allocated a colour range. The `colorBlend()` method is used to pick a colour from somewhere between two colour values (`c1` and `c2`) and to use this as the `fillColor` for the line.

```
Attributes.colorBlend (c1, c2)
fillColor = (Attributes) lineType.colorBlend (c1, c2);
```

The `Attribute` class, at this point, can be summarized as accommodating the following set of variables:

Attributes:
width
string
colour1
colour2

Table 4:11 Key variables of Attributes class: Moving Wallpaper

Creating an `Attribute` object with differing values for each variable can be used to distinguish between types of lines, for example a flower and a branch:

Attributes:	Flower
width	8
string	"FF"
colour1	(0, 112, 0)
colour2	(0, 256, 0)

Table 4:12 Example of Attributes, Flower: Moving Wallpaper

Attributes:	Branch
width	4
string	"FFBFFf"
colour1	(0, 112, 0)
colour2	(0, 256, 0)

Table 4:13 Example of Attributes, Branch: Moving Wallpaper

A new type of `Attribute` is created; a 'parent' class which allows collections `Attributes` to be grouped together, creating another means of reducing the code, making computational descriptions increasingly concise.

The new parent object is accommodated into to the `Attribute` class via a new constructor method which only takes one parameter:

```
Attribute (string) ; // parent attribute object.
```

This example demonstrates how the `Flower` attribute is used as into a 'container' of `Petal` lines:

Attributes:	Flower (f)
width	--
string	"p-p-p-p-p-p"
colour1	--
colour2	--

Attributes:	Petal (p)
width	12
string	"FF"
colour1	(0, 112, 0)
colour2	(0, 256, 0)

Table 4:14 Example of how 'sub' Attributes can be created: Moving Wallpaper

'Parent' attributes are also used to create 'segment' and 'end' attributes, which group together entire strings in order to represent a complete section of a line. Thus 'segments' (s) define sections of the main part of a line, and 'ends' (E) define the structure at the end of a line. Segments and ends are added to more succinctly describe the structure of an entire line. Thus a branch string which may previously be written as follows:

```
B = "FFBFFBFFBFFBFFBFFBFFBFFBff"
```

can be simplified as follows, to become a series of repeating branch 'segments' (s) and a endings (E) as follows:

```
B = "SSSSSE"
```

```
when:
```

```
S = "FFB"
```

```
E = "ff"
```

In this way each 'S' and 'E' is used to summarize a group of letters. This is a more succinct way of describing a line, allowing greater, easier and more eloquent control of structure.

The grammar of the line is now simplified; a complex series of branching, structural decisions are summarized into a set of grammatical instructions (e.g. "SSSE"). The ability to 'reduce' the definition of a line into few of 'segments' (S) and 'ends' (E) encourages one final development of the code structure during this phase; replacing the single string with an 'array', a choice of strings. Replacing a single string with a choice of number of different strings generates greater variety between each segment and ending elements of the line. Rather than having the same defined 'segment' or 'ending', each segment or ending is randomly selected from a list of different segments and endings. The following gives an example of two lists (or 'arrays') of segments and endings; `segArray` and `endArray`:

```
segArray = {"FF", "FLLF", "FFF"};  
endArray = { "FFF", "FLL", "FFff"}  
  
Segment = new Attribute (segArray);  
End = new Attribute (endArray);
```

Each segment (S) and end (E) therefore selects a string from one listed in the array. The Engine class is modified to include a method which selects one string from the array: `Engine.convertString()`. This is called from the Target constructor:


```
s (String) = e.covertString (atts.s);
```

This concept is extended as arrays of strings are used for other types of lines. For example a `leafArray ("F", "FF")` is added to randomize the size of leaves. The idea of allowing difference and variation within the branching structure of the line is important and one which came from studying the work of botanical forms used by William Morris, in whose work the controlled variety of organic shapes form a key element of the design.

4.4.2 Extending the Vocabulary: Visuals

The process of developing and extending the vocabulary of the work yields important changes in the visual aspects of the project. The beginning of the ability to differentiate between different elements of the visuals allows the development of a wider range of shapes and forms. The botanical structure of the work is developed to include combinations of leaf, flower, branch, sub-branches and stems:



Figure 4.17 Screenshots from 'MovingWallpaper_March12_BASIC' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/4_ExtendingVocab/March12_Basic

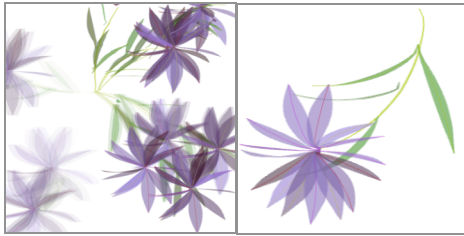


Figure 4.18 Screenshots from 'MovingWallpaper_April1_C' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/4_ExtendingVocab/aprilc/

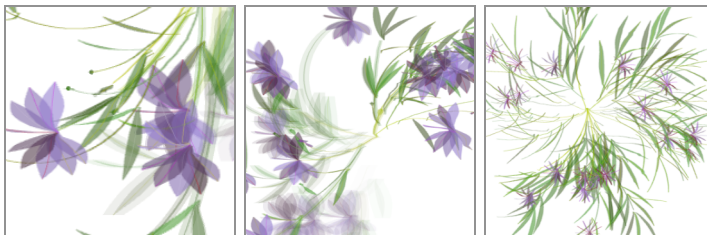


Figure 4.19 Screenshots from 'MovingWallpaper_March19b' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/4_ExtendingVocab/March19b

4.4.3 Extending the Vocabulary: Process

The combined effect of extending the vocabulary and choice of strings, together with the introduction of the `Attributes` class provides a computational data system which can be used and extended to create a greater range of shapes, forms and lines. It is a computationally concise, flexible and elegant system for describing and defining lines and their structures.

The system is *concise*:

The `Attributes` class encapsulates core data about a line. Each letter of a string (`F`, `B`, `f`, etc.) represents a single function. A string of characters therefore concisely encapsulate a 'stream' of logical commands and functions. Lines are described by a series of strings. A "FFFFB" line (some growth plus a single branch) would therefore have a different structure to a

"FBFBFBFff" line (three branches at regular intervals, ending with a single flower). The addition of the 'parent' `Attribute` classes adds an extra degree of concision, allowing a greater level of detail to be expressed even more succinctly. A flower, for example is expressed more eloquently as "f" rather than as a string of petals; "ppppppppp". The use of specific names, ('Stem', 'Leaf', 'Branch') to describe each of the `Attributes` instances creates an important natural-language link between the abstract data and the type of the visual aspect of the line described. This use of language makes the passing data more obvious, accessible and 'natural'.

The system is *flexible*:

The concise nature of the vocabulary based system means that any slight change to a string will modify and add nuance, to the structure of a line. Simple adjustments to the string of a `Plant` or the 'rules' of the `Engine` allow a greater range of lines to be produced. The programme is therefore able to create a variety of lines and define new types of line by extending these rules. Once the basic `Attributes` class is established new 'types' of lines can easily be described and defined as with their own data types; Leaves, Flowers Branches, etc; each can be defined with their own set of properties. The use of lists of string arrays adds additional nuance and subtlety to the work, resulting in different combinations of line 'segments' and line 'ends', further developing the idea of controlled variety. The *flexibility* and *pliability* of the work in allowing specific details for each type of line to be created, and the ability to define a wide variety of lines, is an extension of the original OOP class-based structure.

The system is *computational* (abstract, data based, rule based):

This project shows a greater use of and understanding for the core elements of computation, allowing the broader concepts of programming to be used effectively. The use of strings (i.e. within the rules of the L-system) and classes (OOP) to define the visual structure and parameters of the line

generates a more computational and abstract solution. Combinations of data created using strings and classes define large elements of the work. The broader use of data elements allows a broader and more abstract definition of objects which includes a greater degree of complexity and detail. The work therefore reflects an increased ability to understand, use and *work with* the creation, definition and flow of data and abstract computational concepts. Unlike the previous project, the system has computational integrity; the structure is developed *with* the core concepts of the programming data structure.

The system is *elegant*:

The flexible, concise computational description of lines maintains a degree of integrity and elegance. The ability to change, modify, add subtlety and nuance to the description of structures and visuals provides a greater degree of visual and computational elegance.

4.5 Translation and Rotation: Adding 3D Elements

By drawing reference to traditional ideals of beauty and form, the visual and aesthetic direction of the project has been concerned with creating an environment which allows the simplicity, and beauty of plant and flower patterns to grow, form and develop across the screen. The emphasis on organic structure and growth has been reflected in the development of the code. A flexible data structure has been developed to allow a broad range of elements (leaves, branches, flowers, etc.) to be defined and randomized, within a tightly defined data structure, enhancing the core ideals of beauty and structure. String data defines the structure of the work to allow digital plants to grow, develop, react and fade. This phase of the work begins to add visual detail and beauty within the set structural parameters of the project: Elements of 'growth' and 'decay' are added to the line, as an 'unfurl' element is included to enhance the naturalistic movement and form of the

plants and flowers. Detailed examples of code for this section can be found in appendix 2.5.

4.5.1 Translation and Rotation: Code

The code developed for this phase of the project focuses upon the relationship between each of the balls which constitute each segment of a single line. Creating a calculation to rotate each ball through the z-axis, the flat, two-dimensional line is transformed into a line which begins to show attributes and movement in three-dimensions. A 'translation' calculation is developed, as a separate sketch, to calculate the movement and rotation of points around each other. The rotation amount around each point is calculated according to sine calculations on each of the x, y and z axes. The translation calculation is added into the body of the Moving Wallpaper project, as an `unfurl()` method. The calculation uses a set of variables which determine the amount of rotation or 'spin' around each segment as well as the 'target' rotation value towards which the line moves:

Variable	Description
<code>spin</code>	The initial rotation of the line (angle of the petal)
<code>spinInc</code>	The subsequent rotation for each point of the line This affects how curved the leaf / petal is
<code>unfurlTarget</code>	The final rotation for the entire line For example: -16 = a fully opened petal 12 = closed petals (rotate in towards centre)

Table 4:15 Overview of variables used for 3D rotation of line: Moving Wallpaper

The `unfurl()` method continually re-calculates the amount of rotation needed until the line reaches its target rotation.

```

void unfurl ( ) {
    float spinIncDiff = unfurlTarget - spinInc;
    spinInc += spinIncDiff *0.008;
}

```

Setting the amount of 'spin' and 'spinInc' determines the amount by which each flower and petal is unfurled.

```

Line ( ) {
    float spin; (initial rotation value)
    float spinInc; // how much each segment is incremented (to get
    a curve)
    float unfurlTarget = 2;
}

```

Initially the unfurl() method is applied only to the petal lines allowing the flower to open and 'bloom':

```

void fineParents ( ) {
    ...
    if (lineType.type == "petal") {
        spinInc = 20;
        spin = 60;
    }
    else {
        spinInc = 0;
        spin = 90;// random (40, 100);
    }
}

void run ( ) {
    ..
    if (lineType.type == "petal") {
        unfurl (); // rotate line around it's axis
    }
}

void unfurl ( ) {

```

```
float spinIncDiff = unfurlTarget - spinInc;
spinInc += spinIncDiff *0.008;

} } }
```

4.5.2 Translation and Rotation: Visuals

The introduction and use of the `unfurl()` method re-emphasizes the organic fluidity of the visuals and the structure; providing an extra dimension to the work which is in-keeping with the organic growth and data flow of the project. The addition of the rotation and translation detail gives each of the lines a greater feeling of growth and decay; emphasizing the organic movement and 'evolution' of the plants. Initial visuals were experimental, technical tests of the 'unfurl' idea:

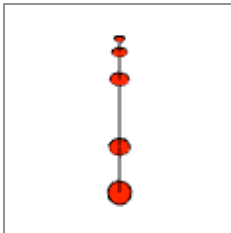


Figure 4.20 Screenshot from 'translateExample12C' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/5_Translation/translateExample12c

Functional elements from the test pieces were added into the main body of work to produce visuals which begin to show more organic attributes of growth and decay as the petals smoothly open and close.



Figure 4.21 screenshot from 'Moving Wallpaper_June13_LRG2Color' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/5_Translation/MW_June13_LRG2color



Figure 4.22 Screenshots from 'Moving Wallpaper_June23_unfurl_shape3' (2008) accessed from:
http://www.random10.com/movingwallpaper_research/applets/5_Translation/MW_June23_unfurl_shape3

4.5.3 Translation and Rotation: Process

The simple visual fluidity and elegance added to the work by the rotation of the petal lines adds an interesting aspect to the work. Generating the calculation however proved to be a little tricky and relied upon the understanding and use of calculating the position of objects in 3D space. Using a separate sketch to create the calculation was useful. The translation calculation was easily incorporated to the `Line` class, but required a number of extra variables to control the movement. Use of single variables and conditional statements adds some complexity to the line class, the unfurl movement is controlled by the combination of a number of different variables and conditional statements. The visual effect is good, adding a degree of visual subtlety to the work.

4.6 Final Development and Variations

A key element of this project is the concept of visual and computational *flexibility* used for creating organic, variable growth. During the evolution of the project, decisions regarding the concept and structure of the programme have been made in order to accommodate flexibility and fluidity as part of the data structure. The `Attributes` class groups many individual variables representing the visual properties of a line into a single, definable set, or 'instance', providing a way in which the properties and attributes of each new line can be easily defined, accessed and managed. Similarly the use of string data to encapsulate and condense the otherwise complex logic of conditional statements into a single set of characters creates a flexible, accessible structure which allows for change, nuance and subtle alteration.

This final phase of work continues to test and stretch the system in terms of its flexibility and variability, adding a further layer to the data structure in an attempt to extend the amount and the variety of visible attributes which can be produced. A new class, the `Plant` class, is created into which groups of data are collated e.g. `Attribute` instances and other single numeric variables. A `Plant` object is therefore defined as the collection of lots of different types of data and line attributes: a means of combining branch, flower, petal and stem attributes together into a single group. This therefore introduces a whole new series of data elements and details which are able to add subtle difference variation and greater flexibility into the structure.

4.6.1 Final Development and Variations: Code

The `Plant` class is defined along the same lines as the `Attributes` class; the class is has no real functionality other than its main purpose of grouping sets of data together. The initial version of the plant class is used to group together a set of `Attributes` which constitute all the different lines within a

Plant structure (i.e. Stem, Leaf, Branch and Flower). The basic Plant constructor function is written as follows:

```
Plant (Stem, Branch, Leaf, Flower, Petal)
```

The Plant class therefore becomes a container for a group of specific line Attributes. Additional variables are subsequently included to control and define other visual elements of the plant structure:

Variable	Description
lo_ & hi_	controls the minimum and maximum sine curve values and the shape of the plant lines.
petalCount	controls the amount of petals of a flower.

Table 4:16 Additional variables added to control visuals: Moving Wallpaper

Subsequent versions of the Plant class add an increasing amount of detail. A second Plant constructor is created to accommodate the added variable elements:

```
Plant (Stem, Branch, Leaf, Flower, Petal, lo, hi, petalCount)
```

An added level of the data structure is developed and the Plant class is used to define an increasingly huge variety of attributes and variables which control and define many different attributes of a 'plant'. The following table outlines the core set of variables included in the Plant class and which are used to define specific elements of each plant.

Plant Class		
Variable	Initial Value	Description
decayRate	0.3;	controls the minimum and maximum sine curve values and the shape of the plant lines.
refurlRate	0.02	controls the amount of petals of a flower.
blurRate	0.05	controls speed of blur, if used.
decay	false	determines whether flowers 'die' and decay.
flowerBlur	false	defines whether blur effect is applied to flowers.
leafBlur	false	defines whether blur effect is applied to leaves.
drawLines	false	defines whether outline strokes are used.
blurAmount	10	defines amount of blur.
blurAlphaFraction	0.2;	defines percentage amount of blur transparency.
blurNum	1	defines the 'amount' of blur.
leafFollowParentColor	true	defines whether all leaves are same colour.
flowerFollowParentColor	true	defines if all petals on flower are same colour.
leafGrowth	0.95;	how much leaves grow.
leafStart	2.0f	starting size of leaf.
flowerGrowth	0.7;	how much subsequent flowers grow.
flowerStart	1.0f	starting size of flower.
flowerDev	{ -4, 15 }	min and max amounts flower can deviate.

Table 4:17 Overview of Plant class variables: Moving Wallpaper

The use of the `Plant` class as an over-arching container for all the attributes and properties of each plant structure is a convenient and easily changeable means of setting and defining lots of visual properties which can be applied

to each new output of the programme. The structure of the code is now established; the rest of the work is concerned with experimenting and manipulating the functions and data structure to see how much visual variety can be created.

4.6.2 Final Development and Variations: Visuals

Up until this point, the structure and parameters of the visual elements of the work have been created but the full range and visual potentiality of the structure has not been fully explored. A large amount of emphasis has been placed on developing and defining a flexible, usable data structure which can produce the widest possible variety of visual forms. Having reached a stage with the work which most fully expresses the level of visual detail and behaviour possible, the visual expression and variety of the project is explored in a much fuller and wider way. Initial experiments begin to test the potential for generating a range of shapes and forms:



Figure 4.23 Screenshots from 'MW_June27_PlantTypes' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/6_FinalDevelopStages/MW_June27_PlantTypes



Figure 4.24 Screenshots from 'MW_July1_PlantShapes2' (2008) accessed from: http://www.random10.com/movingwallpaper_research/applets/6_FinalDevelopStages/MW_July1_PlantShapes2

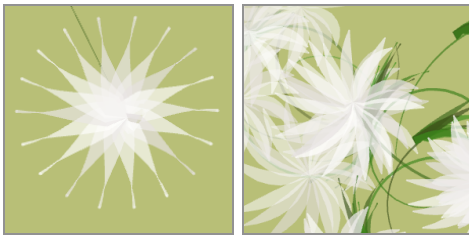


Figure 4.25 Screenshots from 'MW_July2' (2008) accessed from: http://random10.com/movingwallpaper_research/applets/6_FinalDevelopStages/MW_July2

Following these experiments a further, wider range of work exploring the visual possibilities was produced.

