

TOWARD SEMANTIC MODEL GENERATION
FROM SKETCH AND MULTI-TOUCH INTERACTIONS

A Dissertation
Presented to
The Academic Faculty

by

Chih-Pin Hsiao

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Architecture

Georgia Institute of Technology
December 2015

COPYRIGHT 2015 BY CHIH-PIN HSIAO

TOWARD SEMANTIC MODEL GENERATION
FROM SKETCH AND MULTI-TOUCH INTERACTIONS

Approved by:

Prof. Ellen Yi-Luen Do, Advisor
College of Architecture
Georgia Institute of Technology

Prof. Charles Eastman
College of Architecture
Georgia Institute of Technology

Prof. Baabak Ashuri
College of Architecture
Georgia Institute of Technology

Prof. Mark D. Gross
Department of Computer Science
University of Colorado Boulder

Prof. James Foley
College of Computing
Georgia Institute of Technology

Date Approved: September 4th 2015

To My Wife Yi-Chen Yu

ACKNOWLEDGMENT

First, I thank to my parents for supporting me in all of my educations. In the process of designing interactions for the prototype, I realized that I am not creating a tool, but an instrument for augmenting creativity in a way similar to playing musical instruments. Growing up, I learned to play many instruments, e.g. violin, piano, trumpet, etc. It is my parents who introduced me to the music world. The experiences of playing music instruments made me wonder whether we can create seamless interactions between human and artefacts, especially for supporting creativities in the architectural design process.

Special thanks to my advisor, Prof. Ellen Do. It is her who brought me into the Ph.D. program at Georgia Institute of Technology. Entering the Ph.D. program at Georgia Institute of Technology was my dream before deciding pursuing graduate degrees in the US. During the time in Georgia Tech, Prof. Ellen Do made sure that I was receiving adequate educations, trained me writing academic papers, and introduced many great professors in the School of Architecture and School of Interactive Computing to me, including Prof. Charles Eastman, Prof. James Foley, Prof. Craig Zimring, Prof. Baabak Ashuri, Prof. Thad Starner, Prof. Gregory Abowd, Prof. Ali Mazalek, Prof. Brian Magerko, and many other professors in the related area worldwide, especially Prof. Mark Gross. Each professor has profound impacts on how we should create interactions that supports our daily life, creativity, and most importantly, design activities. Some of these professors provide further supports by becoming my committee members. I am really thankful of having the best possible committee members in the world when pursuing this area of research.

Let me also say thank you to all the friends I met in ACME lab established by Prof. Ellen Do also, including Nicholas Davis, Hyungsin Kim, Andy Wu, Richard Lee, Samantha Wang, Matthew

Swarts, Adetania Pramanil, James Hallam, etc. Among them, I would like to thank Nicholas Davis especially. Nick is a gifted researcher in the area of cognitive science and creativity research. He provided many insights into how humans think creatively and suggestions on how to improve the interaction designs. More importantly, he shared a lot of interests with me and thus we were able to discuss infinite potentials of the researches in this area. Together, we developed theories, prototypes, user studies that fill up almost all my research lives in the last three years of my Ph.D. journey. My dissertation would not have happened without Nick being around.

Also, thanks to the members and professors in Design Machine Group (DMG) at University of Washington where I pursued my master degree. Prof. Brian Johnson guided and helped me establish the fundamental knowledge and skills for doing the research in this area. Daniel Belcher is one of the experienced colleagues at DMG that gave me many suggestions of how we can create technologies that human users can interact with them. Furthermore, Prof. Brian Johnson introduced Richard See to me as the supports from the industry. Richard is one of the pioneers in Building Information Modeling (BIM) industry. I learned a lot of BIM related knowledge from him during the time working with him. He created the opportunities for me to practice the programming skills learned in the school.

In addition, thanks to my mentors when I was interning at GE Global Research and HP Labs, including James Lin and Gregory Cook. You broadened my view of researches and created the chances of working with other researchers and developers in the industrial settings.

Of course, I could not finishing this dissertation without talking to my wife, Yi-Chen. I really appreciate the absolute love, helps, and supports from her, especially moving between many places for different interns, taking care of our kid, giving me the energy every time when I felt frustrated.

TABLE OF CONTENTS

ACKNOWLEDGMENT	iv
LIST OF TABLES	ix
LIST OF FIGURES	x
SUMMARY	xiii
Chapter I. Introduction	1
1.1 Solving Ill-Defined Problems	2
1.2 Goals of the Dissertation	3
1.3 Focus on Sketch and Multi-Touch Interactions	4
1.4 Structure of the Dissertation	6
Chapter II. CAD Tools and Design Cognition	7
2.1 CAD Programs	7
2.2 The Right Tools at the Right Time	8
2.3 Interoperability in Design	10
2.4 The Early Stage of Design	13
2.4.1 Sketch in the Early Stage of Design	13
2.4.2 Design Cognition	15
2.4.3 Vague Goals	17
2.5 CAD Programs for the Early Stage of Design	18
2.6 Conclusion	21
Chapter III. Related Work	23
3.1 Sketch-Based Interfaces for CAD	23
3.1.1 Sketch Recognition for Recognizing Drawing Intent	24
3.1.2 Simplifying Tasks in 3D Geometry Creation	25
3.2 Support of Rapid Creation for 3D Geometries	28
3.2.1 From Simple Sketch Lines to Complicated 3D Geometries	28
3.2.2 Supporting Idea Explorations in 3D	30
3.3 Automation in 3D Computer Graphics	36
3.4 Creating Constraints and Rules through Sketch Interactions	38
3.5 The Right Interactions	41
3.6 Conclusion	43
Chapter IV. Use Scenario of SolidSketch	46
4.1 Preliminary Design	47
4.2 Building Templates with Rules and Parameters	48
4.3 Extending Dimension Tool	50
4.4 Manipulating Existing Geometry with Multi-Touch Interactions	52
4.5 Array Creation Tool	53
4.6 Editing Through Stylus	54
4.7 Making Holes and Boolean Subtraction on Solid Shapes	55

4.8 Transit to Next Stage of Design	56
4.9 Summary	57
Chapter V. System Details	59
5.1 System Architecture	59
5.2 The General Controller Layer	62
5.3 Real-time Processing	63
5.4 Sketch Segmentations and Primitive Fitting	66
5.5 Symbolic Gesture Recognition	70
5.6 Checking the Contexts	72
Chapter VI. Data Mapping and Internal Data Structure	75
6.1 The Parametric Objects	75
6.2 Two-Dimensional Geometries	76
6.2.1 Polyline and Polygon Shape	76
6.2.2 Handling Sketch Editing	78
6.2.3 The Stripe Shape	81
6.3 Three-Dimensional Geometries	82
6.4 Constraints	85
6.5 Operations	86
6.5.1 List of Operations	86
6.5.2 Generate Feedback in Real Time	88
6.6 Mapping to Semantic Data	91
6.6.1 Shape Representation Concepts	91
6.6.2 Root Attributes	92
6.6.3 Generic Voiding	93
6.6.4 Other Concepts	93
6.6.5 Interoperability between SolidSketch and BIM Programs	96
6.7 Conclusions	98
Chapter VII. Case Study and Discussions	100
7.1 Comparison of Existing Programs	100
7.2 Design of the Case Study	105
7.3 Data Analysis from Recorded Videos	109
7.3.1 The Interaction Sequences Observed from Recorded Videos	112
7.3.2 Exploring Spatial Configurations	115
7.3.3 Studying the Forms of the Building	116
7.3.4 Creating Precise Designs	120
7.4 Semi-Structured Interviews	121
7.5 Results and Discussions	123
7.6 Summary of the Dissertation	124
Chapter VIII. Future Work	129
8.1 Improving Interactions	129
8.1.1 Improving Precision in Sketch When Necessary	130

8.1.2 3D Object Selection	131
8.1.3 Embedding More Semantics	132
8.1.4 Additional Ways to Build Template Modules	132
8.2 Incorporating More Computational Powers	133
8.2.1 Interoperability	133
8.2.2 Simulations	134
8.3 Conclusion	135
APPENDIX A Case Study Materials	137
1. Training and Tutorial Session	137
2. Post-Participation Questionnaire	137
3. Semi-Structured Interview Questions	139
APPENDIX B Case Study Results	141
1. Trending Models for Selected Data from Video Coding	141
1.1 Trending Model for "Sketch" Codes from Participant 1	141
1.2 Trending Model for "Sketch" Codes from Participant 2	141
1.3 Trending Model for "Sketch" Codes from Participant 3	141
1.4 Trending Model for "Sketch" Codes from Participant 4	141
2. Semi-Structured Interview Transcripts	142
2.1 Session 1	142
2.2 Session 2	143
2.3 Session 3	145
2.4 Session 4	147
References	150

LIST OF TABLES

Table 2.1: The comparisons between traditional sketch tools and CAD/BIM tools in different aspects discussed in this chapter	22
Table 2.2: The mouse clicks and movement gestures	26
Table 3.1: The summary of different sketch prototypes.	44
Table 6.1: The inputs, outputs, and weights for the operations used in the system.	89
Table 7.1: The interaction to achieve fundamental operations in SolidSketch.	100
Table 7.2: Comparison of features supported in different 3D sketch tools.	103
Table 7.3: The participants of the case study and their brief information	105
Table 7.4: The final code scheme used to record the participants' actions	109
Table 7.5: Six categories of activities used to categorize participants' actions.....	110
Table 7.6: SolidSketch in the criteria from Table 2.1	126

LIST OF FIGURES

Figure 2.1: The workflow of the current design process.	9
Figure 2.2: The Model View Definition for IFCWall in Concept Design BIM 2010	11
Figure 2.3: The iterative procedures of the design process.....	13
Figure 2.4: The interfaces of Rhinoceros (left) and Grasshopper (right) are very different	19
Figure 3.1: Electronic Cocktail Napkin.....	24
Figure 3.2: SILK	24
Figure 3.3: The process of finding skeleton spline in Teddy	29
Figure 3.4: The process of elevating spline and construct polygonal meshes in Teddy	29
Figure 3.5: Left: After sketching a simple blob, the system will automatically generate the structured annotations. Right: The results of interactions from on the right	29
Figure 3.6: TreeSketch.....	30
Figure 3.7: Yang et al. proposed a method to fit the sketch lines into predefined template	31
Figure 3.8: The process of using the perspective sketch tools Tolba proposed	32
Figure 3.9: The Mental Canvas	32
Figure 3.10: The system analyzes the positions and connections of the sketched lines, and optimizes the 3D projections of sketch lines to determine the mesh geometries	33
Figure 3.11: Analytic drawing tool	34
Figure 3.12: Left: The type of intersections Chen’s prototype is using. Right: The user needs to sketch the perspective views carefully for the system to reconstruct the 3D positions	35
Figure 3.13: True2Form	35
Figure 3.14: Li’s system automatically analyzes the topology of different parts of a model	36
Figure 3.15: After analyzing topological structure of the building façade (left), the system knows which parts can be scratched and squeezed (right).....	36
Figure 3.16: After analyzing the pattern on the façade, users can stretch the model in different directions	37
Figure 3.17: Through analysis of the geometry, the system acquires the topology and computes the probability density of the shapes	38
Figure 3.18: ParSketch.....	39
Figure 3.19: Lineogrammer	40
Figure 3.20: Vignette	41
Figure 3.21: Kitty	41
Figure 3.22: The combination of multi-touch and pen interactions creates a rich interaction environment	42
Figure 3.23: Eden (left) and Kang’s (right) prototypes	43
Figure 4.1: Peter uses SolidSketch to sketch out his initial design ideas and build rough models that contain building information	46
Figure 4.2: Stylus gestures (left) and multi-touch interactions (right).....	47
Figure 4.3: The axis widget	48
Figure 4.4: Peter can sketch out his initial ideas quickly in freehand sketch mode	48

Figure 4.5: The steps to creating a simple wall template with two operations.....	49
Figure 4.6: Left: After hitting the Plus button again, Peter can draw the icon and assign the type of template object he is creating. Right: The system will store the rules and parameters and create an icon on the left panel of the UI	50
Figure 4.7: The system generates a wall from template in real time	50
Figure 4.8: Peter sketches a new line at the end of the profile of a wall	51
Figure 4.9: Peter extends a one-dimensional line to a 2D surface through a sketch interaction ..	51
Figure 4.10: Extending a 2D shape to a 3D solid model.....	52
Figure 4.11: Basic multi-touch gesture in SolidSketch	52
Figure 4.12: Creating an array from a shape	53
Figure 4.13: Peter adjusts the number of elements in an existing array	54
Figure 4.14: By drawing a curved line near to the slab profile, Peter can quickly create a balcony and experiment with its size and shape	55
Figure 4.15: By drawing a closed polygon inside of the slab profile, Peter can create a hole for stair access.....	55
Figure 4.16: Peter can employ a scratch gesture to accomplish Boolean subtraction while selecting a base geometry	56
Figure 4.17: The simple house structure that Peter is able to create in 30 minutes.....	57
Figure 5.1: The overall system architecture for SolidSketch.....	60
Figure 5.2: The overall sketch/multi-touch analysis pipeline for detecting intentions to execute commands or related actions	61
Figure 5.3: The detail sketch recognition process for SolidSketch	62
Figure 5.4: The logic flow (top) and the pseudocode (bottom) for dynamic recognition	64
Figure 5.5: Preset scratch gesture samples.....	65
Figure 5.6: The line fitting algorithm is not able to fit the stroke into one line segment on the left, whereas it has high confidence of fitting the stroke into one line segment on the right	67
Figure 5.7: The results of sketch segmentation and primitive fitting. Blue segments are straight line segments, and orange segments are arc segments	68
Figure 5.8: The pseudocode for identifying candidate corners and primitive fitting	69
Figure 5.9: The pseudocode for recognizing sketch gestures	70
Figure 5.10: The example sketch strokes, the segmentation results, and the rules for each gesture	71
Figure 5.11: The interaction and contextual information used in the procedure of checking the contexts	73
Figure 5.12: Performing a right-angle gesture on an edge behind a solid shape	74
Figure 6.1: The general parametric object and the major parametric object types used in SolidSketch.....	76
Figure 6.2: The UML diagram for major 2D objects used in SolidSketch.....	77
Figure 6.3: Polyline versus polygon. The system creates meshes to fill in the polygon	77
Figure 6.4: The sketch editing algorithm taking the larger shape of the two candidate shapes...78	
Figure 6.5: The pseudocode for merging a new stroke into existing lines	81
Figure 6.6: The offset algorithm consists of four steps.....	82

Figure 6.7: The UML diagram describing Boolean Subtraction and swept solid parametric objects used in the system	83
Figure 6.8: Two types of swept solids in SolidSketch	84
Figure 6.9: Simplified relationships created through the interactions between various parametric objects for a solid shape	85
Figure 6.10: The actual parametric structure of the chair shown on the right	86
Figure 6.11: The operations (left) that compose the wall template (right).	87
Figure 6.12: The operations (left) for a complicated template that consists of openings and columns on a wall	88
Figure 6.13: The execution order for the operations shown in Figure 6.12-left by using a naïve topological sorting approach	89
Figure 6.14: The Directed Acyclic Graph from the operations in Figure 6.12 (left) and its ideal execution order	90
Figure 6.15: The pseudocode for computing the order of executing the operations	91
Figure 6.16: The Model View Definition Diagram for a wall created by SolidSketch	94
Figure 6.17: The Generic Voiding concept from CDB 2010.....	95
Figure 6.18: The Generic Containment concept from CDB 2010.....	95
Figure 6.19: The data produced in SolidSketch can be imported into other BIM programs	97
Figure 6.20: The building model created in SolidSketch can be edited in Revit for adjusting precision and adding more details	98
Figure 7.1: The selected site for the task	106
Figure 7.2: The view that records the participant’s actions during the experiment	107
Figure 7.3: The total and average duration in the video recordings.....	111
Figure 7.4: An interaction sequence starts and ends with navigation activities	113
Figure 7.5: (1) Robert created an interior wall in the middle of the canvas. (2) He learned that he can use an existing wall for the wall he just created. (3) He deleted the wall he just drew. (4) He zoomed in and copied the wall. (5) He moved to the desired location and drew another interior wall. (6) He used the wall for the other three rooms.....	114
Figure 7.6: Participants 1 to 3 start the design task by quickly sketching out the floor plans	115
Figure 7.7: The solid red lines represent the actual running total of duration (in seconds) while the participants use freehand sketch to study spatial configurations during the experiment sessions	116
Figure 7.8: Participant 1 (Tommy) studied the floor plan again after he had done some freehand sketches at 25 minutes into his study session.....	117
Figure 7.9: The running sum of the duration throughout the entire study	118
Figure 7.10: The count of using the scratching gesture to delete the geometries and undo/redo to reverse the recent creation of geometries	119
Figure 7.11: The orthogonal shape Tommy created at the beginning of the study session	121
Figure 7.12: Designs of simple houses using SolidSketch from the four participants	124
Figure 8.1: After selecting a two-dimensional template, the user can quickly sketch a closed polygon (left) to generate the template floor plan and building (right)	133

SUMMARY

Designers usually start their design process by exploring and evolving their ideas rapidly through sketching since this helps them to make numerous attempts at creating, practicing, simulating, and representing ideas. Creativity inherent in solving the ill-defined problems (Eastman, 1969) often emerges when designers explore potential solutions while sketching in the design process (Schön, 1992). When using computer programs such as CAD or Building Information Modeling (BIM) tools, designers often preplan the tasks prior to executing commands instead of engaging in the process of designing. Researchers argue that these programs force designers to focus on how to use a tool (i.e. how to execute series of commands) rather than how to explore a design, and thus hinder creativity in the early stages of the design process (Goel, 1995; Dorta, 2007). Since recent design and documentation works have been computer-generated using BIM software, transitions between ideas in sketches and those in digital CAD systems have become necessary. By employing sketch interactions, we argue that a computer system can provide a rapid, flexible, and iterative method to create 3D models with sufficient data for facilitating smooth transitions between designers' early sketches and BIM programs.

This dissertation begins by describing the modern design workflows and discussing the necessary data to be exchanged in the early stage of design. It then briefly introduces the modern cognitive theories, including embodiment (Varela, Rosch, & Thompson, 1992), situated action (Suchman, 1986), and distributed cognition (Hutchins, 1995). It continues by identifying problems in current CAD programs used in the early stage of the design process, using these theories as lenses. After reviewing modern attempts, including sketch tools and design automation tools, we describe the design and implementation of a sketch and multi-touch program, SolidSketch, to facilitate and augment our abilities to work on ill-defined problems in

the early stage of design. SolidSketch is a parametric modeling program that enables users to construct 3D parametric models rapidly through sketch and multi-touch interactions. It combines the benefits of traditional design tools, such as physical models and pencil sketches (i.e. rapid, low-cost, and flexible methods), with the computational power offered by digital modeling tools, such as CAD. To close the gap between modern BIM and traditional sketch tools, the models created with SolidSketch can be read by other BIM programs. We then evaluate the programs with comparisons to the commercial CAD programs and other sketch programs. We also report a case study in which participants used the system for their design explorations. Finally, we conclude with the potential impacts of this new technology and the next steps for ultimately bringing greater computational power to the early stages of design.