Botanical structure and form remained a strong influence for the visual development of the work throughout the project, reflecting the emphasis that the Arts and Crafts movement placed upon natural forms in their design. William Morris stressed the importance of knowing and observing natural form before 'conventionalizing' it into ornamental pattern or design (Morris, 1882). The emphasis that Morris placed on the observation of natural form is applied here to the computational environment. The following photographic images illustrate the observations and references made to a variety of plant and flower shapes, in developing a range of visual form for the Moving Wallpaper project.



Figure 4.26 Researcher's own plant and flower photographs used as visual reference.

Using these observations and photographs as source material and giving close consideration to the shape, colour, and variety of each form, the Moving Wallpaper project then attempted to abstract ('conventionalize') the plant shapes into a range of computational designs. The process of generating these computation forms can be seen as a type of what Morris described as decorative conventionalization. The shapes and forms are simplified to their essential structure and then abstracted into individual configurations of data detail. Each plant shape is converted into a data string which determines branch, leaf and flowers properties; adjusting individual variables to add nuance of shape and levels of variation. The colours, structure, shape and form of plants, observed and photographically recorded, were used as visual reference for creating and extending the visual range of the work. The following page (fig. 4.27) shows the variations produced. Comparison between the photographs (fig. 4.26) and the computational versions (fig. 4.27) illustrates the way in which the original forms and shapes are abstracted and simplified into the computational structure.

Figure 4.27 Screenshots of the final variations of Moving Wallpaper work accessed from:
http://random10.com/movingwallpaper_research



Having set and established the core structure of the work, this final stage places emphasis on manipulating variable values, string data, etc. to set and achieve different types and shapes of plant. Usage of the `Plant` class and the `Attributes` class, together with the usage of string data all combine to define the individual shape, form, colour, structure and general attributes of each element, allowing a greater level of control over the detail of the work. Setting individual attributes is easier than in the Colorcalm system, however there are still many details to be set and used to control and define each visual element of the work. The final structure is one in which all the visual details of the work can be set and changed. This not only includes the broad elements regarding number of leaves and flowers but also the detail, for example, the amount of 'randomness' in the line to control leaf curls, and the speed at which plants grow or decay. The system is one of 'controlled variety', in which each element of work can be 'randomized', but done so within a set of defined parameters. Every type of line, leaf, stem branch petal and flower is assigned variations which define its growth, speed, form, shape size, etc. within set parameters, which define its 'randomness'. The amount of detail in the system affords a great deal of control over the element of the work. The process during this final phase is therefore one of experimentation and 'play' with the data and variable values. This can be a detailed and time consuming process. However, given that the structure is less cumbersome than that of the Colorcalm work, making changes and defining details of the work is less difficult, allocating and finding specific data to define specific details is more easily done.

4.6.4 Summary

The Moving Wallpaper project represents the second phase in the researcher's movement towards the use of programming code as a means of creative design. In comparison to the initial Colorcalm project, it highlights improvement and development regarding the use and understanding of the

computational material, reflected by more fluid and flexible structures and visuals.

i. Code

The project represents a piece of work which is based more firmly around broader computational concepts and structures than was the case with the Colorcalm project. The approach to the project, right from the initial phase, emphasizes a way of working with code which reflects a clearer understanding of computation as an abstract data structure. Rather than being initially concerned with the visual details of the work, the Moving Wallpaper project begins by considering the computational, conceptual and behavioural elements of the line, i.e. how it is structured, how it may behave, grow and develop. The project abstracts the concept of the line into a broad structure of inter-connected classes (*Ball*, *Line*, *Spring*, *Target*) which represent individual conceptual and behavioural elements of the project. This class structure represents a clearer abstraction of the work into related concepts which provide the ability to add and develop the functionality of the work. The strong, flexible, modular structure defines the allocation and flow of data and represents the function structure and behaviour of the work, rather than just its visual properties.

ii. Process

Recognition of the importance of data allocation and flow in this project is illustrated by the use of string data, summarizing and condensing the complexities of an organic branching structure into a series of individual characters, each representing a visual element of the plant growth. The ability, for example, to use "FFBFBFf" as a means of describing a plant with two branches (B) and a flower (f) highlights the developed flexible vocabulary of the system. The flexible computational structure of the Moving Wallpaper work is therefore one in which data is described concisely and is allowed to flow freely through the system. The work is both structurally and

visually much more flexible than that of the Colorcalm work; elements of variance, randomness and visual nuance are able to be added to the code and to the visuals of the work.

The structure affords a greater range of visual expression and development; details of each element of the work can be accessed and changed more easily. For example, simple changes to the string data or to elements of the `Plant` class can easily alter the visual detail of the work. Describing a wide variety of shapes and plant structures is therefore easier, and visual nuance and subtlety can be added to specific leaf, branch, petal or flower elements of the work. The process of the work is generally more intuitive and flexible than that of the Colorcalm work.

iii. Visuals

Visual fluidity and flexibility is evidenced by the 'reactive' nature of the work. The behavioural organic structure of the line with its naturalistic movement, growth and decay is defined and supported by the data structure which allows for development of the work. External input (e.g. mouse movement) can be used as a means of making the visuals move and react. Overall, the work represents the development of a flexible data structure which is computationally and visually more fluid affording greater access to the process of visual and structural development and experimentation.

5 Overview and Analysis

5.1 Introduction

The contextual review (chapter 2) considered the similarities and differences between traditional, software-centred and computational design, looking specifically at the ethos, material, process and object. The review highlighted the idea that whilst both software and computation share a common digital environment, the ethos and approach toward process and material reveal key areas of crossover between traditional and computational work. It was suggested that the process of computational work is informed and motivated by ideas and approaches which link it to ideas and motivations of the Arts and Crafts movement. Using the initial review as the context for practical work, the research then considered the computational process in more depth: looking specifically at the development and use of code as a material for creating visual design work. This chapter will therefore provide an overview and analysis of the practical project work considering both the Colorcalm and Moving Wallpaper projects highlighting key elements of the computational process for each piece of work. Although both projects produced a range of visual work, this chapter does not attempt to make an evaluation of the aesthetic success of each project; the visual outcomes of each project when considered are done so in the context of, and in relation to, the overall process and workflow.

5.2 Computational Process: A Unique Workflow

At the heart of this discussion is the desire to understand the process of using computation as part of creative practice. Having completed both pieces of work, conclusions can begin to be made about the overall process

for the project work and the generic workflow of each. Consideration of practice during the development of each project has highlighted similarities in the broad approaches to the computational process, but differences in emphasis and development of each of the pieces of work. Reflection of the Colorcalm work during its lifecycle informed the approach and development of the subsequent Moving Wallpaper work. This chapter will begin by making comparisons with both traditional and software-driven design, highlighting ways in which computational design can be seen to have its own unique workflow and design process.

The process of contemporary software-centred, digital design has been outlined in this research (2.1.4) as a cut-and-paste, re-mix and re-edit approach to creating work; a process defined by the movement and combination of image data (text, graphics, sounds, etc.); a loose process of "import and export" (Manovich, 2006) in which design elements are moved and shared as data objects between a network of software packages. This import / export network association between different software packages is illustrated as follows:

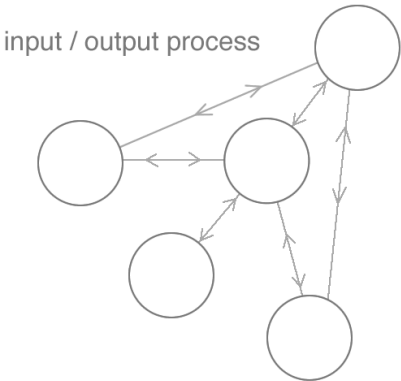


Figure 5.1 A workflow diagram illustrating the network flow between different pieces of software

In contrast to the copy-and-paste approach of software-driven design, the research (2.3.3) has also highlighted how the process of traditional design practice can be characterized as a formal, skilled process of interaction between maker material. The traditional process of design can be defined as a variation of the design cycle, a process of design, creation and evaluation in which work is developed by a 'feedback' process during which making is informed and generated by the process itself. Ideas about the traditional processes of design, informed by the influential approach of Morris and the Arts and Crafts movement, have been referred to as a process of applied skill and understanding, in which the maker seeks to use, understand and 'honour' the structure and parameters of the material. The process of applied understanding is therefore a way in which the details of the material are generated and understood; an ongoing dialogue whereby the will of the artist or designer is expressed through the parameters and properties of the material. This idea can be visualized as a simple 'feedback loop' between the object and material, an iterative process in which one side informs the other, illustrated as follows:

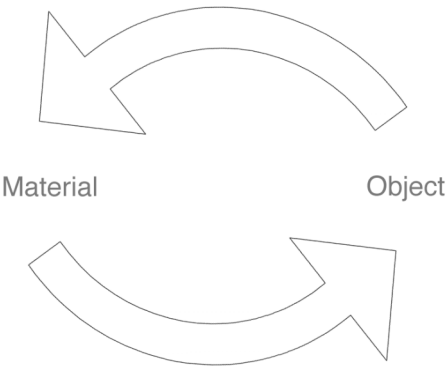


Figure 5.2 A workflow diagram illustrating the feedback loop of the craft-centred design process

The idea of 'dialogue' is something which also has resonance with the creative practice and use of code. Computational design work operates in tension between two separate elements; the work is both abstract and visual, conceptual and aesthetic. It must function correctly as code ('obey the laws' of the syntax and logic) as well as being successful as an aesthetic piece of work. It is possible, therefore, to align the dialogue of craft, between material and maker, to a computational dialogue between visual and technical parameters. However, the process of creating both pieces of project work, demonstrates that the dialogue of computation is more complex than a simple push and pull between material and aesthetics. The model of the traditional design cycle is one which cannot be directly transposed onto the process and workflow of the computational design process.

The non-physical, abstract, nature of computation means that the structure of code has to be *created* as part of the design process. Unlike traditional material, computational material is not in a pre-determined state ready to be manipulated; before being constructed into a specific structure, the code exists only as generic 'rules' of number, logic, data and concept (2.2.2). These rules form the 'layers' of work which are developed and manipulated throughout the design cycle of *both* the Colorcalm and Moving Wallpaper work. The computational workflow therefore defines, constructs and manipulates the material, and the data structure, as the project moves from abstract concept through to the final visuals. This process is reflected by a computational cycle in which the work moves from concept to the data structure and data detail, a process during which the structure is given form. This computational workflow may be illustrated as follows

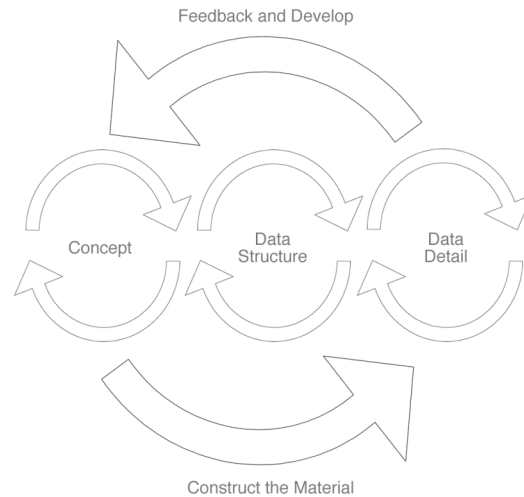


Figure 5.3 A diagram illustrating the computational design workflow

The specific phases of the project workflow can be summarized as follows:

Concept: In which the overarching concept is developed and where the core concepts and ideas for the project are established.

Data Structure: In which the visual structure and behaviour of the work is developed. Moving the concept into a more defined set of classes, functions and behaviours. The broad architecture of work.

Data Detail: Adding and manipulating the data detail of the work: specific variables and algorithms.

Each project undergoes a similar series of iterations from which the material of code is developed. Moving from initial concept, development of data structure and manipulation of data detail, the process is one in which the material of code is conceptualized and constructed. The overall process gradually moves from a 'broad' cycle of engagement during the early phases of work down to a narrower cycle of work during the latter stages of the project. The cycle of the process operates and evolves as a continual

alternation ('flow') between conceptual development and manipulation of the data detail. Each cycle of work generates the detail and parameters of the project.

As the process moves through its phases, the addition of computational detail (e.g. functions, classes, data, variables, methods etc.) add form and structure to the previously unformed computational environment. The loose conceptual data structures become more detailed and well defined, and movement within the workflow becomes increasingly difficult. As the abstract concepts gradually gain more solidity, the material moves from being a completely 'fluid', conceptual amalgamation of computational conceptual possibilities to a very specific structured set of classes, functions and variables. Opportunities for making significant changes to the nature and character of the work decrease, and a greater amount of focus is placed on changing or adding nuance and subtle variation to the detail of the work. The process can therefore be summarized as a journey from fluidity to rigidity. The computational material begins formless as a concept, into a more defined set of classes functions and algorithms; the work moves from a loose fluid concept to well defined rigid detailed structure.

The generic computational cycle outlined by both of the projects, therefore represents an ongoing process of development during which the concept, data structure and data details are developed, processed and constructed. The computational work begins as a formless, open-ended set of concepts and possibilities which, during the course of the work, move towards an increasingly solid computational structure defining a specific range of digital artefacts and visuals. Unlike software-driven or traditional craft-like work, the final solution is not fixed to a single design or visual artefact but is a piece of work which embodies a range of possibilities.

The cyclical, iterative workflow, and the nature of the computational language emphasizes the importance of structured learning, understanding

and development as part of the computational process. The required skill-set of programming, centred around a non-intuitive understanding of the rules and syntax of code, means that a developed understanding of the syntax and structure of code becomes part of the computational workflow. Iteration through the workflow of the computational process builds up knowledge and understanding of each individual data structure. The development and construction of the computational material, moving from an unformed, open-ended, range of possibilities to a defined data structure, therefore characterizes the process as a conceptual workflow of understanding; a process of engagement with the material during which concepts, material and visuals emerge.

Whilst both computational and software-centred design share similar abstract digital environments in which work is created by the movement or manipulation of digital data, the approach reflected by the workflow of the computational design process highlights key differences. The ordered logical, iterative process of understanding and development contrasts with the import / export network model of software-centred design. Whereas software design is about moving data between different software environments, computational design, as its workflow suggests, is centred around creating and manipulating data elements through an ongoing dialogue of structural development and manipulation. The structured iterative workflow of computation, therefore, represents a means of thinking, understanding and manipulating the 'material' of code which resonates with that of the iterative cycle of the traditional craft practice of 'making'.

The workflow of computational design can therefore be seen as an interesting intersection between digital and traditional processes; a workflow which takes the ethos of traditional design and applies an ethos of making to the digital data-based environment of the digital universe.

5.3 Computational Process: Detail

Having considered a generic computational workflow, through which both projects move, it is now necessary to compare the detailed structure and usage of code for each piece of work. Each phase of the workflow (concept, data structure and data detail) offers differing abilities to develop, change and manipulate the material of the project using different amounts of 'flexibility'. The differences between the Colorcalm and Moving Wallpaper project are highlighted by their approach to, and use of, the detailed phases ('layers') of the computational process. Each of these phases will now be discussed in turn with specific reference to each project.

5.3.1 Concept: Constructing the Material

The work begins as a computational concept; before any work can be created, or manipulated, key concepts which establish the foundational elements of the work have to be put into place. Drawing upon a range of abstract computational ideas (recursion, iteration, randomness) initial decisions are made which lay the foundation for the nature of the computational material. This initial abstraction of the work defines the parameters and direction of the work. The conceptual 'layer' of the code is therefore the phase in which initial decisions are made regarding how the ideas for the work are to be abstracted into a computational structure. For both pieces of project work, the key conceptual decision regards the 'computational' definition of the line. There are significant differences between the conceptual set-up of each project.

Colorcalm

The Colorcalm project is based upon a simple literal, visual definition and understanding of a line. The project attempts to translate a traditional concept of drawing marks on paper into a screen-based computational environment. This literal approach is reflected in the initial set up of the work, in which each object leaves behind a trail, rather like plotting the arc of a pen tracing a path across paper.

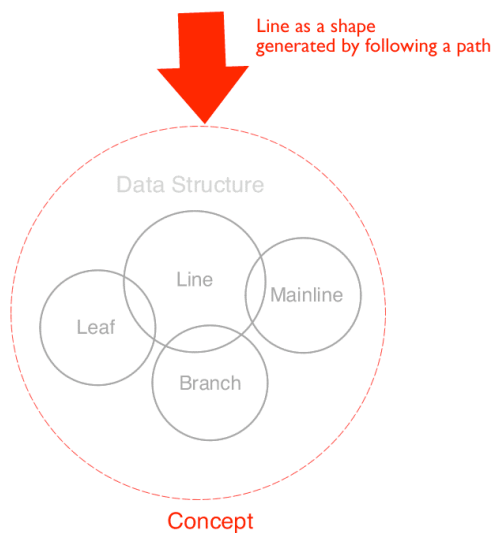


Figure 5.4 An illustration of the concept element of the Colorcalm project

This simple, core concept informs the development of the rest of the project. The fundamental idea of the line defined as the trail of an object moving across the screen informs all elements of the work, and is repeated for all visual elements of the drawing; stems, leaves, branches and flowers. The simple, literal nature of the concept of the line, abstracted as a single class, allows little room for development or expansion. Subsequent development of functions and classes continue to lay emphasis on the initial idea, expanding the individual visual qualities of the lines without developing the concept.