Chapter I. Introduction

Design is a challenging cognitive activity, involving planning, remembering, experiencing, creating, defining, and solving “wicked problems” (Rittel & Webber, 1973). Design problems are considered “wicked” because they are usually ill-defined (Eastman, 1969; Shneiderman, 2007). The design process often includes numerous attempts at creating, practicing, simulating, and representing ideas using a variety of media and tools. The creativity inherent in solving these ill-defined problems often emerges when designers explore potential solutions while interacting with the media engaged in the design process (Schön, 1992; Sawyer, 2011). However, when creating computer programs, interface designers and developers often assume that users already have preconceived ideas of the tasks they will execute. Many creativity researchers, however, argue that designers employ sketching as a tool for design exploration since it can represent different semantics and thus afford designers the expression of ambiguous intentions (Goel, 1995; Gross & Do, 1996). In addition, designers can gradually refine and form clear ideas through sketching. Since recent design and documentation work has been computer-generated using Computer Aided Design (CAD) or Building Information Modeling (BIM) software, transitions between ideas in sketches and those in digital CAD systems become necessary. These computational tools often require complicated commands for expressing the rules, constraints, and parametric and topological information of geometries. Thus, when designers must focus on exploring the design possibilities and solving design problems, they often find it difficult to utilize these tools. Researchers in the design community argue that commercial computational tools, such as AutoCAD, force designers to focus on how to use a tool (how to execute series of commands) rather than how to explore a design, and thus hinder creativity in the early stages of the design process (Dorta, 2007).

1.1 Solving Ill-Defined Problems

How can creativity support tools (CSTs) help users to address ill-defined problems in the early stages of design? Researchers in the design and creative technology community have begun exploring cognitive theories to explain design activities and gain insights through observations of these theories, such as embodiment (Varela et al., 1992; Hornecker, 2005), situated action (Le Dantec, 2009; Schön, 1992; William J, n.d.), and distributed cognition (Hsiao, Davis, & Do, 2012; Hutchins, 1995). By emphasizing how users interact with their environment to continually construct meaning, these theories provide a foundation for designing creativity support tools for ill-defined problems. In particular, instead of explicitly planned goals and tasks, embodiment describes how cognition emerges through active interaction, coordination, and negotiation between a person and the surroundings for exploring alternatives (Hollan, Hutchins, & Kirsh, 2000). Sense-making is a gradual process whereby a person works to understand how to coordinate actions with his or her surroundings to produce outcomes that make sense. It thrives on creativity in the moment to enable humans to form “good enough” models and hypotheses for how things in the world work. Such hypotheses are evaluated and refined through real-time interactions between a human and his or her surroundings, which is described as “perception-action loops” in enaction theory (Hanne De Jaegher, 2010). During the design process, sense making is the process in which a designer tries to gradually fit his creations into predefined and emerging constraints. Design researchers argue that designers typically engage in these perception-action loops during the design process, called “conversations” with the materials of a design situation (Schön, 1992), since they need to “negotiate, discuss, and even exchange” ideas with the physical world without other cognitive burdens. We posit that if we wish to design interactions that provide the same experiences for solving ill-defined problems as traditional design tools, such as sketching, we should account for the embodied thinking process that

enables users to rapidly express and explore their ideas through interactions with the media. This objective raises the first question of this research:

How can we combine the direct input and real-time feedback in sketch interaction with the computational power of CAD/BIM in a single tool?

1.2 Goals of the Dissertation

Some may argue that computer programs mainly assist designers with generating complicated forms, simulating reality, and managing large projects that cannot be accomplished without them. By using computer programs, designers can also store expert knowledge they have developed over time and then reuse it when needed through generative design systems, case-based reasoning, expert systems, shape grammars, and other aspects of the process. From this perspective, computer programs accomplish what traditional media and tools cannot. In other words, some may assert that computational and traditional tools serve different purposes in the design process. Instead of introducing a novel method of facilitating computational powers for various design possibilities, the goal of this work, therefore, is to advance the user experience of employing computational power in the early stages of design.

In addition, this dissertation advocates for a new approach that both enables designers to directly interact with digital geometry and determines the relationships among the geometries. It aims to ameliorate the gaps in technologies intended to transition designers between design activities in digital and physical environments. In other words, it aims to resolve the issues existing in the current process in the early stages of design: designers often need to switch between physical and digital tools to verify whether their design intents violate any constraints established in different work environments. Despite the many attempts to employ sketch interactions for supporting early design explorations on computers (Bae, Balakrishnan, & Singh, 2008; Schmidt, Khan, Singh, &

Kurtenbach, 2009), these tools lack the ability to create parametric and semantic descriptions that define geometries and relationships between them used in CAD programs, such as SolidWorks, CATIA, Rhino 3D, etc. Thus, we argue that one important piece to bridge this gap is to include ways of recognizing and assigning constraints to geometries so that early design intentions can be created and passed on to subsequent stages of the design process. One of the goals of this dissertation is to translate human interactions to computer-understandable format that can then be utilized in subsequent tasks of the design process. This goal raises the second research question:

How does the combination of sketch and CAD/BIM tools affect the early stages of design? Can immediate feedback and continuous geometry generation by using sketch and multi-touch interactions support rapid, flexible, and iterative design ideations similar to using traditional sketching?

1.3 Focus on Sketch and Multi-Touch Interactions

Drawing from images in their minds, designers often represent designs with lines and circles that form bubble diagrams that help them to structure spatial relationships. The lines could represent walls, regions, and spaces to which certain properties are either explicitly or implicitly assigned. Properties could be the qualities and quantities of materials for a space or materials for a wall. To translate this information into a computer program, a designer may need to execute a “create wall” command by clicking a button and then using the mouse to draw walls on the canvas of the program. After drawing the walls, the designer may execute a “create space” command, and after drawing these basic elements, she may refer to the equations or values¹ in a table to define the properties of the elements. If walls are curved, designers may add control points and adjust the handles for desired

¹ The equations and values for properties are part of “constraints” in a design (Gross, 1986).

curvatures. This scenario, a common one in the daily routine of designers when they use a computer program, demonstrates that computer programs are tools that require many steps to create an object with details. However, after a number of computations, designers in the later stages of the process may discover errors that must be fixed. Such changes often require a return to the initial stage of the design process and further efforts to build models.

The previous paragraph presents an example of designers who often use their hands to sketch and manipulate their designs. We argue that sketch and multi-touch interactions can become signals for a system to create parametric information and geometry with relationships. Thus, in this dissertation, we narrow the investigations to sketch and multi-touch interactions. To maintain perception-action loops and reduce cognitive burden in the early stages of design, we propose that interactions should take place directly on geometries, including manipulating shapes and setting geometric relationships. Such an approach raises several technical questions, the most obvious being *what techniques could be used to disambiguate sketch and multi-touch input in 3D environment when users are creating geometrical and topological information*. Another is determining *how a computer program should react when designers draw lines and execute commands; that is, what specific feedback should a computer program provide a designer?* In addition, we prefer that the system take over some jobs and automate certain recognition processes to simplify particular tasks, but not others. The human should have some degree of control because the system will sometimes misunderstand a human's intention. Thus, it should be asked: *What are the strategies for conveying the interpretations from a computer?* This issue raises a final question of this dissertation: *To what extent can a system record parametric and semantic information from such sketch interaction techniques?*

1.4 Structure of the Dissertation

This chapter defines the scope and motivations of the dissertation by briefly discussing the drawbacks of current computational tools in the early design stage, and then introduces the theoretical background that raises the several questions that need to be answered in this dissertation. The next chapter, Chapter 2, reveals more details of the theoretical background, including a review of modern cognitive theories and their implications, with discussions of the workflows used in the current design environments. Chapter 3 mainly discusses the previous works that employed sketch interactions for enhancing our design capability and automating complicated tasks, but still failed to support design workflows. Through a use scenario with a hypothetical user, Chapter 4 introduces the interaction designs of a sketch and multi-touch prototype system and examines how constraints should be set using sketch gestures, and what feedback the system should provide. Chapter 5 provides the overall architecture, detailing algorithms that facilitate sketch and multi-touch interactions. Chapter 6 continues the presentation of technical details, with the emphasis on data structures and how the interactions are translated into building objects that can be exported to BIM tools. Chapter 7 discusses the means and results of evaluating the proposed prototype system based on the main objective of the system: providing a way for designers to explore ideas of designs rapidly without the burden of translating the data into the BIM-enabled formats. More importantly, this chapter will discuss the findings from a case study with expert users. Chapter 8 concludes the dissertation with potential benefits to architectural designers who use the tool, SolidSketch, in their design process and the near-term future work required to make this happen.

Chapter II. CAD Tools and Design Cognition

2.1 CAD Programs

CAD programs have been used to create models of designs for more than 50 years. One of the first CAD programs, SketchPad (Sutherland, 1963), has many useful features that provide computational power to assist with drawing, including storing design templates, instantiating predefined drawing objects, setting various constraints among geometries, and solving constraints from input parameters. These features influence the fundamental concepts inherent in modern CAD programs, such as parametric modeling, generative design, case-based design, and expert systems. These fundamental concepts enable developers to build computer-aided architectural design (CAAD) systems with embedded design knowledge. For example, CAAD tools help designers to apply their design expertise and record design intents that assist those in the AEC field to increase their productivity, avoid errors, and accomplish projects that could not be done without them. Furthermore, designers use generative systems and parametric modeling to automate complex design creations governed by rules and parameters that describe their design intents.