Moving Wallpaper

In contrast to this, the Moving Wallpaper project takes a broader, more abstract, 'computational' approach to the idea of the line. Ideas which define each line in the Moving Wallpaper project are based on a broader understanding and application of computational concepts. The line is not thought of as the movement, the path of a single object, but as a series of separate, yet interconnected, data elements. In this way the single line becomes defined as a trail of individual data elements which, whilst being connected as a single 'line', also maintain their own functionality and identity, being subject to individual manipulation from internal and external sources. Increasing emphasis a line as a data-based, computational, object is reflected by a greater emphasis placed on the behavioural, rather than the visual, qualities of the line. The diagram, below, illustrates the key concepts which constitute the core concept of the Moving Wallpaper line:

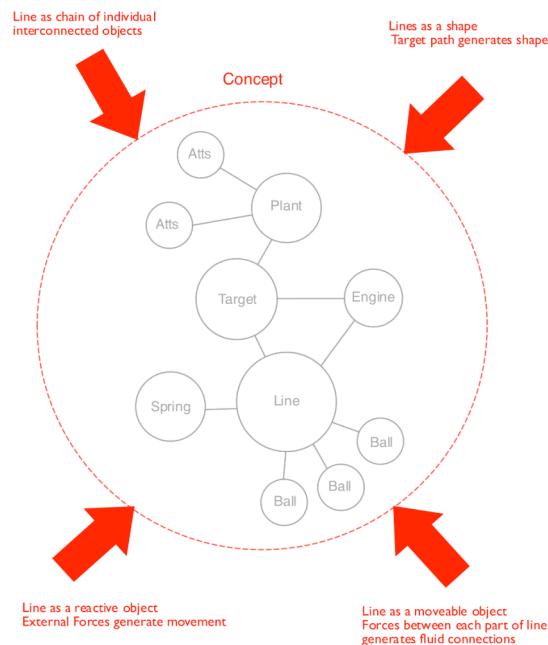


Figure 5.5 An illustration of concepts informing the Moving Wallpaper project

Emphasizing the line as a computational-based, behavioural object, rather than as a screen-based version of the paper-based line, (i.e. as the movement of a single pen on paper) gives the line a wider conceptual base to work from, and encourages a more fluid development of the code. The concept of the line is not fixed by a narrow definition based around the traditional line, but is defined as a computational entity which has freedom to grow and develop. The line, as a collection of combined forces impacting upon a series of individual data elements, affords greater room for fluid development and allows shapes to be more easily changed and moved. The broader concept of the Moving Wallpaper project therefore emphasizes the computational 'fluidity' of the line.

5.3.2 Data Structure: Broad Architecture of the Work

The second phase, or 'layer' in the workflow deals with the data structure of the work. The data structure defines the 'architecture' of the project, establishing the core classes, methods and functions through which the detail of the individual variable data will flow. The data structure flows from the concept, and is the means by which the initial idea is abstracted into a computational structure. The data structure 'layer' therefore provides the structure within which the individual data elements (variables, etc.) can 'flow', and therefore forms an important link between the concept and the detail of the individual data elements. The initial concept of each project is therefore developed into an appropriate data structure which handles, defines and controls each of the visual parameters and functions of the work. Both projects develop their data structure from the foundations laid by the conceptual framework by applying an Object Orientated Programming (OOP) approach to the structuring and handling of the data elements of the code. Differences in concept are reflected by the creation of the different data structures which define the parameters of the project. The following is a comparative discussion of each data structure.

Colorcalm

The Colorcalm data structure is borne from the literal, narrow definition of the line outlined by the concept phase, and its use of data is a reflection of this. The following diagram gives an overview of the structure of the Colorcalm project. Each of the circles (below) represents a core class within the Colorcalm project.

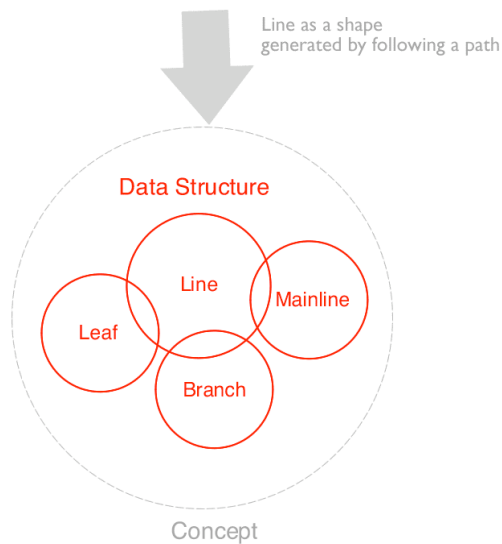


Figure 5.6 An illustration of the data structure of the Colorcalm project, and the relationship between classes.

The Line class defines the initial line, and remains the same throughout the lifespan of the project, forming the foundation for the rest of the work. This class is developed by means of an 'inheritance' structure in which the core elements of the line are repeated with some minor changes to define different types of lines. Each new class is based upon the generic Line class which is used to define each new specific line type i.e. 'Branch', 'Leaf' and 'Stem'. Names of the classes reflect the literal allocation of the OOP structure into individual types of lines. Each new class is therefore an amended duplication of the 'core' line class. As the diagram illustrates, the

data structure is narrowly defined and developed, limited to the core concept of the single line class. The data structure does not develop the concept, rather it adds layers of similarity within an already limited and narrowly defined structure, allowing only limited movement and flexibility between each of the classes.

Moving Wallpaper

In contrast to the narrow and rigid structure of the Colorcalm project, the data structure of the Moving Wallpaper project provides a broader and more conceptual data structure which reflects the broader nature of the concept. The core concept of the line is represented by a number of separate classes, and data entities, defining the line as a collection of interconnected data elements.

Having initially divided the line object into a couple of different classes (`Ball` and `Line`) the Moving Wallpaper data structure continues to develop a broad, abstract class structure for development and growth of the lines. The addition of new classes introduce new visual and behavioural attributes to the visuals. The nature and functionality of the line therefore continues to grow and develop, reflecting the computational concept of the line which is not fixed to a narrow single idea and definition. The abstract interpretation of the line is reflected by its data structure: rather than creating a single class to define each line, the concept of the line is defined by a number of different classes.

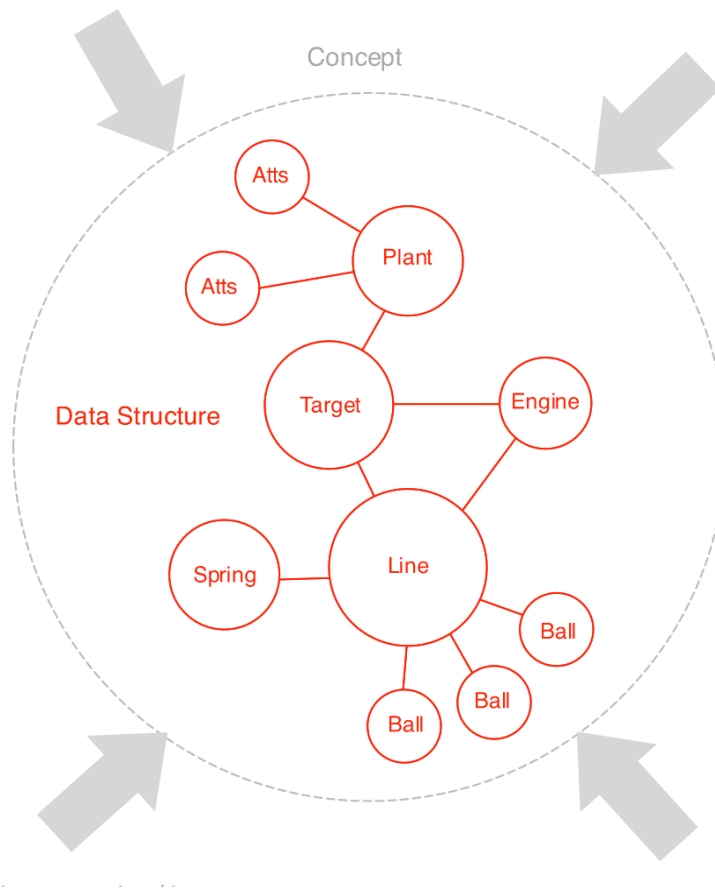


Figure 5.7 An illustration of the data structure of the Moving Wallpaper project, and the network of inter-related classes.

As the diagram illustrates, the Moving Wallpaper data structure places its emphasis upon the use and application of inter-related objects and classes, which allows for greater flexibility in creating and manipulating lines.

Unlike the previous Colorcalm diagram, (fig. 5.6) in which elements of data structure are rigidly connected (overlapping), the diagram (above) illustrates how the relationship between core classes and objects in the Moving Wallpaper work is based around a more fluid, modular structure in which bonds between conceptual element of the work can be more easily added or modified. The way in which the Moving Wallpaper work is developed reinforces the concept of the line as an abstract computational object, emphasizing the unique computational qualities and characteristics of the screen medium (2.2.4). Functions, classes and behaviours are added

throughout the process to explore and change the movement, responsiveness and behavioural qualities of the line, generating 'living', fluid, organic lines. Emphasis on data as the core concept and foundation of the work gives the project a fluid, organic structure which makes a more considered use of the material of code. The use, understanding and application of data, 'classes', in this way creates a piece of work which demonstrates computation integrity, work which makes greater use of the natural use of the core concepts and structural elements of code.

5.3.3 Data Detail

If the data *structure* provides the architecture through which data 'flows' throughout the work, then the data *detail* concerns the allocation, use and manipulation of the individual data values (number, characters, strings, etc.) used to define the individual attributes of the work, its look, movement and behaviour.

A core part of each project is the use of numeric data. Both the Colorcalm and Moving Wallpaper projects allocate numeric values to each visual attribute of the work. Within each project, the use of numbers constitutes the main way in which details are allocated, manipulated and controlled. Simple mathematical algorithms are used to assign and change the data values which directly affect the visual details of the work. The flow and movement of the number data changes, however, for each piece of project work. The following is a discussion comparing the flow of individual data detail for each project.

Colorcalm

The usage and allocation of data detail in the Colorcalm project operates within the narrowly defined data structure, and follows the simple structure of the data elements. The architecture of the structure follows a linear path in which data values are passed from line (parent) to mainline, to branch to leaf in a direct process. The flow of data elements can be summarized as follows:

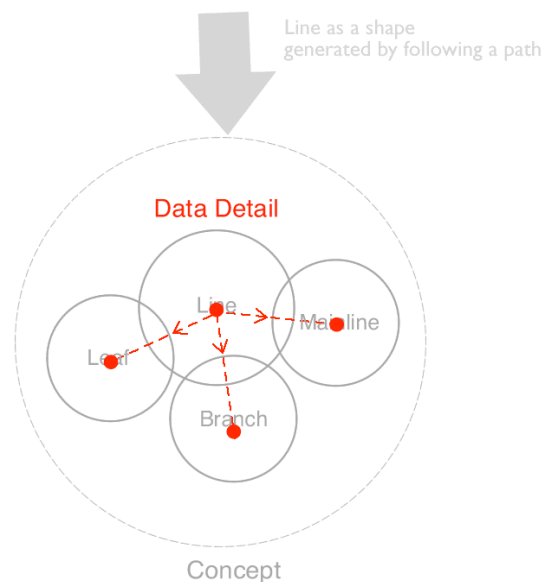


Figure 5.8 An illustration of the 'flow' of data between classes in the Colorcalm project

As the project grows, so increasing numbers of individual numeric variables are assigned to the line class to build up the detail of the work, each assigned to a specific aspect of branch line, leaf line or main line. The result is a huge amount of similar variables, each controlling a specific element or attribute of one type of line (3.6). Variation within the data structure is limited to the use of individual variables and conditional statements used to control the 'logical' structure of the work.

The conceptual 'foundation' of the Colorcalm project is literal and narrow, and as the single idea is extended further and further, more is added on top of the narrow conceptual, computational base, until the entire structure begins to 'creak and wobble' making it difficult to manage and change. Lots of individual variables build up the detail of the work, each assigned to one specific aspect of the visuals, which creates an enormous amount of individual pieces of data. Reliance upon conditional statements to create variety of growth, speed of movement, colour, etc. results in a rigid, 'mechanistic' system which offers limited scope for control or variation. Increasing numbers of conditional statements applied to growth, structure, speed of movement and colour, etc. add to the complexity and rigidity of the data structure. The result is that the structure quickly becomes rigid and difficult to manipulate.

Moving Wallpaper

Use and allocation of data detail in the Moving Wallpaper project reflects the broader modular structure of the architecture, which allows data to be handled in a more flexible way. The following diagram outlines the movement of the data within the architecture of the work:

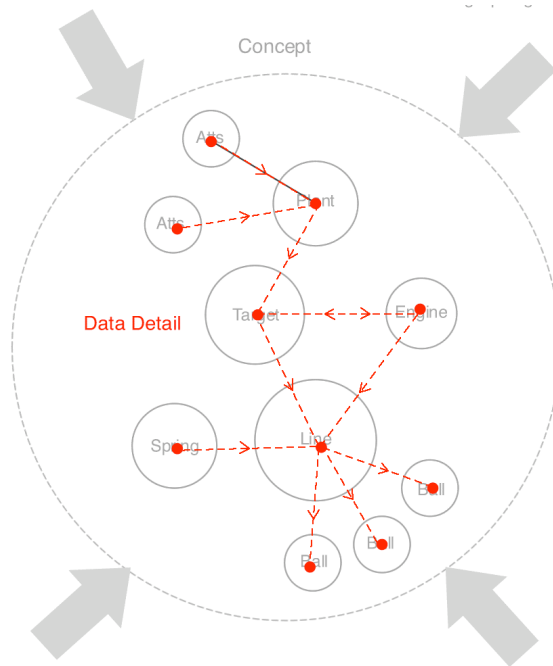


Figure 5.9 An illustration of the 'flow' of data between classes in the Moving Wallpaper project

Data creation, movement and handling is central to the thinking and structure of the Moving Wallpaper project, creating the ability to easily add to, allocate, change and manipulate data elements is a central idea which drives the work, and informs the visuals. This is reflected by the allocation, use and flow of data thorough the project. A good example of this is the use of 'string' data as a means of summarizing structural conditions: substituting a complex and unwieldy set of 'if statements' with a series of characters which make decisions regarding structure and growth affords a greater degree of flexibility and nuance, enables easier and more direct manipulation of the structure to occur (4.4). This more considered, data-centred, approach to creating the work, affords a greater amount of change, and nuance in the graphic element of the work, which is reflected by the data architecture and its usage and manipulation.

5.3.4 Summary

Having considered the use of concept, data structure and data detail in each of the projects, it is now necessary to summarize the findings and to outline the differences between the two projects. The difference in emphasis is summarized in the following table:

Code 'Layer'	Colorcalm Project	Moving Wallpaper Project
Concept	literal simple narrow	broad conceptual computational
Data Structure	simple linear	modular abstract
Data Detail	mechanical repetitive rigid	fluid flexible computational

Table 5:1 Summary of difference in code structure between Moving Wallpaper and Colorcalm projects

The Moving Wallpaper project is focused on an understanding of computation as a data based material; a project in which emphasis is initially placed on the use of conceptual rather than visual elements of the work and which demonstrates a broader use, allocation and understanding of data. Rather than focusing on a narrow concept and use of data as individual number values to be allocated to specific elements of the line (e.g. colour width and shape), the Moving Wallpaper project uses collections of data (e.g. classes and strings) to inform and develop the work. Data detail is not limited to a list of variables, but given a broader part to play in the development of the project. Similarly, the modular nature of the class system reflects a computational concept which recognizes the importance of flexible data structure to inform the final visual artefact of the project. The Moving

Wallpaper project explores the *concept* of computational line; right from the beginning of the project, the ideas and behavioural qualities of computational lines are being explored and developed.

In contrast to this, single numeric values and their use and evaluation, form a much greater foundation to the Colorcalm project, and are used not only for each detail of each line, but also extensively to control the overall structure of each plant, leaf, etc. The Colorcalm project has a narrower computational, and conceptual basis to work from. It places greater initial emphasis on visual development of the work and relies more heavily on the use and manipulation of individual numbers and single variable values. These values are not integrated into a wider data structure, as they are with the Moving Wallpaper project. In summary the process and use of code in the Colorcalm project can be defined as being:

literal: a simple, literal translation of the idea of drawing lines in screen environment.

visual: definition of structure (line class) and functions to detail and concentrate on the aesthetic details and development of the line.

simple: a narrow concept defined by an emphasis of drawing, organic *looking* lines.

numeric: (mechanical / rigid): a structure which is heavily based around the modification of numeric data and simple calculation / evaluations.