In architectural design, generative design systems are among the most popular methods of representing design expertise in practice. The best-known generative design system is Grasshopper for Rhinoceros. When using this system, architects can specify simple parametric rules to create forms that fulfill specific needs. At the architectural firm, NBBJ, Nathan Miller adopted Grasshopper to study design alternatives quickly by adjusting parameters (Miller, 2009). He built a set of Grasshopper rules for preventing columns in a stadium from blocking the views of individuals in the audience. If members of a design team wish to adjust the number of seats in a stadium or change the form of the building, they can modify the design and regenerate new models automatically to fulfill such requests while still adhering to Miller's set of rules about columns. Another example of

Grasshopper is developing a set of rules for automatically generating a parking structure. Many such rules are currently available to download online,² providing design templates to designers so that they can adjust the rules and parameters to fulfill their needs.

These parametric rules with topologies that define the relationships between geometries are the building blocks of Building Information Modeling (BIM) tools. BIM programs, such as Revit, have predefined relationships among geometric objects so that the system can automatically propagate modifications when a designer changes his designs. These geometric relationships are critical aspects of BIM model specifications that determine what kinds of rules can be applied between parts (Eastman et al., 2008). For instance, if a user moves a wall connecting to another wall, BIM programs know how to programmatically create the connections. When users add a window to a wall, BIM programs know where and how to create the frames of the window.

2.2 The Right Tools at the Right Time

Interoperability between BIM programs is crucial in architecture, engineering, and construction (AEC) fields. Since different AEC professionals use diverse programs and tools for a range of purposes in various stages of design, the interoperability between these programs becomes crucial. For instance, Figure 2.1 outlines a workflow depicting how architects might utilize different programs in the various stages of designing a building. In Figure 2.1, at the beginning of the design process, designers may collect data to create a building program with Excel spreadsheets that contain building specifications, including qualitative information and quantitative information. They would use these spreadsheets for communicating with clients to ensure that the design fulfills their needs. Designers may also use the spreadsheet to document specific building codes to determine whether the building

² <http://www.grasshopper3d.com/>

adheres to government standards and regulations. The data become the initial constraints of the design.

Once the designers have established the initial design constraints of the building, they start to sketch out designs and play with different ideas that can roughly fulfill these constraints. While exploring, they develop additional constraints that shape the design of the building. Then, designers start to sketch the forms in perspective drawings or build low-fidelity 3D models using programs such as SketchUp. This type of early 3D digital model gives designers a realistic rendering and serves as a tool to help facilitate more detailed communication with clients. During this process, designers also specify materials and other information, such as views of the particular room and lighting, related to the quality of a design in a digital model. This information would then be entered into the initial building information model, called the Design BIM (shown in the middle of Figure 2.1).

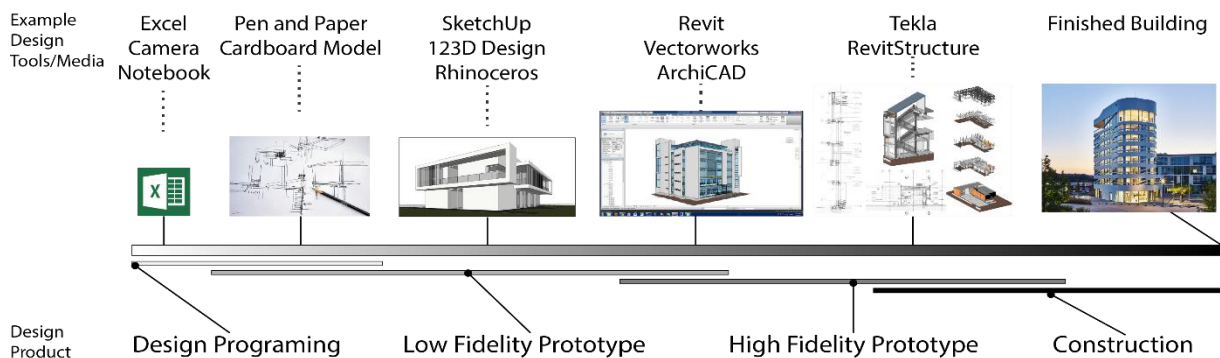


Figure 2.1: The workflow of the current design process. Architectural designers need to use different tools along the different process. In the middle of design process, they would start to use BIM tool for documenting their designs comprehensively. But they are still use drawing in the early stages of design.

Once designers begin to use Design BIM tools, interoperability is provided by the software programs to exchange design information between a variety of programs for different purposes, e.g. construction drafting, detail simulations, construction management, etc. This new generation of CAD tools helps professionals in the AEC field to optimize their workflows and facilitates communications with other professionals. Thus, it dramatically reduces costs at later stages in the design process (Eastman, Teicholz, Sacks, & Liston, 2008).

2.3 Interoperability in Design

Since designers need to use various tools and software programs to complete a building project, they rely on exchanging information between these tools. If developers of different software programs can define object structures rigorously in their systems, they can prevent numerous problems and frustrations when users import and export data between programs. Therefore, a neutral building information format such as Industry Foundation Class (IFC) plays a significant role in this domain. However, because IFC includes a large number of entities related to the life cycle of a building, those software developers in the AEC field would find it difficult to select information for the exchanges. The Model View Definitions (MVDs) aim to define specific schemas and create object structures with their necessary properties for different use cases (Eastman, Jeong, Sacks, & Kaner, 2009). These MVD structures are publicly available on the MVD website.³

In this dissertation, we focus on creating a tool to support the early stages of design roughly in the range of creating the “Low-Fidelity Prototype” in Figure 2.1. Currently, the closest MVD for the stage is “Concept Design BIM 2010 (CDB10)” (See, 2009). Figure 2.2 shows the overall information required for an IFCWall in CDB10. The light orange boxes in Figure 2.2 represent concepts for particular categories of semantics in an IFCWall. A concept can be constructed by multiple concepts to complete the required semantics. Some concepts are inherited from their parent IFC objects, and can also pass attributes to their children IFC objects. For instance, IFCWall is inherited from IFCRoot and thus has the attributes originating from IFCRoot, which are called Root Attributes. These attributes contain all of the basic information required for all of the objects in a building information model, including “GUID”, “BIM Object Owner History”, “Name”, and “Description”. The purpose of

³ <http://www.blis-project.org/IAI-MVD/> and <http://www.buildingsmart-tech.org/specifications/ifc-view-definition>

the BIM Object Owner History is to record the editing history for the particular IFC object. In other words, editing history in IFC is designed at the object level instead of at the file level to enable systems to track design intents in every single IFC object.

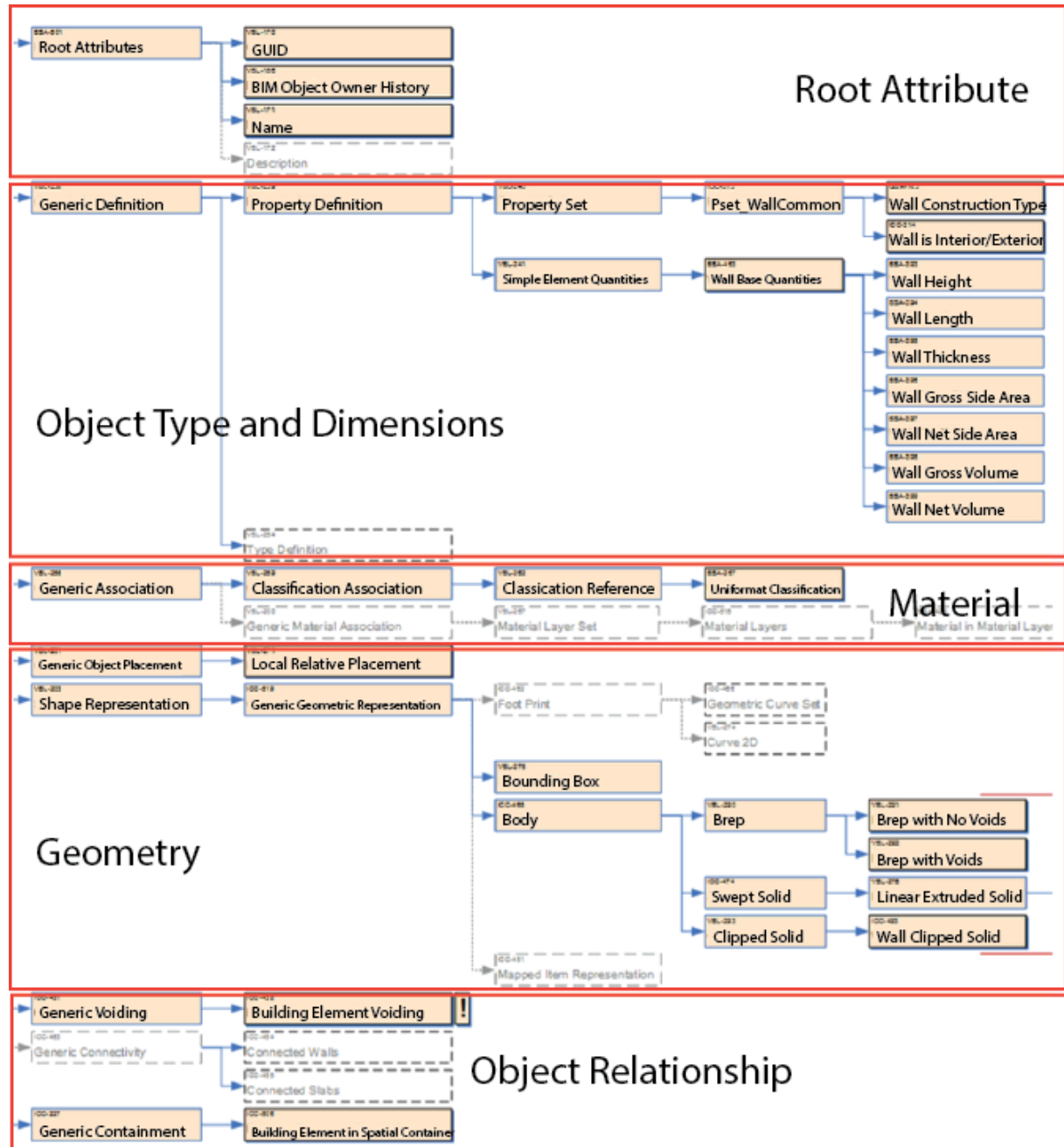


Figure 2.2: The Model View Definition for IFCWall in Concept Design BIM 2010. There are five categories of concepts for an IFCWall in Concept Design BIM 2010. The boxes in this diagram represent concepts. The boxes with solid outlines represent required concepts, while those with dashed outlines represent optional concepts.