Comparatively the Moving Wallpaper's use of code can be summarized as being:

conceptual: based on the development of data structures.

computational: (data based): seeing the problem / issue as being a data based one and recognizing that the visual objects can be defined by a convergence of different data elements.

organic / fluid: work shows a greater emphasis on the definition and *flow / movement* of data / objects, use of natural phenomenal : forces, growth, etc.

5.4 Visuals

Having considered, summarized and compared how each of the projects (Moving Wallpaper and Colorcalm) handles the different cycles of the process and aspects of the code, it is now necessary to compare the outputs of the coded work, the visual aspects of each project, the 'artefacts' of code. Computational artefacts are the 'flip side' of the code; the computation structure provides the environment for creating and making the final visual artefact experienced by the end user. It is impossible to consider the visuals without reference to the code, the material, which created it. Discussion of the computational artefacts will therefore be done with reference to the code structures which created them.

Both projects share common visual intentions and aims i.e. to align the visuals, shapes and forms created from code with those of traditional decorative craft. Both pieces of work are particularly inspired by the decorative shapes and forms of William Morris' wallpaper. Each piece of work therefore displays some common visual themes. The slow spread and growth of the botanic shapes across the screen, gives the visuals of both

pieces a decorative feel and flavour. The graphics are intentionally simple and slow, 'quiet' in nature, to allow and encourage a contemplative approach to the work. A visual style which has more to do with traditional design and traditional form than with the high-speed, visually excessive nature of screen based graphics. Discussion of the computationally based objects (2.2.4) outlines key characteristics of the computational artefact i.e. that they are visual, time based and behavioural ('reactive'). Reference to these attributes will inform this discussion.

5.4.1 Colorcalm Visuals

The work of the Colorcalm project reflects the simplicity of the computational structure from which it is created. A direct line of comparison can be seen from the initial 'sketches' to the final phase of work. The original idea of the line as a 'pen' tracing a path across the screen, is reflected in the nature and type of visuals that are produced, as plants move and grow across the screen in a simple, yet elegant, manner, and the structures become more complex a clear association is maintained between the early and the later pieces of work.

Throughout the work, the visuals maintain simple, organic qualities, reflecting the structure of the organic plant life by which they are inspired. Variation of lines and plants is, however, relatively limited, and the nature and quality of the strokes becomes repetitive. Although the work possesses some visual elements of organic shape and form, the Colorcalm pieces are visually flat and unresponsive. As the work evolves over time, visual variation is limited to colour, branch width, shape and number of leaves. Each variation of the 'wallpaper pattern' design contains the same, or similar, branch leaf structures. Visual experimentation with shape, curve and colour is undertaken (3.5) but at its heart, the work remains fundamentally the same

as the early line sketches. This is illustrated by the following group of screenshots which demonstrate the fundamental visual similarities between the earliest experiments and the latest pieces of work.



Figure 5.10 A visual comparison of stages of development from the Colorcalm project

As has been outlined (2.2.4), a key aspect of the computational object environment is its dynamic and visual fluidity, generated by the computational code. The ability to create and use interactive, dynamic systems enables computational work to develop, change, react or interact. The final artefacts of the Colorcalm work, however, do not demonstrate this element of computational 'organicness' inherent to the environment. The visuals have some decorative elegance but lack a fluid or behavioural element on screen, and appear as if they have been animated as part of the traditional linear process. The work does have elements of generative randomness, a system which creates different shapes and forms each time it is run, which does express some elements and characteristics of computational dynamism. However, there is a mechanistic (static and rigid) feel to the visuals, which lacks substantial organic or behavioural qualities. The Colorcalm computational artefacts demonstrate little visual variation or fluidity elements which form core characteristics of the computational environment.

5.4.2 Moving Wallpaper

Just as the visuals of the Colorcalm work are a reflection upon the concept and data of the code, so similar comparisons can be drawn between the code and the visuals of the Moving Wallpaper project. As we have seen, the computational structure of the Moving Wallpaper project is more broadly based than that of the Colorcalm work, using a wider, more flexible, data structure to draw and generate lines. The nature and behaviour of the resulting visuals reflect the change in approach and concept towards a more computationally centred piece of work. The fluidity of the data system allows easier, more flexible access to generating a wide variety of shapes and forms. The Moving Wallpaper project can be used to generate a greater range of 'plant life' and allows a range in shape colour and size of flowers, stems leaves and branches to be generated within the same application. (See figure 4.27 in chapter 4.6.2).

The ability to control and randomize small details with the plants (e.g. leaf curls and shapes, flower blooms, etc.) gives each plant an organic-random feel allowing each plant and each leaf to maintain individual qualities of form and shape within the broader parameters of the plant structure. The visual value of adding subtle changes to each leaf or petal allows the work to maintain a look of individuality and has echoes of William Morris prints which combine repetitive pattern design with a look of organic randomness and 'naturalness'. The combination of 'randomness' entwined within a wider organized structure is one which resonates the visuals of the wallpaper design with ideas of generating graphics within a computational structure.

The following images highlight the way in which the Moving Wallpaper pattern (*left*) echoes the individual 'organic randomness' of the leaf shape, and visual rhythm of the William Morris pattern (*right*).



Figure 5.11 A comparison between Moving Wallpaper visual and William Morris's Willow pattern image from: http://www.williammorrystile.com/small_images/black_willow_sm.jpg

In addition to the range of visual vocabulary afforded by the Moving Wallpaper project, the work also has the capacity to use and introduce visual and behavioural fluidity. Unlike the 'static' drawn shapes of the Colorcalm project, the leaves and plant forms of the Moving Wallpaper project are reactive; once on the screen they can be manipulated ('blown' around) by mouse movement. The leaves, flowers and branches express the visual and behavioural fluidity of the computational structure. The Moving Wallpaper data structure, therefore, generates a wide variety of shapes and forms, which express visual and behavioural qualities, reflecting some key attributes of the screen-based, computational environment.

In addition to the differences in 'fluidity' and of the 'reactive' qualities of the projects, there are also differences between the final aesthetic 'texture' of each of the projects. The graphic element of the Moving Wallpaper visuals have perhaps lost their simplicity. They tend to have a greater computational aesthetic, being more graphically computational with bolder and larger forms; an aesthetic which includes some use of the three dimensions of the screen, making more considered use of the nature and characteristics of the screen environment. In comparison, the texture of the Colorcalm work maintains more of a simpler hand-drawn aesthetic. An overall summary of the differences between the two project is outlined as follows:

	Colorcalm	Moving Wallpaper
Summary of Code Concept and Structure.	'Line as a pen' simple, direct concept and structure	'Line as reactive, behavioural object'. computational.
Visuals	simple organic	graphic. using elements of three dimensions.
Time-based Evolution	variety limited to colour and shape	greater sense of growth and decay. broader variety of shape, form and structure.
Behavioural / 'Reactive' Qualities	no behavioural qualities	reactive to external and internal forces.

Table 5:2 Overall summary of visual and structural differences between Moving Wallpaper and Colorcalm projects

The process of increased understanding, highlighted by the development of the work from the Colorcalm project to the Moving Wallpaper work is one which reflects personal development experienced by the researcher. Beginning the project as a designer with limited experience in the use of computation, the researcher started with a clear, but limited concept regarding the use of code to create work. The initial pieces of work centred around an understanding of computation as a means to manipulate numeric values associated with visual elements on screen. Assumptions about the direct and simple association between number and visual were made: it was thought that if the researcher could 'get the numbers right' then the visuals would follow. Data structure was assumed to be something which could be built around the core individual numeric manipulations. The limitations of this simple idea are witnessed by the limitations of the Colorcalm work and its unwieldy, number-heavy structure.

Acknowledgement of the limitations of numeric manipulation, together with the need to develop work to reflect more of the characteristics of the computational environment, led to a much wider use and understanding of the computational material. A broad appreciation of data and the recognition of the intrinsic importance of data structure, helped to develop a view of computation which encompassed a wider range of data types. The understanding of computational material as a data *structure*, which allows the movement and flow of a variety of data types, is one which began to develop during the Moving Wallpaper project and is reflected by the nature of the work. Seeing the computational material as a structure, rather than a series of numeric values and algorithms, helped to reveal the wider potential of the material as a conceptual behavioural environment.

Working with computational material has been a difficult but rewarding process. Inspired by the ideal of sculpting elements of code, the development of each stage of the project provided a challenge to the researcher's understanding and application of the rigid rules of programming structure and syntax. Despite the difficulties, the experience of coding, and of learning from mistakes, has offered a valuable insight into the intricacies of the code environment as a 'material'. The process of working on both the Colorcalm and Moving Wallpaper projects can be seen as a kind of 'apprenticeship' in which active involvement with the syntax, logic and structure of code generates a clearer vision of the potential of the environment and a clearer view of the characteristics of the material. This 'apprenticeship' has revealed the significant 'layers' of the computational material, characterized by the creation and manipulation of numeric detail and data structure.

A core characteristic of the computational material is its close association with numeric values. Within each project, number values are associated with individual visual elements which determine elements of motion, colour, line

shape, and line width, etc. Connections between number and visual are interwoven deep into the heart of the material, a key part of the creative and technical process within both pieces of work. Understanding the relationship between individual number values, number pattern and visuals therefore became an important part of understanding the material structure and character of the code. Creating and manipulating number patterns through the use of mathematical functions and calculations (e.g. sine waves, randomness, incrementation, etc.) generated visual change and variance within the work. Within both the Colorcalm and Moving Wallpaper projects the process of 'play' and experimentation with variables, used to test and experiment the inter-relationship of number values, is the means by which core aesthetic details (line weight, shape and form) are developed. Both practical projects have shown how, as numeric detail is added, numeric patterns and values emerge which change the inter-relationships between individual number values. Subtle variance and change is achieved by manipulation of individual numbers and algorithms. Changing the rate or type of calculation adds nuance and change to individual elements of the visuals. Organic curves and loops were, for example, initially created in the Colorcalm work by using values generated by a simple 'sine' wave, in which numbers used to determine the frequency of the wave affected the nature and 'quality' of the line. As the work developed, the algorithm progressed and changed to include a greater amount of numeric values around the sine wave calculation thus changing its attributes and allowing a wider variety of flowing curves and loop shapes to develop. Similar algorithms were used to change other visual qualities of the line e.g. colour and angle. Understanding the flow and change of numeric values and patterns as they shift is more than a mathematical or 'technical' exercise and becomes a key part of exploring the material and experimenting with aesthetic subtly and nuance. The ability to create and change numeric data is therefore a core characteristic of the computational material, affecting the type and nature of change as the visuals move and grow across the screen.

Beyond the detail of individual numeric values lies the broader data structure of the computational material: a further 'level' of the material, revealed through practice, which defines how numeric data detail is assigned and flows through the structure of the work. The development of data structure via the creation of classes and functions constitutes a core element of the behavioural characteristic of the material. Every function and class within both the Colorcalm and Moving Wallpaper projects defines behavioural attributes of the material structure of the work, adds conceptual layers and extends visual and behavioural characteristics. Each class can be seen as a conceptual 'building block', defining core characteristics of the material, which is reflected through the visual and behavioural attributes of the work. Throughout the project lifecycle, the addition of classes, and the creation of relationships between them enabled the material to evolve: extending the detail and the scope of the work, adding new conceptual and behavioural layers, and creating opportunities to develop the computational material.

The conceptual data structure of the Moving Wallpaper project created a flexible material which afforded greater manipulation of the core conceptual elements of the work, allowing greater opportunities to grow and develop a wider range of visual and behavioural possibilities. The separation of elements in the Moving Wallpaper work into distinct classes associated with specific attributes of the work (e.g. line shape, branching structure, etc.) allowed a loose 'flow' of associations between each element with the potential to be independently changed and altered. On the other hand, manipulation of the more tightly defined structure of the Colorcalm project was much more difficult, its more rigid data structure based on a series of similar but related sub-classes, offered limited opportunities for development. Therefore the construction, development and manipulation of the classes and functions can be seen to define the core characteristics of the material; each specific element of the data structure defines the specific affordances of the material structure for each project. Nuance and subtlety within the

work is developed by altering the scope and inter-relationship between classes and other data elements, which together define the material structure of the work, its capability and flexibility. Creating 'bonds' between concepts by integrating classes together, or extending the specific scope of an individual class, allows the conceptual, behavioural and visual range of the material to be extended and changed. Although both projects share common material qualities in terms of basic numeric manipulation, each project has developed its own material characteristics based on the computational layers created through the structure and inter-relationship of core classes and functions.

6 Conclusions

The research question centres around the identification of key features and characteristics of the practice of computation, when applied to the context of creative design. In seeking to address the overall aim, the research has outlined three core objectives which relate to the consideration of the context, the process and the material of creative computation. These are expressed as the following questions (1.1.2):

- How does creative computation fit within a broader traditional and software-centred, digital design landscape?
- What are the key features of the process as a means of making: how can the relationship between the maker and the computational material be characterised?
- What are the key features of the material: how can the relationship between the material and visuals be characterized?

The conclusions will, therefore, summarize findings in relation to these questions and, as the project aims to be useful to other designers interested in using code as part of their practice, will reflect upon elements of the researcher's own experience. The research has addressed the three core questions relating to the context, process and material of creative computational design.

6.1 Characteristics of Context: Computation in the Design Landscape

The research highlighted three key design areas for initial comparison and discussion: the Arts and Craft movement; digital (software-centred) design; and computational design. Areas of difference and similarity between these were investigated and the themes of 'ethos', 'material' 'process' and 'object' were identified as the research headings for discussion and comparison. These themes, used to structure the contextual review, enabled the discussion to move beyond technology or media-led analysis towards a concentration on the process. Using the themes as structural headings, areas of commonality between computational work and the other areas of design (i.e. Arts and Crafts movement and software-centred design) were identified. The findings are summarized in the table in 2.4.

The non-physical nature of the digital object emphasized areas of intersection between computational and software-centred design; both being environments in which object of work is part of the abstract digital environment. However, investigation into the ethos of computational design revealed key areas of overlap with attitudes and ideas of the Arts and Crafts movement. Despite differences in the 'physicality' of material, the requirement to use, understand and 'honour' the material of design is identified in both the Arts and Crafts movement and computational design, revealing interesting areas of overlap regarding the significance of material and process. A common approach to the material, and process of design, links attitudes of computational with traditional craft-centred values and these links inform the basis and direction of the research. The summary in 2.4 details these areas of commonality, highlighting, in particular, the invisible forces and attitudes which underpin traditional and computational approaches i.e. how the use of code within a context of 'making' may be seen to reflect the ethos and attitudes of traditional craft in creating

contemporary digital objects. Taking a broader understanding of 'material' as central to the understanding of craft, makes it possible to draw comparisons between the attitude and ethos of traditional and computational design. Even though the process of making in the computational environment is abstract and conceptual rather than hands-on and tactile, the *mental* engagement of computational designers, their keenness to use, understand, reflect and 'honour' their computational material, reflects the ideals of traditional design expressed through the ethos of the Arts and Crafts movement.

The research has shown how the process of making using code is a valuable part of computational design, a means by which the material is understood, reflecting the concerns, values and *attitude* of craft. The ideas and ethos of computational designers whose engagement with, and workmanship of, the programming environment supports the idea of code as a type of 'digital material'. The final pieces of computational work manifest the skilled understanding of the designer, being the result of both the tactile (visible) and 'mental' (invisible) forces acting upon the unformed material whilst enhancing the inherent qualities and characteristics of the material itself. Usage of code as a means of creating unique pieces of work emphasizes the distinctness of the computational environment; a freedom from the limitations of pre-written software and an experimental approach to creating 'handmade' digital forms (2.2.1).

Computational design, therefore, represents an interesting combination of attitudes taken from craft, and technology. The use of code reflects a desire to *understand* and to use the computational medium, providing the ability to create digital objects by sculpting (crafting) the environment of the computer. Despite the lack of physical, or tactile engagement with the material, the computational approach places value on the 'hand-made', individually unique objects, which express an understanding of the material of code and place an emphasis on the process of playful experimentation. As such,

computational design has been seen to mirror some of the core approaches and the ethos of the Arts and Crafts movement, as well as the 'spirit of craft' (Nakamura & Fitzpatrick, 2008). It is now possible to consider how this ethos is translated into the process and material of computation.

6.2 Characteristics of Process

Having considered the context of computation, a review of the themes of the computational *process* will be summarized in relation to the Arts and Crafts movement and software-centred design. The research has shown that each design process is distinctive, dictated by the individual nature of the material, and Chapter 2 of the research sketches out some key features and approaches relevant for each area. The research highlights how the *approach* and *importance* of process of traditional (craft) and computational work have some striking similarities, and although physically different, the attitude and approach of the computational design process can be seen to align it with traditional craft.

6.2.1 A Creative Process of Making: Similar to Traditional Craft

The use of code in a creative design context can be defined as a process of 'engagement' with the material of computation, a means of creatively engaging with the 'stuff' of the computer environment (2.2.3). This is reflected by the language used to describe the process of work: code is not simply written, but is "manipulated" (Maeda, 1999, p.69) or "resculpted" (Gerhardt & Jarman, 2007, p.392). This re-iterates the idea of the process as a key part of the creative act, an extension of the idea of code being the computational *material*. The process of creating, using code, is therefore an essential and valued part of the creative design process; a fluid, iterative dialogue of experimentation, exploration and play, through which limitations

and possibilities of the material are explored, understanding of the material developed, and creative accidents and unexpected results are embraced, (2.2.3). Just as the structure, the solidity and the flexibility of the traditional craft material is tested and manipulated by physical intervention, so the computational material is also pushed and pulled, tested and explored, through the process of development.