In addition to the root attributes introduced previously, there are four other categories of concepts in building elements in CDB10: (1) object type and dimensions, (2) materials, (3) geometry, and (4) object relationship. In the object type and dimensions category, CDB10 aims to specify information for the type of analysis used in early stages of design. For instance, it stores the type, e.g. concrete wall, and the gross volume of an IFC object so that a simple plugin can read it for calculating whether the costs of the building roughly meet the building program mentioned in Section 2.2. Materials are another important layer of semantics for the purposes of energy simulations and quantity take-off. The materials can be specified in each layer of a building element using this concept. However, at this stage of design, the designers would only have rough ideas of the materials they will use in particular building elements. Thus, the related concepts are not required in the data exchange and are marked as optional in CDB10.

The concepts in the geometry category are crucial in CDB10 since various information can be derived from them. The concept “Generic Geometric Representation” contains the different options for representing geometrical information that can be used for exchange in early stages of design. Developers can choose one of the following concepts when exporting the geometrical information: (1) “B-rep”, (2) “SweptSolid”, and (3) “Clipped Solid”. For geometrical relationships, the concept “Local Relative Placement” suggests hierarchical placements between geometries.

Finally, the concepts in the “object relationship” category describe various relations in a building model. For instance, in IFCWall, the concept “Building Element in Spatial Containments” describes in which story a particular wall is situated within a building. The concept “Building Element Voiding” defines the openings that make the “holes” in a wall. Also, the concepts “Connected Walls” and “Connected Slabs” specify which walls/slabs this IFCWall connects to, and what type of connections exist between the two objects.

This MVD addresses interoperability, specifically at the moment that users finish a concept design, and then supports data exchange for the purpose of simulations or simple analyses. Other professions in the AEC field involved in this stage can utilize this MVD to ensure that the data they need are available from architects. In sum, this MVD represents a framework that allows us to place more emphasis on a particular data exchange in the early stages of design.

2.4 The Early Stages of Design

2.4.1 Sketch in the Early Stages of Design

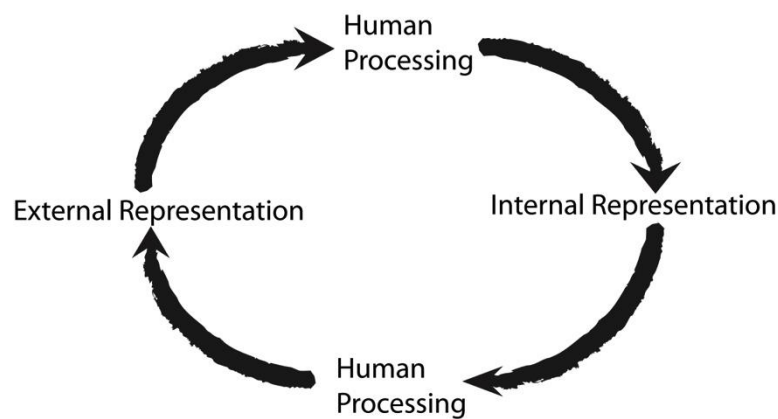


Figure 2.3: The iterative procedures of the design process

At the very beginning of the design process, designers define problems, requirements, and design principles that conform to the building programs that clients provide as well as the contextual information supplied by the environment on a proposed site. Based on this information and previous experience, designers draw various ideas and images from their minds. In this process, designers typically produce a rough sketch as an external representation of a general layout that might solve or redefine those ill-defined problems (Eastman, 1969). As they fill in the sketches with more details (representing design), they may discover, upon scrutiny, that some parts of the design could be improved, or that recent additions to the design have created problems. As a result, the designers may have to revise the drawings. During this iterative process, shown in Figure 2.3, designers must

not only adhere to the requirements, design principles, and physical representations, but they must also respond to emergent ideas while re-evaluating their previous representations.

The process of understanding abstract concepts of the design is not a straightforward input and output activity as suggested by information processing theories (Card et al., 1986). Rather, it is associated with long- and short-term memory as well as internal representations that convey images and concepts in our minds (Medyckyj-Scott & Hearnshaw, 1993), and external representations such as sketches or models. As designers attempt to interpret external representations, they receive various types of sensory inputs, such as tactile, auditory, and visual sensations. Upon receiving the information, they store it in working memory (short-term memory), which is meant for active processing. However, since working memory is limited, designers must represent the mental images for each input repeatedly (Gül, 2009) for concept reconstruction (Nersessian, 2008), and store them as physical artifacts, each of which should have unique purposes. Thus, Noë argues that it is often easier for someone to act on the environment and experiment with how different interactions affect the media (Noë, 2004). For instance, a 2D floor plan sketch should help designers to organize and represent the layouts of spaces, the intended circulation paths among them, and patterns of the architectural elements. A drawing in perspective represents views similar to the object being conveyed and thus accomplishes what a floor plan cannot. Designers switch between different views and examine their emerging ideas on demand. In this view, intentions emerge but are also transformed in and through interaction with the materials.

According to their experiments with architects and advanced students (Suwa & Tversky, 1997), Suwa and Tversky found that architects usually employ sketch for the following purposes: (1) finding emergent properties of spaces, objects, shapes, and sizes; (2) defining local and global spatial relations; (3) defining functional relations including views, lights, and circulations of people; and (4) developing conceptual knowledge. They found that expert architects focus more on complex visual

relations and functional relations at the same time, such as shapes, sizes, angles, and circulations, than architectural students do. In other words, experts are able to explore visual and functional relations simultaneously when employing sketches to study their designs.

Recent design research suggests that freehand work might not be the only way for designers to explore, develop, and form initial ideas in the early stages of design (Bilda, Gero, & Purcell, 2006). Johnson suggested that verbalizations are another major conceptual tool for initiating the design process (Johnson, 2005). This seems to conform to Schön's famous phrase, "*Designing as reflective conversation with the materials of a design situation*", which beautifully describes how designers form their ideas in the early stages of the design process by utilizing the media they are comfortable with (Schön, 1992).

2.4.2 Design Cognition

Based on empirical work investigating design meetings, Le Dantec argues that these "conversation" processes are influenced by three facets of inseparable cognitive processes: embodied, distributed, and situated cognitions (Le Dantec, 2009). Embodiment shows the importance of environmental affordances (Gibson, 1979; Varela, Thompson, & Rosch, 1999), which have been championed by HCI researchers as an effective interaction design principle for years (McGrenere & Ho, 2000; D. Norman, 1990). Affordances have recently taken on a more prominent role as novel input technologies, such as tangible and natural user interfaces, emphasizing direct manipulation and continuous feedback with actions (Dourish, 2004; Ullmer & Ishii, 2000). In fact, such embodied interactions commonly occur in architectural design processes when designers use sketches and gestures to express the quality of designs externally (Le Dantec, 2009), implying that embodied cognition involves the coordination of actions and thoughts without the designer's attention (Bloch, 1998).

Distributed cognition showed the importance of offloading tasks onto environments and other people working on the same tasks (Hutchins, 1995). In his book, *Cognition in the Wild*, Hutchins reported his findings of how the crew members on a battleship plan their navigation through drawing symbols and routes on a map as well as communicating the orders throughout different levels of crew members (Hutchins, 1995). In HCI, this perspective changed the primary unit of analysis from one of human processes information to socio-technical systems with distributed cognitive properties (Hollan, Hutchins, & Kirsh, 2000; Liu, Nersessian, & Stasko, 2008). In the architectural design environment, architects often utilize bubble diagrams, symbols, and other professional drawings to represent, explore, and communicate ideas (Do, 1998). Distributed cognitive analysis considers how these representations are propagated and processed by people, computers, artifacts, and environments through time. In the scenario of architectural design, generating an entire plan with its details directly from one's mind is virtually impossible. Instead, the design process requires many iterations and refinements beginning from one designer and eventually including multiple designers. In addition, since many creative discoveries involve emergent spatial and functional relationships in a sketch, the cognitive process of design is not linear (Suwa & Tversky, 1997). Even though the process may appear to take place inside the minds of architects, it is often distributed among their mental processes and the continuous feedback provided by the "materials of a design situation" (Schön, 1992).

Finally, situated action showed that the information processing view of fixed goals guiding actions was not entirely accurate. Suchman found that humans do not execute fully formed plans in their interaction with technology, but rather perceive affordances that guide actions that are adjusted based on real-time and continuous feedback (Suchman, 1986). When systems are designed with an information processing model of cognition, there are often breakdowns between the situated actions that users take and the linear model determining interaction design. For this reason,

Suchman has advocated using ethnographic methods to explore the micro-details of everyday actions and their explorative processes. These ethnographic methods are often used in qualitative studies with inductive reasoning during observations in the recorded video or audio data *in situ*. These analyses can reveal how actions are non-linear and largely dependent on feedback from the environment as well as sociocultural contexts in which the actions are embedded. These methods are widely used in studying design processes introduced previously, e.g. (Lakoff & Johnson, 2003; Le Dantec, 2009; Tversky, 2002).

2.4.3 Vague Goals

The previous section discussed the way designers think. Instead of describing design as a linear process that occurs in a step-by-step manner, researchers argue that design cognition is an iterative thinking procedure in which designers have vague goals that can be revised during interactions (Sawyer, 2000; Tversky, 2002). In addition, a vague goal acts as a set of constraints that directly influences what appears to be interesting or salient, as well as how specific types of interactions might provide more information about emerging hypotheses (Andreas, 2010). A vague goal is similar to an objective in that it can be reflected on, elaborated on, and specified in more detail. However, it significantly differs from the current notion of “goal” in planning-based information processing because it does not constitute an action in any way. Instead, it constrains and suggests potential actions that could yield productive changes in an emergent process of cognition (Hanne De Jaegher, 2010). In the design process, while trying to achieve goals, individuals are aware of several steps that may need to be executed. Ideally, after completing one step, a designer will take the changes from that step and apply additional steps to accomplish the ultimate goal. However, in reality, designers always need to adjust the past changes according to the current changes. Such changes may even alter the original goals since they may not be able to fit the new goals. Instead of considering the design process as a series of executed behaviors such as scripts or plans, they can be seen as

continuous and emergent improvisation with the environment. Attention and the conscious experience of an agent become the common threads that stitches the flow of each action together (Csikszentmihalyi, 1991)

Once a designer is fully engaged in her design process, she is “in the flow” (Csikszentmihalyi, 1991), and the “rules of the game,” or the “features of the design world” (Schön 1992) have been established. Each design process would presumably feature many sets of constraint satisfaction processes that are operating concurrently, and sometimes competitively, to drive the design process forward (Suwa & Tversky, 1997). In this procedure, we require a lens that filters perception and highlights various *signifiers* (Norman, 2008) for interaction, which can help to identify problems, reveal insights, and further refine the overall (vague) goals. Therefore, establishing deep engagement in the design process involves helping designers to solve ill-defined problems with different constraints. Switching between these constraints and modes of reasoning is therefore critical to sense-making in the early stage of design. The flexibility and low cost of sketching are naturally well-suited to these tasks. Using freehand sketch tools (i.e., tools that quickly become ready-to-hand), designers can quickly become fully immersed in their creative tasks (Dourish, 2004; Ihde, 1979). Conversely, conventional CAD tools have a high barrier to entry and require constantly switching between modes for achieving tasks. Thus, designers must have clear goals before executing commands.

2.5 CAD Programs for the Early Stage of Design

Even though CAD tools help designers to create complex geometries from generative modeling or derive benefits from computer simulations, researchers argue that current CAD systems hinder creativity in the early stages of design (Lawson, 1999; Gül, 2009). Even though generative systems such as Grasshopper (depicted on the right side of Figure 2.4) provide a way of visualizing the “mind-

map” of how parametric rules are constructed and related, current implementations usually require users to switch between two workspaces, creating views for defining rules and representing geometries.

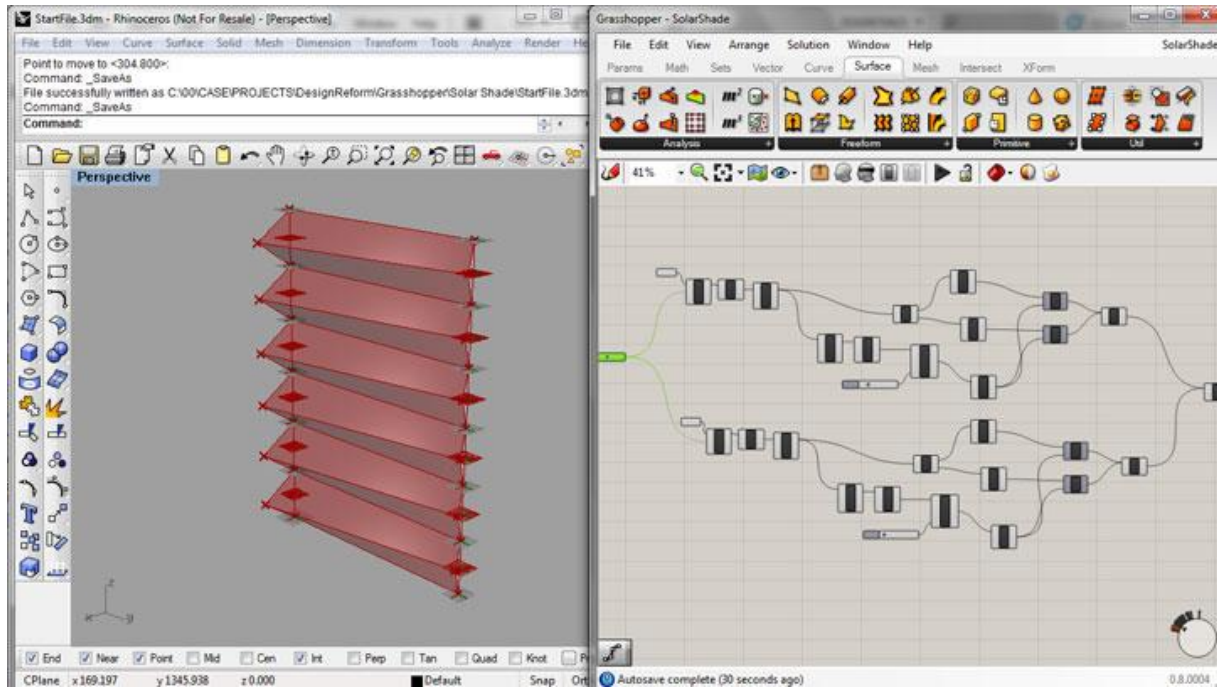


Figure 2.4: The interfaces of Rhinoceros (left) and Grasshopper (right) are very different. The users need to switch between two different interfaces to accomplish design tasks

Since users are forced to think through the steps they need to employ when using CAD tools, these tools do not support improvised and rapid explorations during the early design stage. They often require users to specify many parameters and to manipulate control points on shapes when designers modify their designs. Typically, the mouse is employed as a tool to indicate where the key points are located on a digital canvas after users execute commands. Conversely, during sketching, the interaction is embodied since hand movements directly create lines and shapes on paper. While modifying shapes, designers usually sketch on top of the existing lines to shape the desired form.

Before using CAD applications to create a complex design, users must understand the concepts of the complex geometry modeling that underlies these commands, such as different types of curves, solid modeling, and surface modeling. For instance, as most users cannot distinguish between a

regular arc and a Bezier curve, they would have to use trial and error to create the curves they wish to instantiate. If they are unfamiliar with the concepts, this action may require additional cognitive load to understand how to create the geometries. Thus, by not enabling continuous interactions and feedback, CAD tools do not promote rapid exploration and iterative design process. As a result, they fail to support improvisation and exploration in the early stage of design.

Current CAD tools also do not infer semantic information from users' continuous input. The tools receive discrete input events from the mouse and the keyboard, and then translate them into semantic information, such as the data in Figure 2.2, with UI components. Lacking the capability of recognizing user's intent, the system requires further actions from users. With the addition of more features, new user interface (UI) buttons, boxes, and other elements have emerged in modern CAD programs, especially in BIM programs. However, with sketch and multi-touch inputs, the system should be able to take the continuous stream of input data to infer user intentions.

In reality, instead of generating and exploring creative ideas, traditional CAD tools mainly support incremental refinement, documenting the details of a design, facilitating communication among disciplines, simulating buildings, and validating designs (Tversky, 2002). These claims have been supported by researchers who have argued that CAD tools force designers to focus on how to use the tool rather than to design. As a result, these tools hinder creativity in the design process (Dorta, 2007). Even though architectural firms have widely adopted CAD and BIM tools, they still utilize traditional media for design exploration (Gül, 2009) because designers can engage in "conversation" with these media (Schön, 1992). In his study, Gül found that architectural designers were not able to combine their conceptual explorations (e.g. setting design constraints) and physical realizations (e.g. creating geometries) together while designing in digital environments (Gül, 2009). To ameliorate this problem and better support this conversation (e.g., the dynamic nature in the design process), HCI

researchers have attempted to build prototypes in various media, especially in the domain of sketch interactions.

2.6 Conclusion

In this chapter, we started from the engineering perspectives to discuss the design process for later stages of design. We introduced the tools used by designers in architecture design workflows with their purposes in different stages. We also discussed the importance of interoperability since it serves as a bridge between various tools in the later stages of design. These modern CAD tools provide designers with features for automating their workflows, but users still need to input important parameters and commands in a step-by-step manner.

In the later part of the chapter, we discussed the fundamentally different cognitive processes engaged when designers employ traditional tools in the early stage of design (i.e. sketching) versus CAD tools in the later stages of design. As summarized in Table 2.1, these traditional tools support designers to (1) easily offload cognitive tasks onto the media, (2) enable the iterative nature of the design process so that designers can continually explore different emerging hypotheses, (3) augment embodied movements through continuous feedback from the media, and (4) help designers to document constraints through quick and simple drawings, such as symbols. However, the traditional sketch tools often fail to support (1) generative rule creation, (2) 3D geometry creation, (3) design automation, (4) interoperability, and (5) simulations. In the next chapter, we will review the previous sketch prototypes that enhance our capabilities of creating computer graphics rapidly and iteratively. We will also discuss what we can contribute in supporting the designers to make the workflows smoother in the early stage of design.