Although not a physical process of manipulating or crafting a physical material, the language-based act of programming embodies a 'closeness' between the computational material and the maker, which encourages close understanding of individual objects and an expression of ideas. Understanding of the computational material informs the design decisions and concepts of the work, and generates a means of developing a relationship with the material of code. The process of play and experimentation, tests and establishes the boundaries of the work, a process through which a closer understanding of the material is generated.

The process of working with computation is, therefore, a key part of the creative act and can be seen to be equally important as the final artefact itself. The importance of process is reflected in comments by computation designer Ed Burton:

I tend to be more interested in process than product. Rather than create singular static forms I want to create processes that may result in countless potential forms emerging overtime.

(Burton, 2007, p.263)

Manipulation of code through experimentation, therefore, plays an important role in generating a greater technical understanding of the material, broadening and developing the scope of the work and informing creative decisions and direction of the work. The emphasis on skilled manipulation of

code, and understanding of the material, resonates with the ideals of traditional practice and the Arts and Crafts movement.

The significance of the process as a means of play, experimentation and understanding has been reinforced by the researcher's own personal experiences whilst developing the Colorcalm and Moving Wallpaper projects. From the project based work, I have learnt that the process, and practice of computation is a highly significant discipline. This has informed my own understanding of the 'material' both technically and creatively, and helped to develop a broader understanding of the nature and characteristics of the material.

The technical aspect of developing an understanding of the intricacies of code proved to be both frustrating and rewarding in equal measure. The very definite and well defined rigidity pertaining to the rules of computational syntax and structure provided a constant technical challenge which the necessity of getting work done made unavoidable. However, the continual development of knowledge offered a great deal of freedom of thought and allowed the development of ideas afforded by the understanding of computation.

The process of working with code not only allowed the development of increased *technical* understanding but also provided an important way of gaining a *deeper* understanding of the nature of the process and of the material itself. The workflow of practice and experimentation provided an environment in which to explore and understand the *essence* of the material; establishing what is / is not an appropriate and workable concept and approach to impose upon the computational environment.

Generating a greater ability to interpret and develop ideas through the lens of the material of code and seeing the importance of creating ideas *in*

harmony with the environment has enabled me to develop the 'head' knowledge of how individual elements of code work. This has led to an understanding of the more important, intangible, knowledge of the 'feel' and characteristics of the material.

6.2.2 A Distinct Workflow: Unlike Traditional Craft

Having considered the process in general through contextual research, the practical element of the research provided a means to investigate and understand, in more depth, the *specific* nature and key features of the computational process, highlighting *unique* elements of the computational process. Although the research has highlighted ways in which the approach towards material and process of computational design links with the ideas of Arts and Crafts (and with those of wider traditional craft-practice) there are clear differences and distinctions between the processes. Craft-centred making is not characterized as means of manipulating abstract symbols on screen, as with computation, but through the movement and manipulation of physical material. The physicality of the traditional material is expressed and reflected by a process of creative engagement, characterized as a flow between hand, eye and mind; a process of mental engagement and understanding between maker and material informed by visual and tactile feedback; 'tacit knowledge' (Dormer, 1997). Unlike the computational process in which code has to be written and compiled before the visuals are seen, feedback within the traditional environment is immediate, there is a direct flow between the material and visuals which informs the process. In contrast to the immediate physicality of traditional material, computational material is *created* as part of the process via phases of creation, development and manipulation. Beginning as a purely conceptual 'blank sheet of paper' the process of writing code literally gives shape and form to the material of work, moving from initial concept through iterations of data development, manipulation and experimentation from which the artefact

emerges. Chapter 5 brings together the ideas and work from the two practical projects and highlights the distinctions and characteristics of computational workflow, identifying three key phases or 'layers' of work:

- concept
- data structure
- data detail

Each phase is a necessary part of the cyclical design process and an important part of the work, allowing the development of different aspects of the work, from the outline of the concept, to the manipulation of the detail, and allowing change and development of the work throughout the project. Movement between each of the cycles of the process is common to both the Colorcalm and Moving Wallpaper work and constitutes the 'flow' of the process, allowing the development of a fluid approach to the work. Far from being a straightforward, mechanical, process from idea to product, the process of creative computational work is characterized as a evolving dialogue between the material and visual elements of the work, in which the growth of the computational structure and of the work is generated. Both the Moving Wallpaper and Colorcalm projects allow work and ideas to evolve through several iterative cycles of development and experimentation, generating a feedback loop of 'play' and development during which the specific structure and nature of the work is explored.

The nature of computation means that the development of the structure is inseparable from the development of the artefact. The fact that the final piece of work must show variety of forms, means that the structure - the architecture of the work - must also be flexible, providing enough scope to create a wide variety of shapes and forms. The material must engender flexibility; the ability to generate a wide variety of ideas, shapes and forms. Flow and flexibility must be in-built into the structure.

The process of using code can be summarized as a journey from fluidity to rigidity, an iterative dialogue in which the parameters of the material are constructed, manipulated and explored. Flow between phases of the work, layers of the material, allow and encourage the continual development and evolution of the project. As the work moves from the fluid concept phase right down to the detail of the data the potential for change becomes increasingly less.

For the programmer, the 'invisible' structure which creates the visual is the substance of the work. The process of creating a piece of work is, therefore, about creating and manipulating these 'invisible' structures. The resultant fluid, dynamic visual computational artefact demonstrates tension between the abstract, text based nature of the material of code and the organic, fluid, reactive nature of the artefact. The need to use and generate a flexible, structure becomes as important as the visuals themselves. Concepts, structures, abstractions, which are made to form the architecture of the project, define the direction and visual quality of the project. Each computational material is a unique construction of data organization, data structure and data detail. As the material is structured, so therefore the flexibility and 'flow' are built into the material. Although each project operates through the same generic phases of work (concept, data structure, detail), the construction and the fluidity of each project is different. Each structure, therefore, must embody its own fluidity and flexibility from the process of structuring and manipulating the material.

What has been highlighted, from both the contextual and practical research into the process of computation design, is the idea of fluidity and flexibility. Both the computational structure and the process of building and generating the structure benefit from a type of computational fluidity in order to develop an approach which does not become rigid, or a structure which begins to 'creak and wobble' as seen in the Colorcalm project. Both the Colorcalm and

Moving Wallpaper projects set out to achieve broadly the same type of work. However, the flexibility of structure, code and thinking of the Moving Wallpaper work is reflected in the final piece, which had a greater degree of visual and computational fluidity. Brian Smith (2006) defines a way of problem solving using programming as 'computational flexibility', a way of thinking and designing in the computational environment, in which the design process is informed by a creative use and understanding of computational structures and processes. This project has highlighted that computational flexibility in thought, approach, structure and artefact is key to the use and practice of computation. This flexibility is only achieved through the development, exploration and understanding, gained through the process of making, as a means of understanding specific capabilities of the computational material. The development of the Moving Wallpaper work would not be possible without the development of the Colorcalm work, and the Colorcalm project allowed the possibilities of the medium to be realized and made visible.

Personal and practical experience of the development of practical work reinforced and focused these conclusions into practical findings relating to personal work. The importance of computational flexibility is one which I discovered to be a key element to the computational process and workflow, and one which made a vital contribution to the outcome of the work: this is something which must be founded during the initial stages of the workflow. Spending the bulk of the initial phases of the project developing a clear and well structured conceptual element of the project was vital. Developing a greater number of conceptual prototypes to establish the characteristics and behaviour of the work, rather than the visuals, was important in establishing a strong conceptual base for the work.

Both projects undertaken as part of the research had a similar workflow but each with different emphasis which is reflected in the 'flexibility' of the work.

The Colorcalm project showed less development of the initial conceptual and behavioural elements of the work moving onto the detail of the data elements more quickly. In contrast, the Moving Wallpaper allowed the concept and data structure elements of the work to develop before honing the individual data details. The greater visual and structural flexibility of the Moving Wallpaper project reflected these core differences in process and workflow.

6.3 Characteristics of Material

The previous section has discussed key elements of the computational process, including the way in which the process is a means by which each individual computational material is constructed and developed. The idea of the 'materiality' of code is one which provides a useful way of thinking about and framing the discussion regarding creative usage of code. The contextual review (2.2.2) established the idea of computational material, highlighting the way in which a broader understanding to materiality in the context of creative practice, is one which is supported both by practitioners (Marius Watz) and theorists (McCulloch, 1998), providing a useful platform for the discussion of the rest of the work. Analysis of the practice based research in Chapter 5 has identified that the construction of the computational material is divided into three phases of work, which constitute the layers of the computational material. The computational process iterates through each phase of the work, developing each as a new 'layer' to its overall structure, and developing the computational material as a layered construction of the concept, data and numeric elements. These layers, therefore, provide each project with its specific material qualities as the individual configurations of concept, data and number (classes, functions and algorithms) provide each computational structure its own character and affordance. When considering the material of computation, we must therefore consider the concept, data, numeric layers of the material and the role and significance of each as a

means of engendering a fluid, flexible material which embodies the ability to generate a wide range of visually creative, fluid pieces of work. Each layer of the material will now be considered in turn.

6.3.1 Concept

The concept is developed over the first phases of the work and forms the foundation for the development of the rest of the work. Looking at both the development of both pieces of work, we can see how the definition of the concept impacts upon both the direction and the *flexibility* for the rest of the work. Once established, the central concept (e.g. ideas determining what the computational lines should be, how it to behave, be created, etc) proves to be an important and influential foundation stone upon which the rest of the work is based. Modifications and alterations to the concept have the greatest impact upon the rest of the work. For both pieces of work, the concept phase offers the greatest potential for change and flexibility regarding the direction and nature of the project work. At this point of the project development, ideas are open to interpretation, experimentation and adjustment.

Comparison of the concept development for each project shows that the Moving Wallpaper project develops a wider, more broadly computational approach to the idea of generating organic lines and shapes, and so provides a wider base upon which to develop the rest of the work. Rather than being narrowly defined as single shape the Moving Wallpaper work considers the 'line' as a culmination of ideas (forces) which combine to generate the 'reactive' lines of the Moving Wallpaper project. The conceptual 'flexibility' of the Moving Wallpaper work, in generating a broad concept from which to develop the work, allows for a broader manifestation of a data structure, and a greater range of possible visual and structural outcomes.

6.3.2 Data Structure

The data structure allocates and defines specific classes, functions and methods within the body of the structure, defining the broader nature and architecture of the work. The data structure therefore defines and controls the broad set of computational elements (classes) and determines the bonds and connections (the 'flow') between them. The ability to add, define change, use and modify core data elements within a structure helps define its flexibility. Each data structure can therefore be viewed as the (unique) individual structure for a piece of work. Both the Moving Wallpaper and Colorcalm projects each have quite different means of defining the data concepts.

The ability afforded by the Moving Wallpaper project by its 'modular' approach, defining a conceptual, abstract, discrete set of individual data types (classes), affords more structural fluidity. The computational architecture of the Moving Wallpaper project, therefore, has a greater ability to be enhanced, and added to. It also has a (greater) ability to be changed and altered; it has greater flexibility. 'Flow' between data types and objects is greatly enhanced in the Moving Wallpaper project; each individual object is defined by associations and parameters of other objects, allowing a much greater degree potential for visual flexibility and development.

Flexibility is developed by adding links between objects and by adding parameters between items (e.g. parameters between plant and line objects). For example, the Moving Wallpaper line is constructed by its individual associations with instances of other types of object (Plant, Attribute, etc.)

The structure of the Moving Wallpaper project affords a greater degree of structural manipulation. The nature and construction ('architecture') of the work is more fluid. New objects and data types can be easily added and

included within the structure. In this way, manipulation, and computational flexibility occurs through the development of a fluid data structure (adding more objects and associations to objects). This allows key elements of the work to be manipulated, changed and added, allowing the work to move in different conceptual directions.

In contrast to the Moving Wallpaper project, the data structure of the Colorcalm work is developed in a much more linear fashion. Instead of seeing and understanding the line as a combined series of objects and behaviours, the Colorcalm work structured the project in a more literal linear way, in which a line is defined by a single class in itself. Additions to the structure, i.e. adding different types of lines are generated by duplicating, with slight changes, the line class to create similar 'leaf' and 'branch' classes. Rather than beginning by abstracting the *concept* of the line, (its movement, visuals and behaviour into a broad data structure and classes) the Colorcalm began by putting the visuals and the detail of the line drawing first. The data structure was secondary to the detail of the individual algorithms and variables which generated the line type. The narrow, rigid Colorcalm data structure - based on a few co-dependant classes - began to show its limitations as the scope of the project grew; development of the visual scope of the work quickly revealed the limitations of its structure. In order to accommodate visual development, greater number of variables were added on top of a limited, narrowly based, data structure which quickly began to 'creak' and 'wobble ' under its weight. Added data detail with the limited structure made the computational environment increasingly rigid and hard to manipulate, change or develop. A lack of structural soundness, which put form before structure; culminated in rigid, 'material' which embodied limited amount of scope for visual fluidity change or development.

6.3.3 Data Detail

Data details are the individual variable values which are used throughout the data structure and are used to define specific aspects of each visual, behavioural or structural element of the work. Assigning and changing variables values defines the range and detail of the visual and behavioural element of the work. The ability to assign and manipulate individual data details defines the flexibility of the computational structure, which affects the visual details of the computational work. Once the concept and data structure have been defined and established, the data details control and manipulate small elements of the work. Experimentation with, and manipulation of, these data values are used to make creative, aesthetic decisions about the work. The final phases of both projects focus upon experimentation, manipulation and allocation of the variable values as the final aesthetic decisions about the timings, colours and shapes, etc. are made.

A key element of data detail for any computational project is numeric data. Each project makes extensive use of individual numeric data as a means of defining and allocating values to named visual attributes (e.g. angle, width speed of line, etc.). The values and therefore the visuals, are manipulated via a combination of key, simple, mathematic and logical functions, some of which are outlined in the following table:

Mathematical Function	Example Code
Incrementation and decrementation	<code>x = x+ 1</code>
Random function	<code>x = random (100)</code>
Sine calculations	<code>x = sine (angle) * magnitude</code>
If / then / else statements and decisions	<pre>if (x> 100) { x = 0; }</pre>

Table 6:1 Key mathematical functions used to manipulate variables

Algorithmic manipulation is at the heart of all computational work, and in many instances, manipulation of the visual amounts to manipulation of individual numeric values. To this end each of these simple functions are used, either individually or in combination with one another, to develop alter and change elements of the visual detail for both projects. Although each of these individual mathematical concepts are simple, combinations are used over and over again to provide a variety of interesting shapes, forms and effects and afford a greater amount of flexibility. Altering the context or combining several simple mathematical functions together creates a greater variety of effects. For example combinations of incrementation, 'if' statements and sine functions were experimented with to produce the looping curves and shapes used as a core 'stem' shape in both projects.

Whilst both projects make extensive use of individual data (numeric) elements to set and manipulate visual details of each project, key differences can be seen in the *range* and *spread* (breadth) of data detail used in each project. Whilst the Colorcalm project relies heavily on lots of numeric data to control the details and structure of each plant, resulting in a complexity of individual number values, the Moving Wallpaper project applies a much broader set of data values. These are encouraged by the wider, more flexible data structures of the project, making greater use of different types of data detail (e.g. strings and arrays) to determine the details of the work. The use, for example, of string data as a means of defining the often complex growth pattern, structure and development of parts of the work (stem, branch, leaf, flower petal) provides a greater and more flexible computational vocabulary with which to subtly change and manipulate the structure and development of the visual growth of the work. Complex decisions relating to the organic development and growth are therefore succinctly expressed as data strings ("FBBFBLBLFF**") the interpretation of which is defined within the Engine class. This broader use and understanding of data affords a greater degree of flexibility. Potential for manipulation is made possible by

the broader use and understanding of computational data applied at each stage throughout the Moving Wallpaper project. The Moving Wallpaper project also makes use of classes to combine groups of data elements together into single `Plant` and `Attribute` objects. This allows groups of data detail to be treated and assigned as individual data objects, and provides another level of flexibility by which the data details of the Moving Wallpaper project are handled.

As the project is developed, so the structure and the material of the project is realized. The specific nature of the data structure determines the overall flexibility of the material; each layer has its own means of generating and encouraging the flexibility of the material. The table below is a summary of the findings of this section, and provides an overview of how flexibility is in-built into the layers of the work, and the effect this has on the direction.

	Means of Flexibility	Effect on Work
Concept	The 'flexibility of the concept is determined by an understanding of computation.	Determines the overall aims of the work. Manipulates the direction and scope of the project.
Data Structure	Generates fluidity by creation of a flexible structural 'bonds' between classes.	Changes the behaviour, functionality and timebased nature of the project. Overall variety and change.
Data Detail	Fluidity generated by experimentation with maths and logic.	Changes the visual details of the project.

Table 6:2 Summary of the layers of code, associated flexibility and effect on the visual element of work.

The work of the Moving Wallpaper and Colorcalm projects provided a useful environment during which the importance of building in a flexible, malleable material structure was reinforced in a practical way. Undertaking the practical elements of the research I have learnt the significance of structural flexibility, and demonstrated how a broad application of data can enhance the creative possibilities of the project and develop a flexible, fluid piece of work. The development of the practice changed my own understanding and generated ideas about computation from a simple numerical and algorithmic representation to an appreciation of the importance of a broader understanding of data and data structures.