Table 2.1: The comparisons between traditional sketch tools and CAD/BIM tools in different aspects discussed in this chapter

Criteria	Traditional Sketch Tools	CAD/BIM Tools
1. Low Cost and Rapid Cognitive Offloading	Easy: lower cognitive loads	Difficult: higher cognitive loads due to the need to think through the steps
2. Iterative Design Process	Easy: designers can continually switch between different types of explorations without clearly defined goals	Difficult: designers need to define tasks prior to executing actions
3. Embodied Thinking Process	Supported: physical media provide continuous feedback while designers move their hands	Limited: designers often need to specify the key points or execute discrete commands
4. Constraint Assignment	Supported: through symbols, bubble diagrams, etc.	Supported: through equations, parameters, etc.
5. Generative Rules	Not supported	Difficult: without training, designers would not be able to create and document complicated forms
6. Design Automation	Not Supported	Supported: through generative design systems, expert systems, etc.
7. Interoperability	Not Supported: designers need to re-document their designs in different platforms	Supported: the system can automatically convert the data for other use cases
8. 3D Geometry Creation	Difficult: often requires proper training for sketching out perspective views	Easy: designers can create 3D geometry while they are creating 2D drawings
9. Simulation	Difficult: often requires heavy cognitive loads or complicated calculation for simulations	Easy: often requires only one click for getting different types of simulations

Chapter III. Related Work

The previous chapter discussed design cognition related to the early stage of the design process. It also described what the current CAD programs provide for designers and what should be improved for supporting the early stage of design. In this chapter, we discuss how the prior sketch prototypes support different aspects of the criteria specified in Table 2.1. In particular, we will reveal how these programs retain similar characteristics as traditional sketching, such as low cost and rapid ideation, and iterative and embodied thinking processes, while adding computational capabilities for achieving tasks that were previously difficult to accomplish, such as sketching in 3D, automatically generating geometries, etc.

3.1 Sketch-Based Interfaces for CAD

As mentioned, one of the first CAD programs, SketchPad (Sutherland, 1963), is arguably a program that employs “sketch”-style input. However, because SketchPad requires users to specify key points with buttons for executing commands, researchers argued that this “sketch” input is fundamentally different from natural sketching used in the physical world (Tversky, 2002). The interaction methods employed in SketchPad are broadly used in modern CAD programs with variations of UI elements and intelligent contextual menus. Nevertheless, many CAD users refer to drawing on a canvas in CAD programs with mouse and keyboard as “sketching”. In the “sketch” interactions in a CAD program, users need to constantly change modes on UI panels, specify key points on the main canvas, adjust curvatures by dragging handles, input various parameters in separate tables, and type in additional information throughout the creation of CAD drawings. Instead of discussing this type of “sketch” interaction, this section reviews the early solutions that analyze continuous sketch inputs for simplifying the tasks commonly seen when using CAD tools, including the following two types of prototypes:

(1) *Prototypes for recognizing drawing intents*: This form of prototype usually aims to understand a group of freehand sketch strokes that form meaning of drawing objects in different scenarios, e.g. for architecture design and interface design.

(2) *Prototypes for 3D geometry creation*: Instead of recognizing a group of sketch strokes, this form of prototype usually generates continuous feedback while the user is sketching.

3.1.1 Sketch Recognition for Recognizing Drawing Intents



Figure 3.1: Electronic Cocktail Napkin analyzes the group of sketch lines in contexts for extracting the meaning of user sketches

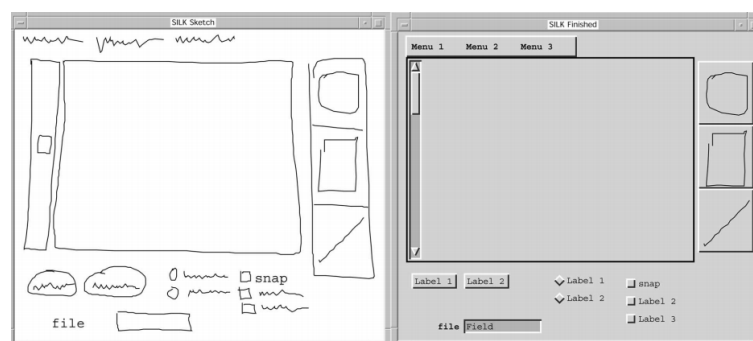


Figure 3.2: SILK converts the user sketch into actual UI components (Landay 1995)

Early sketch prototypes, such as the Electronic Cocktail Napkin (ECN) in Figure 3.1 (Gross, 1996) and Silk in Figure 3.2 (Landay & Myers, 1995), recognize a group of sketch strokes that represent the users' drawing intents, which reduces the need to execute commands manually. Most of these early prototypes analyzed the sketch strokes from simple corner detection, 2D primitive shape

recognitions, and then the composition of these strokes in a group. Through checking the composition of the shapes with preset rules, these systems can infer users' intents from these sketch lines for various purposes, such as making sketch drawings interactive for interface design in Silk, and inferring drawing intents in ECN. These prototypes show the enormous potential of employing computational systems to help users during the early stage of design since they enable freehand sketches instead of forcing designers to specify key points. For instance, VR SketchPad employs a similar approach to recognizing the designer's sketch drawings and automatically converting them to 3D geometries in its database (Do, 2001). However, since this type of recognition method relies on predefined rules, these systems may not be able to comprehensively recognize drawings and symbols that were not defined previously. This is the main reason why these systems offer training sessions so that users can create their own symbols and drawings.

In their studies, Do et al. found that professional architecture designers use a small set of conventional symbols to express design intents in different use cases (Do & Gross, 1997). For instance, architects sketch certain symbols, marks, and hatches with similar configurations that form "primitives" of a drawing object. With the contexts of the surrounding drawings, the system can recognize the design intents and the semantics (e.g. furniture, walls, or windows) of the sketch (Do, 1997). According to the design intents, ECN can offer different tool sets for assisting designers on demand or further affecting sketch recognitions (Do, 1998). Without well-defined contexts, such as specifying use cases or knowing the user's purposes of employing the tool, understanding user drawings accurately in general use cases was still an extremely difficult task.

3.1.2 Simplifying Tasks in 3D Geometry Creation

Other sketch research focuses on supporting users to rapidly create 3D geometry by inferring their interaction intents. Instead of recognizing the drawing intents, these prototypes usually take the contextual information into account for deciding the users' intents of the current actions. SKETCH,

one of the first prototypes in this category, infers some user interactions for simplifying and automating difficult tasks in 3D modeling (Zelevnik et al., 1996). As shown in Table 3.1, the system initially recognizes a set of gestures composed by the combinations of various mouse movements, clicks, and drags to construct five classes of “strokes”. To create 3D gestures, users need to draw the “strokes” in Table 2.1 in specific sequences. For instance, a user can create a cube by drawing three non-collinear strokes that meet at one point. Similarly, the system also has another five sets of mouse “gestures” for users to navigate around the 3D canvas.

Table 2.2: The mouse clicks and movement gestures for five stroke classes (Zelenik et al., 1996)

Mouse Action	Stroke
click and release	dot
click and drag without delaying	axis-aligned line: line follows axis whose screen projection is most nearly parallel to dragged-out segment
click and drag, then “tearing” motion to “rip” line from axis	non-axis-aligned line
click, pause, draw	freehand curve
click with Shift key pressed, draw	freehand curve drawn on surface of objects in scene

These 3D geometries then become the source information to narrow the possibilities of recognition results for automation. For instance, in the scenario of creating a table, the system extends the extruding geometry to the bottom of the tabletop automatically while the user is creating table legs. It can also infer the third dimension when the user draws the shadow on the ground of the target geometry. However, there is a substantial learning curve for users when remembering specific sequences of mouse movements, including clicks and drags, before being able to use the system smoothly. For this reason, Igarashi et al. created Chateau, which suggests possible inference geometries after the user highlights lines in the system through twenty different suggestion engines (Igarashi & Hughes, 2001). The suggestion engines use a rule-based approach to examine the highlighted lines with their contexts to filter out impossible suggestions and generate the geometries.

In the domain of architectural design, Piccolotto developed a system called SketchPad+, which constructs surface meshes through freehand sketch polygons for assisting architectural designers in the early stage of design (Piccolotto, 1998). Unlike SKETCH, which employs gestures for users to execute various commands to create geometries, SketchPad+ utilizes contextual pie menus for executing discrete commands to reduce the need for remembering gestures. However, users draw carefully to create closed shapes on 3D spaces for creating triangle meshes. In this case, users will still need to switch between UI buttons and working canvas to accomplish tasks and create solid shapes, similar to the processes in traditional CAD programs.

In the field of mechanical engineering, Eggli et al. developed a system that allows users to create 3D solid geometries through sketch interactions with the help of the commands from UI buttons (Eggli, Hsu, Brüderlin, & Elber, 1997). This system provides a comprehensive support for creating solid geometries with the help of sketch inputs. Unlike treating each segment of a stroke as a line segment in SKETCH, users can create composite polylines with straight lines and arcs by sketching each segment explicitly. The system can fit these segments separately and combine them together to form a polygon as the profile of a solid shape. Then, users can execute extrusion commands by clicking on UI buttons similar to conventional CAD programs.

These early prototypes were not able to fully account for the flexible nature of sketching for two different reasons: (1) designers sometimes shape an idea through over-sketching; and (2) they sometimes use a stroke for drawing a polyline or polygon. Even though some prototypes, e.g. ECN, support over-sketching features, they often fail to support the accurate segmentation of lines and curves in a polyline. These systems usually require users to draw multiple strokes for constructing a polyline since the sketch segmentation and curve fitting algorithms were not precisely able to recognize what users intended to draw. In SKETCH, the segments of a polyline are created with the sampled raw points of a sketch line. In Eggli's work, instead of segmenting a sketch line into a

polyline with curve and line segments, the users need to explicitly specify these multiple segments separately so that the system can confidently fit them into lines or curves. In other words, to account for the rapid offloading of tasks and the iterative design process, we will require accurate line segmentation and fitting algorithms with the capabilities of over-sketching for modification. These prior works also instruct us not to employ too many gestures, e.g. SKETCH, unless the system can provide suggestions, e.g. Chateau, or users have already trained in a particular domain, e.g. ECN. Even though using UI components may be a choice, e.g. SketchPad+ or Eggli's work, we still need careful interaction designs to keep users engaged in their tasks on the drawing canvas.