When undertaking the Colorcalm work much of the use of code was based around a simple understanding of variables i.e. a variable as simple number values that are changed to alter the visuals on the screen. Definition and use of individual numeric values were seen as the key to manipulating the visuals of the Colorcalm work this is reflected in the visual and the structure.

The development and use of data in a broader context, which included wider use of data objects, strings etc. was adopted for the Moving Wallpaper project. A broader understanding of data thus informed the work and allowed the development of a computational structure which allows for a greater degree of manipulation. Development of the work enabled the researcher to see the importance of building computational flexibility into the material of the work by making best use of 'larger' data structural items (e.g. classes) and allowing the conceptual element of the project to develop before the visuals. Greater emphasis on larger data structural items allows for a computational, more flexible, structure.

6.4 Visuals in Relation to Data Structure

The emphasis of this research is on the *process* of computational design and is therefore not an evaluation of the aesthetic success of the work. Although a significant aspect of each project involves the creation of aesthetic work, consideration of the visuals is done to inform the understanding of computational process and structure. It is, however, worth briefly considering key elements of the visual aspect of the Moving Wallpaper and Colorcalm projects here.

Based on the organic, botanic aesthetic of the Arts and Crafts work, specifically the 'Willow' and 'Jasmine' wallpaper design of William Morris, the core visual inspiration and direction for both projects was informed by organic lines and forms. Each project tackled the idea of developing a computational plant-like structure which defined specific elements and attributes of botanical structures (i.e. stems, leaves, flowers, etc.) but which also embodied a degree of variance and visual fluidity to echo the 'random' nature and structure of organic forms. A central characteristic of the visual development of the project is summarized by the idea of controlled variance. The work has to be controlled enough to generate well defined plant, leaf and flower shapes and types, but fluid enough to encompass the visual variance of these types of forms i.e. to include variance of size, form and movement within an overarching botanical structure.

In this way, the visual element of the work reflects the structural concerns and tensions of the work, which is both rigidly structured and defined according to the logical rules and syntax but also fluid and flexible enough to embody and embrace a range of outcomes and possibilities; a defined structure in which the generative (random) nature of the environment can be expressed.

Structural and computational differences between the Colorcalm and the Moving Wallpaper project are, therefore, reflected by the visual outcomes of each project: The narrowly defined, concept and data structure of the Colorcalm work are mirrored by the visuals of this piece of work. The repeating patterns and line drawings contain some element of visual organisms to them but remain visually flat and unresponsive on screen. Each of the drawn plant structures maintains a high degree of similarity with some variation in colour and leaf structure. The limited visual range and 'responsiveness' of the Colorcalm work therefore reflects the narrowly defined concepts and data structure of the programme.

Similarly, the conceptually and computationally broader structure of the Moving Wallpaper project (in which a greater use and understanding of data is developed) is also reflected by the visual range of the work, and the responsiveness of the graphics on screen, which begin to engender a fluid organic look and behaviour. The basic Moving Wallpaper computational structure engenders a higher degree of flexibility and nuance; a greater range of visuals can be developed from the structure (fig. 4.28), and the visuals on screen move and respond to external input. The more flexible and fluid data structure of the Moving Wallpaper project work is reflected by the visually varied and fluid results on screen.

6.5 Reflections on Methodology

The methodology employed in this project is based on a well-established practice-based research approach: a series of reflective practical projects intended to address the research area, and a contextual review. Where this research has differed from the basic model is that here the contextual review (in particular the Arts and Crafts case study) has not only informed the conceptual framework and the production method of the work, but it has also inspired and shaped the aesthetic direction of the design work (the natural plant forms which reflect not only Morris' wallpaper, but also the

characteristics of recursion in computer programming). The distinctions between process and form have therefore at times been difficult to delineate, but rather than a more general 'reflection', the descriptive, step-by-step, project-based method has intended to identify the key elements and characteristics of programming which might be most useful to peers who are also moving from using design software to using code. Documenting the development of the work, and providing examples of individual algorithms and data structure gives insight which other practitioners may find useful as a basis for their own creative practice. It is hoped that this will be something which informs and encourages a wider exploration and engagement with code as a valuable creative material.

Reflection upon the overall development of this project has highlighted limitations with the specific method and overall methodology of the research. Learning to create and write using the correct programming language, syntax and grammar was sometimes a time-consuming process which impacted upon the time spend developing ideas for the work. Becoming familiar with the correct language and syntax for each phrase or element of the project meant that much of the research time was spent trying to overcome and understand the basic aspects of the language. Technical problems and errors in the code made the research process itself very time consuming which limited the amount of creative experimentation. Further limitations relate more specifically to the overall methodological approach taken by the project i.e. the use of contextual review to steer the aesthetic aims of the work, and the means of disseminating and involving a wider audience as part of the practical process.

Looking back on the project, it is now possible to see that whilst the contextual review provided an important conceptual and aesthetic focus and direction for the work, the project was perhaps overly focused on developing an Arts and Crafts aesthetic. Spending more time allowing the visual focus of

the work to grow and develop through experimentation with code may have produced work which was led by the core properties of the programming language, more directly reflecting the aesthetic qualities of computational material. Whilst the aesthetics of the botanical shapes and forms provided a very useful visual focus for the project, it can be argued that this point of reference perhaps was leading the project too heavily. A 'freer' experimental methodology may have produced different, more material-led, results.

A further limitation of the project's methodological approach lies in the area of open source production and dissemination. The open source community surrounding the Processing environment is a close and developing one which encompasses a broad range of input from artists, designers and developers who involve themselves in code-based practice. This research benefited from the online Processing discussion lists and help fora during the process of learning how to code. On reflection, however, the research could perhaps have sought greater involvement from this community as part of its methodology by making the design work public online; seeking 'live' ongoing input throughout the stages of development and using comments and feedback from the community to help inform the progress and direction of the work. Creating stronger links with the open source community could have enhanced the direction and the findings of the research, a missed opportunity which may perhaps be addressed in the future during further research.

6.6 Future Research

The scope of this research is centred around the development of an understanding concerning the use and process of computation in a creative design environment. Emphasis of this research has therefore been on the contextualization and process of computation, identifying its situation within the broader landscape of design and highlighting key elements and

characteristics of the computational process. Building upon the findings of this research which make a link between traditional craft and computational design, further detailed investigation into the specific nature, characteristics and aesthetics of the computational *object* may be a worthwhile area of future research. This could include a closer examination of the computational object, its place within the tradition of craft, and a re-evaluation of the aesthetic and visible qualities of computationally generated work within the context of traditional craft.

A second area of study arising from the research concerns the process of using computation itself: The research suggests that a requirement for creative engagement with computation stems from the developed understanding and engagement with the 'material' of code. The research has shown that a developed understanding of the language structure of code is necessary for creative engagement. Computationally flexible thinking arises from a concentrated, developed understanding of the material. This idea has implications for the development for future creative practitioners in this field. Against a contemporary backdrop in which the demands of software development and programming work increase and the commercial roles of 'designer' and 'programmer' are drifting apart, how will future creative practitioners develop a use and understanding of the material of code? What ways are there for developing the concepts of computational understanding into design practice? This is perhaps an issue for educational as well as creative establishments to explore.

This research represents a fruitful personal journey for the researcher and it is hoped that future work and investigation into this area will prove to be as rewarding.

Appendix 1: Colorcalm Code

Appendix 1.1: Simple Line Drawings

Sketches: *lineDraw1*, *lineDraw1b*, *lineDrawingSimple3*,

```
float x ;
float y;
float angle;
float radius;
float w ;
float inc;
int timer = 0;
-----

void draw () {
  if (timer % 15 == 0 ) {
    fill (0, 50, 0, 10);
    rect (250, 250, 500, 500);
  }
  fill (256,256,256);
  translate (200, 0);
  if (timer % 1 == 0) {
    lineDraw (x, y, angle, radius);
  }

  if (w < 2 ) {
    reset ();
  }

  timer ++;
}
-----

void reset () {
  x = 0;
  y = 0;
  angle = 60;
  radius = 10;
  w = 20;
  inc = 1;
}
-----

void lineDraw (float xpos, float ypos, float ang, float r) {
  translate (xpos, ypos);
  xpos = cos (radians (ang)) * r;
  ypos = sin (radians (ang)) * r;
  pushMatrix ();
  rotate (radians (ang));
  rect (0, 0, 8, w);
  popMatrix ();
  x += xpos;
  y += ypos;
  angle += sin (timer/10)*4;//+= inc;
  w -= 0.4;
}
```


Appendix 1.2: Branches

Sketches: *lineDrawingSimple3fOOP3*, *lineDrawingSimple3fOOP4*

Line

```
-----  
float x, y, angle, radius, w, inc, timer;  
float xStart, yStart, wStart, angStart;  
float xpos, ypos;  
int segNum, segCount; // amount of segments in the line  
float wDecrement;  
  
-----  
void init () {  
  x = xStart;  
  y = yStart;  
  w = wStart;  
  angle = random (90);  
  segCount = 0;  
}  
  
-----  
void run () {  
  if (w < wDecrement) {  
    init ();  
  }  
  if (timer % 1 == 0 ) {  
    plotPoint (x, y, angle, radius);  
  }  
  
  if (segCount == 30) {  
    makeBranch (x, y, angle, w);  
  }  
  timer++;  
}  
  
-----  
void plotPoint (float xpos, float ypos, float ang, float r) {  
  fill (256, 256, 256);  
  noStroke();  
  translate (xpos, ypos);  
  xpos = cos (radians (ang)) * r;  
  ypos = sin (radians (ang)) * r;  
  pushMatrix ();  
  rotate (radians (ang));  
  rect (0, 0, 8, w);  
  popMatrix ();  
  segCount++;  
  
  incrementShape (xpos, ypos);  
}  
  
-----  
void incrementShape (float xpos, float ypos) {  
  x += xpos;  
  y += ypos;  
  angle += sin (timer/10)*4;//+= inc;  
  w -= wDecrement;  
}  
  
-----  
void makeBranch (float x, float y, float angle, float w) {  
  branchNum ++;  
  myLine[branchNum] = new Line (x, y, angle-10, 10, w);  
}  
  
}
```

Appendix 1.3: Variance and Difference

Sketches: *lineDrawingSimpleInhertance*, *lineDrawingSimpleInhertance2*, *lineDrawingSimpleInhertance2b*

Line

```
-----  
float x, y, angle, radius, w, timer;  
float xpos, ypos;  
int segNum, segCount;  
float wDecrement;  
boolean go;
```

```
-----  
run ()  
plotPoint ()  
incrementShape ()  
-----
```

Mainline extends Line

```
-----  
float xStart, yStart, wStart, angStart;  
-----  
init ()  
  
run () {  
  super.run();  
  if (w < wDecrement) {  
    init ();  
  }  
  if (segCount == 20) {  
    makeBranch (x, y, angle-10, 10, w, -0.8);  
    makeBranch (x, y, angle+10, 5, w, +0.6);  
  }  
}  
  
makeBranch () {  
  branchNum ++;  
  myLine[branchNum] = new Branchline (x, y, angle, r, w, distort);  
}  
  
incrementShape () {  
  super.incrementShape ( xpos, ypos);  
  angle += sin (timer/10)*4;//+= inc;  
  w -= wDecrement;  
}  
}-----
```

Branchline extends Line

```
-----  
float ang_inc;  
float distort;
```

```
-----  
void run () {  
  super.run ();  
  if (w < wDecrement) {  
    go = false;  
  }  
  
}  
-----
```

```

void plotPoint () {
super.plotPoint (xpos, ypos, ang, r);

if (segCount > 10) {
float leafLength = 10;
float leafAngle = ang+80;
float leafDist = 0.5;

for (float i=0; i<r; i+=leafDist) {
float xPoint = cos (radians (ang)) * i;
float yPoint = sin (radians (ang)) * i;
float xEnd = cos (radians (leafAngle)) * w*10 + xPoint;
float yEnd = sin (radians (leafAngle)) * w*10 + yPoint;
line (xPoint, yPoint, xEnd, yEnd);
xEnd = cos (radians (leafAngle-120)) * w*9;
yEnd = sin (radians (leafAngle-120)) * w*9;
line (xPoint, yPoint, xEnd, yEnd);
}
}
}

```

```

-----

void incrementShape (float xpos, float ypos) {
super.incrementShape ( xpos, ypos);
angle += ang_inc;
ang_inc += 0.9 * distort;
w -=0.1;
}

-----

```

Appendix 1.4: Colour, Shape and Form

Sketches: *cc_basicVersion*, *cc_basicVersion2*, *cc_march_02*, *cc_march_03*

Line

```
-----  
float x, y, angle, radius, w, timer;  
float xpos, ypos;  
int segNum, segCount;  
float wDecrement;  
boolean go;
```

```
-----  
run () {  
  if (ratio < 80 ) { inc = 1; };  
  if (ratio > 220) { inc = -1 ; }  
  ratio += inc;  
  frequency = 0.05;  
  magnitude = frequency * ratio;  
  timer += frequency; //timer += 1;  
  plotPoint (x, y, angle, radius);  
}
```

```
plotPoint ()  
incrementShape () {  
  tg -= int (random (-10, 10));  
  if (tg < -200) { tg =-200;}  
  if (tr < -200) { tr =-200;}  
  if (tg >456) { tg=456;}  
  if (tr>456 ) {tr=456;}  
  tr+= int (random (-10, 10));  
}  
}
```

Mainline extends Line

```
-----  
float xStart, yStart, wStart, angStart;  
-----
```

```
init ()
```

```
run ()
```

Branchline extends Line

```
-----  
run ()
```

```
plotPoint ()
```

```
incrementShape ()
```

Appendix 1.5: Petals and Flowers

Sketches: cc_march05_Flower2b, cc_march05_Flower3, cc_march05_Flower3b, cc_aprilTest2, cc_aprilTest3

Line

```
-----  
x, y, angle, radius, w, timer;  
xpos, ypos;  
segNum, segCount;  
wDecrement;  
go;
```

```
-----  
run () {  
  plotPoint ()  
  incrementShape () {  
-----  
-----
```

Mainline extends Line

```
-----  
xStart, yStart, wStart, angStart;
```

```
-----  
init ()  
run ()  
-----  
-----
```

Branchline extends Line

```
-----  
run ()  
plotPoint ()  
incrementShape ()  
-----  
-----
```

Flower

```
-----  
ox, oy;  
plx, ply;  
radius, angle;  
fSize, fSizeMin, lineCount;  
increaseAngle = 20;  
depth;
```

```
-----  
drawLine () {  
  
  if (fSizeMin < fSize) {  
    ellipse (ox, oy , 7, 7);  
    stroke (tr*3, tg*3, tb*3,50);  
    strokeWeight (0.9);  
  
    for (int i=0; i<500; i++) {  
      angle += radians (60);  
      radius = sin (radians (5*angle/3))* fSizeMin;  
      plx = cos (radians (angle)) * radius+ ox ;  
      ply = sin (radians (angle)) * radius+ oy ;  
      line (ox, oy, plx, ply);  
      lineCount ++;  
    }  
    fSizeMin+=4;  
  }  
}
```

Appendix 1.6: Leaf Class and Final Pieces

Sketches: cc_aprilTest4b, cc_aprilTest4c, cc_aprilTest5a, cc_aprilTest5b, cc_aprilTest5c, cc_May1, cc_May2, cc_May2b, cc_May3, cc_May4, cc_May4b, cc_May5_testb, cc_May6.

Line

```
-----  
x, y, angle, radius, w, timer;  
xpos, ypos;  
segNum, segCount;  
wDecrement;  
go;
```

```
-----  
run () {  
  plotPoint ()  
  incrementShape () {  
-----
```

Mainline extends Line

```
-----  
xStart, yStart, wStart, angStart;  
-----  
init ()  
run ()  
-----
```

Branchline extends Line

```
-----  
run ()  
plotPoint ()  
incrementShape ()  
-----  
-----
```

Flower

```
-----  
ox, oy;  
plx, ply;  
radius, angle;  
fSize, fSizeMin, lineCount;  
increaseAngle = 20;  
depth;
```

```
-----  
drawLine ()  
-----  
-----
```

Leaf extends Line {
 counter, leafBreadth;

```
-----  
count ()
```

```
run ()
```

```
plotPoint ()
```

```
incrementShape ()
```

```
  super.incrementShape (xpos, ypos, r_);
```

```

float lineLength = sin (radians (counter*2)) * w * leafBreadth;
strokeWeight (0.8);
float xEnd = cos (radians (angle +90)) *lineLength;
float yEnd = sin (radians (angle +90)) *lineLength;
float xEndb = cos (radians (angle -120)) *lineLength*2;
float yEndb = sin (radians (angle -120)) *lineLength*2;

angle += sin (timer)*magnitude;

w -= wDecrement ;

counter++;
float xEnd2 = (cos (radians (angle +90)) * lineLength) + xpos;
float yEnd2 = (sin (radians (angle +90)) * lineLength) + ypos;
xEnd2b = (cos (radians (angle -120)) *lineLength*2)+ xpos;
yEnd2b = (sin (radians (angle -120)) *lineLength*2)+ ypos;

fill (tr*1.2, tg*1.2, tb*1.2); // leaf fill colour
beginShape ();
vertex (0, 0);
vertex (xEnd, yEnd);
vertex (xEnd2, yEnd2);
vertex (xpos, ypos);
endShape ();
fill (tr*1.2, tg*1.2, tb*1.2);
beginShape ();
vertex (0, 0);
vertex (xEndb, yEndb);
vertex (xEnd2b, yEnd2b);
vertex (xpos, ypos);
endShape ();

}

}

```

Appendix 2: Moving Wallpaper Code

Appendix 2.1: Concept Stage

Example 1

```
Line
-----
startLoc ;
target;
ballCount;
Ball ballArray [];
grow;
-----
run ( )
updateTarget ();
drawBalls ();
calcDistanceAndAddBalls ();
makeBallsDraggable();

Ball
-----
loc;
vel;
draggable;
-----
setLoc ( );
drawBall ();
dragMe ( );
```

Example 2

Early stages, simple line, trail of balls.

```
Spring
-----
stiffness = 0.2;
damping = 0.7; // add some friction
mass = 2.0;
springLength = 5;
-----
updateSpring ( );

Line
-----
startLoc ;
target;
ballCount;
Ball ballArray [];
Spring s;
grow;
-----
run ( )
updateTarget ();
drawBalls ();
calcDistanceAndAddBalls ();
makeBallsDraggable();
makeAllBallsSpringy ( );

Ball
-----
```


Appendix 2.2: Branching Structure

Example 1

Sketches: *LinkingLineTests3*

Engine

```
-----  
interpretChar ( char letter, Target t)  
-----  
if (letter == 'F') { t.timer += 100; }  
  
if (letter == 'B') { t.branch(); }
```

Target

```
-----  
Vector3D vel, loc, acc;  
r = 3; // magnitude (radius)  
angle;  
String s;  
charCount = 0;  
timer = 0;  
counter = int (random (360));  
hi = 400;  
lo = 20;  
magnitude = 100;// random (lo, hi);  
inc = .1;  
Engine e = new Engine ();  
isAlive = true;  
lineNum;  
-----
```

```
addLine ( )  
runTimer ( )  
getLetter ( )  
calcForce ( ) ??  
update ( )  
render ( )  
branch ( )  
-----
```

the String, s, is defined in the Target object and interpreted by the Engine (e)

```
s = "FBFBF<";  
  
-----
```

Line

```
-----  
Ball  
  
-----
```

Spring

Appendix 2.3: Visual and Behavioural Experimentation

Example 1:

Sketches: *LinkingLineTests3, LinkingLineTests3_outlines*

Engine

```
-----  
interpretChar ( char letter, Target t)  
-----  
if (letter == 'F'){t.timer += 100; }  
if (letter == 'B') { t.branch(); }  
...  

```

Target

```
-----  
s = "FBFBF<"; (stem)  
s = "FFFF" (branch)  
...  

```

Line

```
-----  
Target t  
calcBallWidth ( ) {  
width = 180 / ballCount  
}  
  
drawBalls ( ) {  
beginShape (QUADS);  
vertex (b.loc.x -b.w, b.loc.y);  
vertex (prevb.loc.x-prevb.w, prevb.loc.y);  
vertex (prevb.loc.x+prevb.w, prevb.loc.y);  
vertex (b.loc.x +b.w, b.loc.y);  
endShape ();  
}  

```

Ball

Spring

Example 2

Sketches: stage3.0c_Line_Forces_test_LRG // stage3.1_Line_Forces_27_Feb // Line_Forces_27Feb_new

Engine

Target

Line

-- react to mouse movement
void calcForces () {
 // add a force to each of the balls..
 for (int i=1; i<=ballCount; i++) {
 Ball b = (Ball) ballArray.get (i);
 Vector3D force = b.loc.sub (b.loc, mouse);
 // find 'force' diff between ball and mouse;
 // find the distance and the angle
 float d = force.magnitude ();
 d = 1/d * 200;
 float a = force.heading2D ();
 Vector3D newForce = new Vector3D (cos (a) * (d *.1), sin (a) * (d *.1));
 newForce.limit (.15);
 b.addForce (newForce);
 b.update ();
 }
}

'grow' behaviour..// check this.
using new variable: lineExpandAmount, maxWidth

lineWidth += lineExpandAmount
if (b.w > maxWidth) {
 lineExpandAmount = - lineExpandAmount
}

Ball

Spring

Example 3

Sketches: *March1, Moving Wallpaper_March1_b, Moving Wallpaper_March_1B, March1C_silhouette, silhouette2*

Engine

Target

Line

```
-----  
...  
Line ( ) {  
...  
if (parentLinePos == 0) {  
    maxWidth = 2;  
    }  
    else {  
    maxWidth = 30;  
    }  
    }  
...  
void calcBallWidth ( ) {  
...  
float widthInc = (90/ float (ballCount));  
...  
if (b.w >= maxWidth) {  
    lineExpandAmount = -lineExpandAmount;  
    addNewLine (parentLinePos);  
    if (lineWidth < maxWidth) {  
        lineWidth+= lineExpandAmount;  
    }  
    if (lineWidth < 0.1) {  
        removeLine ();  
    }  
    }  
}  
  
void timer ( ) {  
    time++;  
    if (time == 100 && parentLinePos == 0) {  
        addLine ();  
    }  
    }  
void addLine ( ) {  
    Target t = new Target (sinLUT, new Vector3D (width/2, height/2, 0));  
    a.add (t);  
    }  
removeLine ( ) { }  
--- blur???
```

Ball

Spring

Appendix 2.4 : Extending the Vocabulary

Example 1

Sketches: Moving WallpaperBlurEdgeATTRIBUTESMarch18, Moving WallpaperBlurEdgeATTRIBUTESMarch19, March19B, April1, April1_segs&ends.

```
setup ()
String stemString = "F+L-LF-LFBF-fP*";
String branchString = "F+L-LF+L-LF+f*";
String leafString = "FF*";
String flowerString = "-p-p-p-p-p-p-p-p-p-p-p-p*"; ///
String petalString = "FF*"
Stem = new Attributes (1, stemString, 11/2, 12/2 );
Branch = new Attributes (1, branchString, 11, 12);
Leaf = new Attributes (6, leafString, 11, 12);
Petal = new Attributes (12, petalString, f1, f2);

Engine
-----
if (letter == 'B') {t.addBranch(Branch); }
if (letter == 'P') {t.addPlant (Stem); }
if (letter == 'L') {t.addBranch (Leaf); }
if (letter == 'f') {t.addBranch (Flower); }
if (letter == 'p') {t.addBranch (Petal); }

Attributes
-----
color fillColor;
color c1, c2;
float maxW; // maxwidth
String s;
String type;
-----
color colorBlend (color c1_, color c2_, float fract) {
  return color(r1 + r2 * fract, g1 + g2 * fract, b1 + b2 * fract);
} -- plus different types of Constructor ??

Target
-----
Target (Attributes atts, Vector3D loc_, float angle_, int parent, int depth_) {
  addLine(atts); // attach a line object to this target.
}
void addLine (Attributes atts) { }
void addBranch (Attribute atts) { }

Line
-----
Attributes lineType; Line (Attributes atts, Target t_, int currentLine, int
parentLine) {

Ball
-----

Spring
```

Example 2

Sketches: Moving Wallpaper...AttributesMarch19B, April1, April1c, April1_segs&ends. April2, April2_test, April2_B.

```
setup ()
String [] seqArray = {"F++L+L-L--L", "FF+L-L"};
String [] endArray = {"F+LL-L*", "F+LL-L*", "Ff*"};
String stemString = "F+L-LF-LFBF-fP*";
String branchString = "SSSSSE*" ;
String leafString = "FF*";
String flowerString = "-p-p-p-p-p-p-p-p-p-p-p-p*";
String petalString = "FF*"
```

```
color l1 = color (47, 60, 20);
color l2 = color (47, 140, 20);
color f1 = color (132,105,206);
color f2 = color (49,21,37);
```

```
Stem = new Attributes (1, stemString, l1/2, l2/2 );
Branch = new Attributes (1, branchString, l1, l2);
Leaf = new Attributes (6, leafString, l1, l2);
Petal = new Attributes (12, petalString, f1, f2);
```

Engine

```
-----
String convertString (String s) {
if (c == 'S') { ...}
if (c == 'E') {...}
if (c != 'S' && c != 'E') {...}
}
```

Attributes

```
-----
new constructors --
Attributes (float mw, String s_, color c1_, c2_)
Attributes (String s_ )
-- string arrays
Attributes (float mw, String [] s_, color c1_, c2_)
Attributes (String [] sa)
```

Line

```
-----
Ball
-----
```

Spring

Appendix 2.5: Translation and Rotation

Example 1

Sketches: *translateExample10b / 10c / translateExample12 / 12c / Moving Wallpaper_June13 / Moving Wallpaper_June13_LRG2Color / Moving WallpaperJune14 --- Moving Wallpaper_June23_unfurl_shape3*. Key elements of code taken from *Moving Wallpaper_June13*

```
Line
-----
Line ( ) {
float spin; (initial rotation value)
float spinInc; // amount each segment is incremented (to get a curve)
float unfurlTarget = 2;

////////////////////////////////////

void fineParents ( ) {
if (lineType.type == "petal") {
spinInc = 20;
spin = 60; }
else {
spinInc = 0;
spin = 90; // random (40, 100);
}
}

////////////////////////////////////

void run ( )
if (lineType.type == "petal") {unfurl (); // rotate }}

void unfurl ( ) {
float spinIncDiff = unfurlTarget - spinInc;
spinInc += spinIncDiff * 0.008; }

////////////////////////////////////

void drawBalls ( ) {
for each ball ...
float x_angle = sin (radians (spin-i*spinInc)) *r; // xpos
float z_angle = cos (radians (spin-i*spinInc)) *r; // zpos
float xpos = cos (direction) * x_angle;
float ypos = sin (direction) * x_angle;
translate (xpos, ypos, z_angle);
b.screenLoc.x = screenX (0, 0, 0);
b.screenLoc.y = screenY (0, 0, 0);
b.screenLoc.z = screenZ (0, 0, 0);
}
```

Ball

Spring

Appendix 2.6: Final Development and Variations

Example 1 (Plant Class)

Sketches: *Moving Wallpaper_June27_PlantTypes / Moving Wallpaper_July1_PlantTypes (added petal Array)*
/Moving Wallpaper_July1_PlantTypes2/_b

```
setup ()
plantList = new Plant [1];
plantList[0] = new Plant (Stem, Branch, Leaf, Flower, Petal, 20, 300, 68);
String [] petalArray = { "FF*", "FFF*"};
```

Engine

Plant

```
Attributes Stem, Branch, Leaf, Flower, Petal;
float lo, hi;
int petalNum;
// values the same for all plants //
float radius = 7;
float _alpha = 100;
float unfurlSpeed = 0.06;
float decayRate = 0.8;
// Constructors //
Plant (Stem, Branch, Leaf, Flower, Petal) { }
Plant (Stem, Branch, Leaf, Flower, Petal, lo_, hi_, petalCount) { }
```

Attributes

Target

Line

```
void setTargets () {
  if (lineType.type == "petal") {
    spinIncTarget = 2;
    spinInc = 50;
    spin = 90;
  }
  if (lineType.type == "leaf") {
    spinIncTarget = random (10);
    spinInc = spinIncTarget+5;
    spin = 90;
  }
  if (lineType.type != "leaf" && lineType.type != "petal") {
    spinIncTarget = 0;
    spinInc = 0;
    spin = 90;
  }
}
```

Ball

Spring

Example 2

```
setup ()

Engine

Plant {
-----
float decayRate = 0.3; //
float refurlRate = 0.02;
float blurRate = 0.05;
boolean decay = false;
boolean flowerBlur = false;
boolean leafBlur = false;
boolean drawLines = false;
float blurAmount = 10;
float blurAlphaFraction = 0.2;
int blurNum = 1;
int minAngle = 90; // min angle for 'spin'
int maxAngle = 90;
int bCount;
int fCount;
boolean leafFollowParentColor = true; //
boolean flowerFollowParentColor = true;
float leafGrowth = 0.95; // how much leaves grow
float leafStart = 2.0f; // starting size of leaf
float flowerGrowth = 0.7; // how much subsequent flowers
float flowerStart = 1.0f; // starting size of flower
int[] flowerDev = { -4, 15 }; // min and max amounts flower can deviate

Attributes
-----
float unitLength; // how long each F section is
float sineLength; // how long the sine curve is for line
float sineStart;
float minWidth;
float deviation;
float angle; // = 90; // (spin); / default value
int petalNum; // = 15;
float petalAngle; // = 45;
float lo ; //
float hi ; //
-----
Target
-----
Line
-----
calcBallWidth ( )
float w = sineStart;
float magnitude = lineType.maxW;
for (int i=0; i<=ballCount; i++) {
float widthInc = (lineType.sineLength / (ballCount));
b.w = (sin (radians (w))*magnitude)+minWidth;
w += widthInc;
}
-----
Ball
-----
Spring
```

Appendix 3: Research Paper

During the course of the research, a paper was successfully submitted and presented at Siggraph 2006. The title of the paper was 'New Media, New Craft?' and addressed the initial links made between computation and the Arts and Crafts movement. The abstract of the paper reads as follows:

This paper is a reflective study, which considers the use, role, and status of computer programming (when used in a creative context) within the broader context of the ideals and ethos of the late 19th century Arts and Crafts Movement. It seeks to draw comparisons between the role of programming as a means of understanding and manipulating the "material" of the computer environment, and the ethos and attitudes of the craft environment in which artists work with traditional materials. Looking beyond the physical differences between the types of process and artefact involved, this study highlights important areas of commonality of mental approach and attitude which link the ethos of traditional crafts with that of computational artists and designers. It is the contention of this paper that programming, like the traditional crafts, provides a way in which creative people can manipulate or "sculpt" the material of the computer environment. Although it is a reflective and analytical study, the foundation for this discussion derives from creative practical experience and expresses a concern that the role of programmer simply as "engineer" should, in the light of much creative computational work, be re-assessed and re-examined.

Richardson, A., 2006. New media, new craft?. In: Siggraph 2006: *The 33rd International Conference and Exhibition on Computer Graphics and Interactive Techniques*. Boston, Massachusetts, USA 30 July - 3 August 2006. Boston, USA.

Appendix 4: CD of Colorcalm and Moving Wallpaper Work

The CD (attached) contains source documents and code for both the Colorcalm and Moving Wallpaper projects which can be accessed as an offline version of the web site. Both projects and source code are available online at:

http://www.random10.com/colorcalm_research/

http://www.random10.com/movingwallpaper_research/

Whilst every effort has been made to make sure that the work is fully accessible, the nature of the files (exported as java applets) means that performance will vary according to the browser and operating system on which they are run. The Moving Wallpaper work in particular can cause Macintosh based browsers to freeze, and may require the browser to be re-started. With this in mind, the CD also includes a version of the Moving Wallpaper work exported as a series of Macintosh formatted stand-alone applications, which should prove to be more stable for the Mac OS platform. Some of these pieces require a camera attached to the computer, and are set up to run from a Macbook or iMac. In most cases clicking the mouse will trigger new plants to grow on screen. In the case of Moving Wallpaper work, mouse movement often makes the images move.

It is also possible to run the source code from with the Processing environment which is available for download at <http://www.processing.org>.

References

All online material accessed between January 2008 and July 2010

Alexander, A. Goriunova, O. Mclean, A. & Shulgin, A., 2003. *runme.org - say it with software art!* [Online]. Available at: <http://www.runme.org/>.

Andujar, D. et al., 1996. *irrational* [online]. Available at: <http://www.irational.org/irational/>.

Antonelli, P., 1999. Introduction In: J. Maeda, 1999. *Design By Numbers*. Cambridge, Massachusetts: The MIT Press

Austin, T. & Doust, R., 2007. *New media design*. London: Laurence King.

Aydin, E. & Budak, B., 2005. *The Work of Art in the Digital Age* . In: International Symposium of Interactive Media Design (ISIMD). Istanbul, Turkey 5-7 January 2005.

Baudrillard, J., 1990. Ecstasy of communication. In: H. Foster, ed. 1990. *Postmodern Culture*. London: Pluto Press. pp.126-134.

Berzowska, J., 1998. *Computational expressionism: A study of drawing with computation*. MSc. Massachusetts Institute of Technology.

Blokland, E. 2007. RandomFont Beowolf (Interview with Erik van Blokland). In: B. Fry, & C. Reas. 2007. *Processing: A programming handbook for visual designers and artists*. Cambridge, Massachusetts: The MIT Press, pp.169-170.

Bøe, A., 1979. *From gothic revival to functional form*. New York: Da Capro Press.

Bolter, J.D. & Grusin, R., 2000. *Remediation: understanding new media*. Cambridge, Massachusetts: The MIT Press.

British Council Arts, 2005. *My World – crafts and applied arts – British Council Arts*. [Online] Available at: <http://www.britishcouncil.org/arts-aad-design-crafts-and-applied-art-my-world.htm>.

Brown, D., 2007. *Play/create homepage*. [Online]. Available at: <http://www.play-create.com/>.

Brown, D., 2010. *Daniel Brown's*. [Online]. Available at: <http://www.danielbrowns.com/>.

Brown University. 2008. *Jared Tarbell*. [Online]. Available from: <https://wiki.brown.edu/confluence/display/mcm1700n/Jared+Tarbell>.

Bunnell, K., 2004. *Craft and digital technology*. In: World Crafts Council 40th Anniversary Conference. Metsovo, Greece, 2004. Falmouth College of Arts.

Burgoyne, P. & Faber, L., 1999. *Reload: browser 2.0: The internet design project*. London: Laurence King.

Burns, D., 1992. *Designers on mac*. Tokyo: Graphic-sha Publishing.

Burton, E., 2007. Sodaconstructor (Interview with Ed Burton). In: B. Fry, & C. Reas. 2007. *Processing: A programming handbook for visual designers and artists*. Cambridge, Massachusetts: The MIT Press, pp.263-264.

Carlisle, H., 2002. *Towards a new design strategy: A visual and cultural analysis of small-scale pattern on clothing*. Ph. D. Nottingham: Nottingham Trent University.

- Carlisle, H., 2007. The craft of organic programming. In: *New Craft - Future Voices, International Conference*, University of Dundee, Scotland, 04-06 July 2007.
- Cox, G. McLean, A. & Ward, A., 2001 *The aesthetics of generative code*. [Online] Available at: <http://www.generative.net/papers/aesthetics/paper.doc>.
- Cramer, F., 2002. *Concept, notations, software, art*. [Online] Available at: http://www.netzliteratur.net/cramer/concepts_notations_software_art.html.
- Cramer, F., 2003. *Exe.cut[up]able statements: The insistence of code*. In: H. von Gerfried Stocker & C. Schopf, ed. 2003. *Code – The language of our time: Ars Electronica 2003*. Osterfildern-Ruit: Hatje Cantz, pp.98-103.
- Crimp, D., 1990. On the museum's ruins. In: H. Foster ed. 1990. *Postmodern Culture*. London: Pluto Press, pp.43-56.
- Crow, D., 2008. Magic Box. *Eye* 18 (70), pp.20-25.
- Crowley, D. & Jobling, P., 1996. *Graphic design reproduction and representation a critical introduction - reproduction and representation since 1800*. Manchester: Manchester University Press.
- Crutzen, C. & Kotkamp, E., 2006. *Object Orientation*. In: M. Fuller, ed. 2008. *Software studies a lexicon*. Cambridge, Massachusetts: The MIT Press. pp.200-207.
- Cumming, E., 1991. *The Arts and Crafts movement*. London: Thames & Hudson.

Davis, J., 2002. *Flash to the core*. USA: New Riders Publishing.

Davis, J., 2004. Dynamic abstraction machine. In: J. Maeda. 2004. *Creative code*. London: Thames & Hudson, p.142.

Davis, J., 2009. *Joshua Davis Studios*. [Online]. Available at:
<http://www.joshuadavis.com/>.

Design Museum, 2002. *Web Wizards- Design, Architecture, Fashion – Design Museum London* [Online]

Available at: <http://designmuseum.org/exhibitions/Online/web-wizards>.

Design Museum, 2006. *Design library – Design Museum London* [Online]

Available at: <http://designmuseum.org/design/>.

Dormer, P., 1997. *The culture of craft*. Manchester: Manchester University Press.

Dormer, P., 2001. *Meanings of modern design*. London: Thames & Hudson.

Epilog Laser. 2009. *Jared Tarbell – Epilog Laser case study*. [Online]

Available from: http://www.epiloglaser.com/cs_tarbell.htm.

Fiell, C., 2003. *Graphic design for the 21st century, 100 of the world's best*. Köln: Taschen.

Flake, G. W., 1998. *The computational beauty of nature*. Cambridge, Massachusetts: The MIT Press.

Forman, C. et al., 2004. *Setpixel // Index* [Online]. Available at:

<http://www.setpixel.com>.

Fry, B., 2003. *Ben Fry*. [Online]. Available at:
<http://acg.media.mit.edu/people/fry>.

Fry, B. & Reas, C., 2009. *Processing online*. [Online]. Available at:
<http://processing.org/>.

Fuller, P., 1988. The search for a post modern aesthetic. In: J. Thakara, ed.
Design after modernism. London: Thames & Hudson. pp.117-134.

Galanter, P., 2005. *Generative.net - Definitions*. [Online] Available at:
<http://www.generative.net/read/definitions>.

Gere, C., 2002. *Digital culture*. London: Reaktion Books.

Gerhardt, J. & Jarman, R., 2007. The Mini-Epoch Series (Interview with
Semiconductor). In: B. Fry, & C. Reas. 2007. *Processing: A programming
handbook for visual designers and artists*. Cambridge, Massachusetts: The
MIT Press, pp.391-392.

Goriunova, O., 2007. *Digital artists handbook: Software art*. [Online]
Available at: <http://www.digitalartistshandbook.org/softwareart>.

Graham, P., 2003. *Hackers and painters*. [Online]. Available from:
<http://www.paulgraham.com/hp.html>.

Gray, C. & Malins, J., 1993. *Research procedures / methodology for artists
& designers*. Invited chapter in: 'Principles and Definitions: Five Papers by
the European Postgraduate Art & Design Group', European League of
Institutes of the Arts (ELIA). Winchester: Winchester School of Art.

Gray, C. & Malins, J., 2004. *Visualizing research: a guide to the research
process in art and design*. Aldershot: Ashgate Publishing.

Gray, C. & Pirie, I., 1995. Artistic' research procedure: Research at the edge of chaos?. In: *Proceedings of Design Interfaces Conference* Vol.3. The European Academy of Design. Salford: University of Salford.

Hamilton Grant, I., 1998. Post modernism and science and technology In: S. Sim, ed. *The icon dictionary of postmodern thought*. Cambridge: Icon books. pp.53-64.

Harris, J., 2000. *Surface tension - the aesthetic fabrication of digital textiles: The design and construction of 3D computer graphic animation*. Ph. D. London: Royal College of Art.

Heller S., 2002. *The Graphic Design Reader*. New York: Allworth Press.

Heller, S. & Finamore, M. eds., 1997. *Design Culture*. New York: Allworth Press.

Hodgin, R., 2009. *Robert Hodgin portfolio* [Online]. Available at: <http://roberthodgin.com/>.

Hodgin, R., 2010. *All manner of distractions* [Online]. Available at: <http://www.flight404.com/blog/>.

Hopper, R., 2004. Splines nurbs and boolean curves: The poetics of virtual form made flesh. In: *Challenging Craft International Conference*. Gray's School of Art, Aberdeen 8-10 September 2004. Available at: <http://www2.rgu.ac.uk/challengingcraft/ChallengingCraft/papers/richardhooper/rhooperabstract.htm>.

King, M. 1995. *Programmed graphics in computer art and animation*. [Online]. Available at: <http://web.ukonline.co.uk/mr.king/writings/technical/progart.html>.

King, E. & Küsters, C., 2001. *Restart: New Systems in Graphic Design*. London: Thames & Hudson.

Kirschenbaum, M. G., 2005. *The other end of print: David Carson, graphic design, and the aesthetics of media*. [Online]. Available from: <http://web.mit.edu/comm-forum/papers/kirsch.html>.

Krysa, J. & Sedek, G., 2006. *Source Code*. In: M. Fuller, ed. 2008. *Software studies: A lexicon*. Cambridge, Massachusetts: The MIT Press. pp.236-243.

Lehni, J. 2007. Hektor and Scriptographer (Interview with Jürg Lehni). In: B. Fry, & C. Reas. 2007. *Processing: A programming handbook for visual designers and artists*. Cambridge, Massachusetts: The MIT Press, pp.271-272.

Levin, G., 2000. About the Audiovisual Environment Suite *Communication Arts Magazine*, [Online]. May 2000, Competition Issue. Available at: http://www.flong.com/texts/essays/statement_commarts/.

Levin, G. Lia, Meta, & Ward, A., 2001. *Generative design: Beyond Photoshop*. Birmingham: Friends of ED.

Levin, G., 2006. *Interview by Dayna Crozier for Res Magazine June 2006*. [Online] Available at: http://www.flong.com/texts/interviews/interview_res.

Levin, G. & Lieberman, Z., 2007. *Messa di Voce* (Interview with Golan Levin and Zachary Lieberman). In: B. Fry, & C. Reas. 2007. *Processing: A programming handbook for visual designers and artists*. Cambridge, Massachusetts: The MIT Press, pp.511-512.

Levin, G., 2009. *Flong interactive art by Golan Levin and collaborators*. [Online]. Available at: <http://www.flong.com/>.

Lupton, E., 1997. A postmortem on deconstruction. In: S. Heller, & M. Finamore, eds. 1997. *Design Culture*. New York: Allworth Press, pp.113-115.

Lupton, E., 1999. *Design writing research*. London: Phaidon.

Lupton, E. & Cole Phillips, J., 2008. *Graphic design the new basics*. New York: Princeton Architectural Press.

Macdonald, N., 2002. Web Wizards: Designers Who Define The Web, *Eye*, [Online] 43 (11), Available at: <http://writing.spy.co.uk/Articles/Eye/WebWizards/>.

Maeda, J., 1995. *Essay for MDN Magazine March / April 1995* [Online] Available at: http://www.maedastudio.com/1995/mdn2/index.php?category=all&next=exists&prev=exists&this=reactive_graphics.

Maeda, J., 1999. *Design by numbers*. Cambridge, Massachusetts: The MIT Press.

Maeda, J., 2000. *Maeda@media*. London: Thames & Hudson.

Maeda, J., 2004. *Creative code*. London: Thames & Hudson.

Manovich, L., 2002a. *Generation flash*. [Online] Available at: http://manovich.net/DOCS/generation_flash.doc.

Manovich, L., 2002b. *The language of new media*. Cambridge, Massachusetts: The MIT Press.

Manovich, L., 2003. *Don't call it art*. [Online] Available at: http://manovich.net/DOCS/ars_03.doc.

Manovich, L., 2004. *Abstraction and complexity*. [Online] Available at: http://manovich.net/DOCS/abstraction_complexity.doc.

Manovich, L., 2006. *Import / export: Design workflow and contemporary aesthetics*. [Online] Available at: <http://manovich.net/DOCS/workflow.doc>.

Manovich, L., 2008. *Software takes command*. [Online] Available at: <http://softwarestudies.com/softbook>.

Marshall, J.J., 2008. *An exploration of hybrid art and design practice using computer-based design and fabrication tools*. Ph. D. Aberdeen: The Robert Gordon University.

Mazanti, L., 2004. Re-reading the functional: A new position for contemporary craft. In: *Challenging Craft International Conference*. Gray's School of Art, Aberdeen 8-10 September 2004. Available at: <http://www2.rgu.ac.uk/challengingcraft/ChallengingCraft/papers/louisemazanti/lmazantiabstract.htm>.

McCullough, M., 1998. *Abstracting craft: The practiced digital hand*. Cambridge, Massachusetts: The MIT Press.

McLean, A. & Wiggins, G. 2010. *Bricolage Programming in the Creative Arts*. [Online]. Available at: <http://yaxu.org/writing/ppig.pdf/>.

Meggs, P., 1998. *A history of graphic design*. 3rd edition. New York: John Wiley & Sons Inc.

Middendorp, J., 2000. *LettError type and typography: ToolSpace*. [Online] Available at: <http://lettererror.com/content/toolspace/index.html>.

Mignonneau, L. & Sommerer, C., 2003. *From the poesy of programming to research as an art form*. In: H. von Gerfried Stocker & C. Schopf, ed. 2003. *Code – The language of our time: Ars Electronica 2003*. Osterfildern-Ruit: Hatje Cantz, pp.242-249.

Morris, W., 1882. *Hopes and fears for art*. [Online] Available at: <http://www.marxists.org/archive/morris/works/1882/hopes/hopes.htm>.

Morris, W., 1884. *Some hints on pattern-designing*. [Online] Available at: <http://www.marxists.org/archive/morris/works/1881/hints.htm>.

Morris, W., 1893. *Textiles* [Online] Available at: <http://www.marxists.org/archive/morris/works/1893/textiles.htm>.

Nakamura, Y., 2004. The internet tree. In: J. Maeda. 2004. *Creative code*. London: Thames & Hudson, p.110.

Nakamura, Y. & Fitzpatrick, M., 2008. Yugo Nakamura: The craftsman (interview). *Creative Review*, February 2008 pp.46-48.

Natzke, E., 2007. *Natzke*. [Online]. Available at: <http://play.natzke.com/>.

Olding-Smee, A., 2002. *The new handmade graphics: beyond digital design*. Switzerland: Rotovision.

Paul, C., 2003. *Public Cultural Production Art (Software)* {. In: H. von Gerfried Stocker & C. Schopf, ed. 2003. *Code – The language of our time: Ars Electronica 2003*. Osterfildern-Ruit: Hatje Cantz, pp.129-135.

Paul, C., 2003b. *Digital art*. London: Thames & Hudson.

Pevsner, N., 1960. *Pioneers of modern design: from William Morris to Walter Gropius*. Harmondsworth: Penguin.

Poyner, R., 2003. *No more rules: Graphic design and postmodernism*. London: Laurence King.

Pye, D., 1978. *The nature and art of workmanship*. Cambridge : Cambridge University Press.

Raymond, E., 2000. *The Cathedral and the Bazaar* [Online] Available at: <http://catb.org/esr/writings/homesteading/cathedral-bazaar/cathedral-bazaar.ps>.

Reas, C., 2003. *Programming media*. In: H. von Gerfried Stocker & C. Schopf, ed. 2003. *Code – The language of our time: Ars Electronica 2003*. Osterfildern-Ruit: Hatje Cantz, pp.174-179.

Reas, C., 2004. The language of computers. In: J. Maeda. 2004. *Creative code*. London: Thames & Hudson, p.44.

Reinfurt, D., 2005. *Making do and getting by*. [Online] Available at: <http://www.adobe.com/designcenter/thinktank/makingdo/>.

Richardson, A., 2006. New media, new craft?. In: *Siggraph 2006: The 33rd International Conference and Exhibition on Computer Graphics and Interactive Techniques*. Boston, Massachusetts, USA 30 July - 3 August 2006. Boston, USA.

Riedelbauch, G., 2004. Craft and new technologies, implications for practice: A match made in heaven. In: *Challenging Craft International Conference*. Gray's School of Art, Aberdeen 8-10 September 2004. Available at: <http://www2.rgu.ac.uk/challengingcraft/ChallengingCraft/papers/griedelbauch/griedelbauchabstract.htm>.

Robbins, C., 2005. *Christopher Robbins PhpWiki - Dm Digital Materials* [Online] Available at: <http://www.grographics.com/wiki/index.php/DmDigitalMaterials>.

Ruskin, J., 1977. *The nature of the gothic. A chapter of the stones of Venice*. New York & London: Garland Publishing.

Shaw, E., 2007. *Re-locating ceramics: art, craft, design? A practice-based, critical exploration of ceramics which re-locates the discipline in the context of consumption, the home and the everyday*. Ph. D. London: University of Westminster.

Simon, J., 2004. Authorship, creativity and code. In: J. Maeda. 2004. *Creative code*. London: Thames & Hudson, p.46.

Small, D., 2009. *Small design firm*. [Online]. Available at: <http://www.davidsmall.com/>

Smith, B. K., 2006. Design and computational flexibility. *Digital Creativity*, 17 (2), pp.65-72.

Tarbell, J., 2005. *Levitated | the exploration of computation* [Online]. Available at: <http://levitated.net>.

Tarbell, J. 2007. Fractal. Invaders, Substrate (Interview with Jared Tarbell). In: B. Fry, & C. Reas. 2007. *Processing: A programming handbook for visual designers and artists*. Cambridge, Massachusetts: The MIT Press, pp.157-158.

Thackara, J. ed., 1988. Beyond the object in design. In: J. Thackara, *Design after modernism: Beyond the object*. London: Thames & Hudson. pp. 11-34.

Ulmer, G., 1990. The object of post-criticism. In: H. Foster. ed. *Postmodern culture*. London: Pluto Press. pp.83-110.

Van Rossum, J. & Van Blokland, E., 2000. *LettError*. [Online]. Available at: <http://lettererror.petr.com/content/toolspace/index.html>.

Van Rossum, J. & Van Blokland, E., 2003. *LettError type: Fonts and typography*. [Online]. Available at: <http://www.lettererror.com/index.html>.

Ward, A., 2002. *Software art*. [Online]. Available at: <http://www.adeward.com/go/Software+Art>.

Ward, A., 2003. *Signwave Auto-Illustrator 1.2*. [Online]. Available at: <http://swai.signwave.co.uk/>.

Watson, N., 1998. Postmodernism and lifestyles. In: S. Sim. 1998. *The Icon Dictionary of Postmodern Thought*. Cambridge: Icon Books, pp.53-64.

Wattenberg, M., 2004. The art of visualization. In: J. Maeda. 2004. *Creative code*. London: Thames & Hudson, p.78.

Wattenberg, M., 2007. Shape of Song (Interview with Martin Wattenberg). In: B. Fry, & C. Reas. 2007. *Processing: A programming handbook for visual designers and artists*. Cambridge, Massachusetts: The MIT Press, pp.161-162.

Watz, M., 2003. *Teaching – computational design and generative art*. [Online] Available at: <http://workshop.evolutionzone.com/old/>.

Watz, M., 2005. *Generator.x >> Computational design*. [Online] Available at: <http://www.generatorx.no/category/computational-design/>.

Watz, M., 2006. *10*10 - 10 questions for 10 Nordic artists*. [Online] Available at: <http://www.artificial.dk/articles/10x10marius.htm>.

Wild, L., 1997. Art and design, lovers or just good friends. In: S. Heller, & M. Finamore, eds. 1997. *Design Culture*. New York: Allworth Press, pp.92-95.

Womack, D., 2006. Tools to make or break. *Eye*, Summer 60 (15), pp.64-66.