3.2 Support of Rapid Creation for 3D Geometries

3.2.1 From Simple Sketch Lines to Complicated 3D Geometries

To reduce the probability of recognition errors, another group of researchers created systems to simplify complicated geometry creation through a minimal number of sketch strokes with well-defined purposes of the tools. These systems usually employ the sketch stroke as a guideline or skeleton path for populating complicated geometries. For instance, Igarashi et al. implemented the system Teddy (Igarashi, Matsuoka, & Tanaka, 1999) that helps users to create free-form 3D geometries, e.g. stuffed animals, by simply sketching a few strokes. With this system, unskilled users can easily and quickly portray 3D free-form geometries. Teddy inflates 2D blobs into 3D free-form polygonal models through the following steps: (1) finds the skeleton spline of the 2D blobs (Figure 3.3), (2) elevates the spline from the drawing plane (Figure 3.4-b), and (3) constructs the triangle meshes between the spline and the blob (Figure 3.4-c, -d). Based on these geometry generation methods, Gingold et al. generalized the tool by adding structured annotations on the geometry for more comprehensive 3D geometry generations and manipulations as shown in Figure 3.5 (Gingold, Igarashi, & Zorin, 2009).

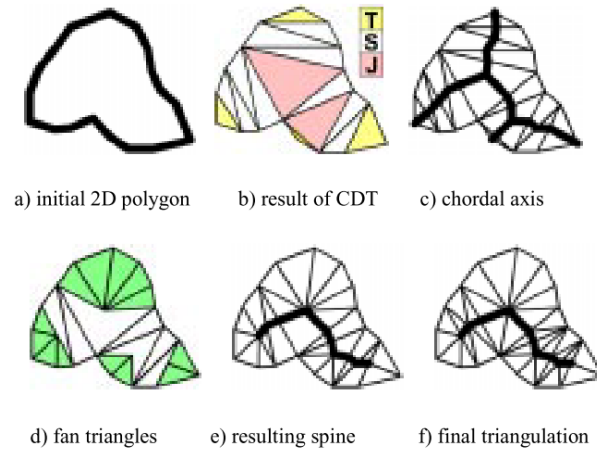


Figure 3.3: The process of finding skeleton spline in Teddy (Igarashi et al., 1999).

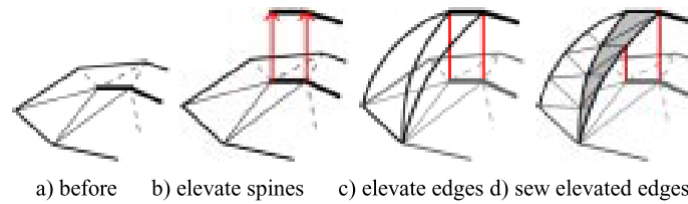


Figure 3.4: The process of elevating spine and construct polygonal meshes in Teddy (Igarashi et al., 1999).

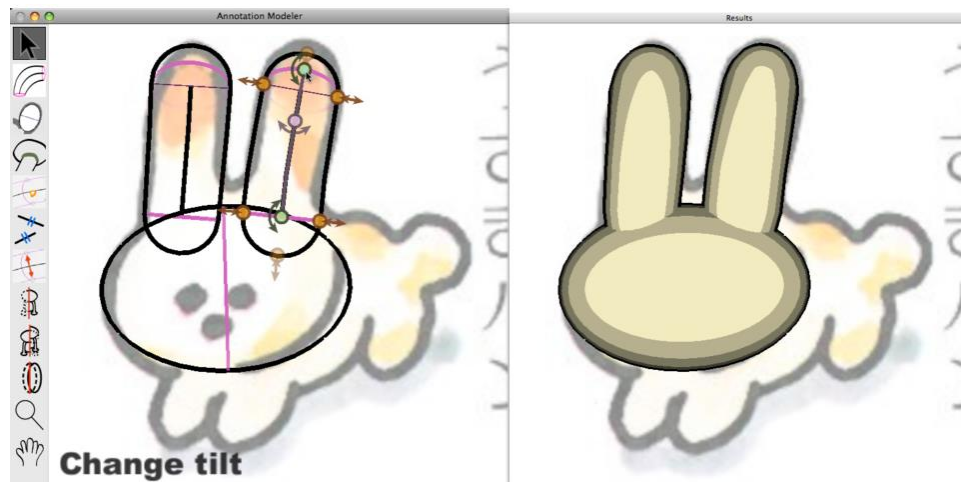


Figure 3.5: Left: after sketching a simple blob, the system will automatically generate the structured annotations to provide the opportunities for interactions on the geometries. Right: the results of interactions from on the right (Gingold et al., 2009).

This category of sketch prototypes is also used in applying guidelines with generative algorithms for generating complicated geometries. For instance, in Figure 3.6, TreeSketch generates complicated tree branches and foliage from several simple sketch lines with predefined generative algorithms (Longay, Runions, Boudon, & Prusinkiewicz, 2012). The users can adjust the types of the trees, the

amount of foliage, and other parameters through interactions with the UI components, such as sliders and radio buttons. When modifying the existing trees, the system predefines some key areas that users may start sketching and categorizes the tree generation methods according to these key areas, such as controlling the direction of the trunk or branch in different types of trees.

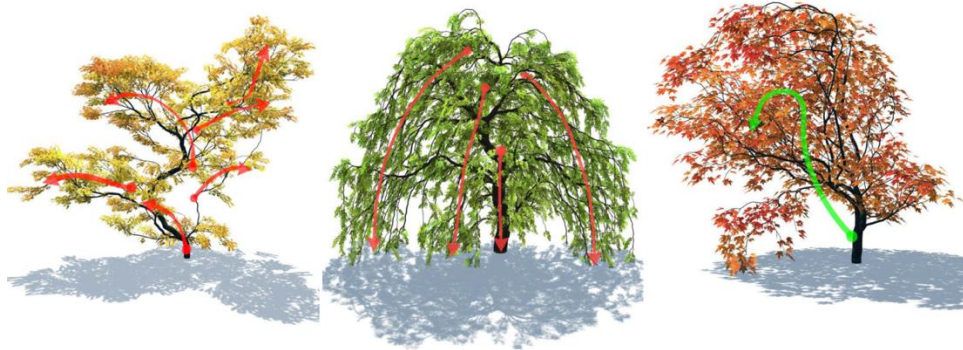


Figure 3.6: Users can select the types of the trees they are creating, and sketch out the direction of the branches. The system will automatically generate branches and foliage according to the sketch directions. (Longay et al. 2012)

Since the purposes of the tools are well-defined, user intentions are narrowed to only several possibilities during the sketching interactions for creating 3D geometries. Thus, these types of systems usually support only specific tasks, which is difficult for our use cases that require flexibility for the iterative process in the early stage of designs.

3.2.2 Supporting Idea Explorations in 3D

To address this problem, as shown in Figure 3.7, Yang et al. created a system that attempts to fit the analyzed sketch strokes onto preset templates in the system (Yang, Sharon, & van de Panne, 2005). Before the system can recognize any sketches, it needs to have 3D models and corresponding 2D template sketch lines. Similar to ECN, it analyzes sketch lines through basic sketch segmentation, grouping of intersections, and relationships between lines as metrics for a template. When the system receives a set of sketch strokes, it will try to identify the best possible template, calculate the transformation between the sketch strokes and the template strokes, and then apply these transformations to the corresponding 3D model. Even though it extends the possibilities of 3D

geometries, it still requires collecting these templates and models for 3D geometry transformation prior to using it.

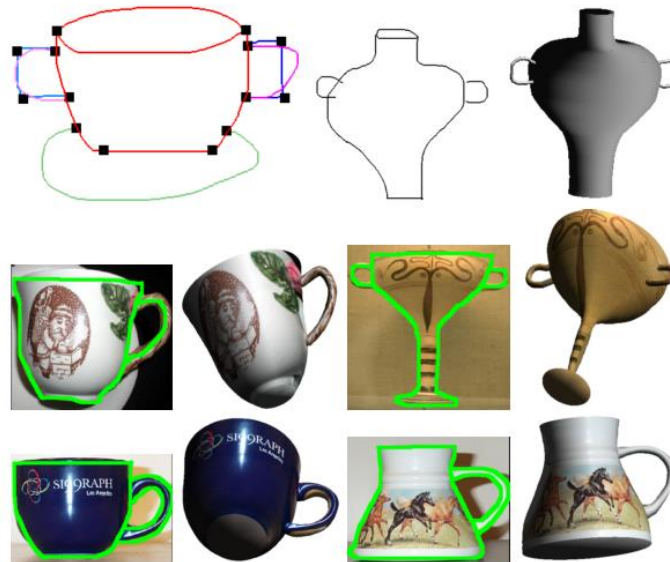


Figure 3.7: Yang et al. proposed a method to fit the sketch lines into predefined template, and then create 3D models from transformations of the templates (Yang et al. 2005).

To remove the requirement of creating templates and increase the flexibility of 3D object creation, another category of research prototypes focuses on projecting the 2D sketch strokes in 3D spaces directly. The goal of these tools is to create an environment in which the designers can sketch their design ideas quickly without the burden of having to consider how to create 3D geometries. As shown in Figure 3.8, Tolba et al. created one of the first systems in this category that takes the user's 2D perspective freehand sketches, and projects onto a unit sphere through manual adjustment and automatic recognitions of vanishing points for calculating the 3D camera location (Tolba, Dorsey, & McMillan, 1999). Once users adjust the sketch lines with automatic feedback from the system, they can rotate to different perspective views. However, when sketching, users were not able to receive feedback about how the system recognizes the lines. Further, the system only places lines into 3D space instead of 3D meshes or solid geometries, and thus it is difficult to be re-utilized in other stages of design.

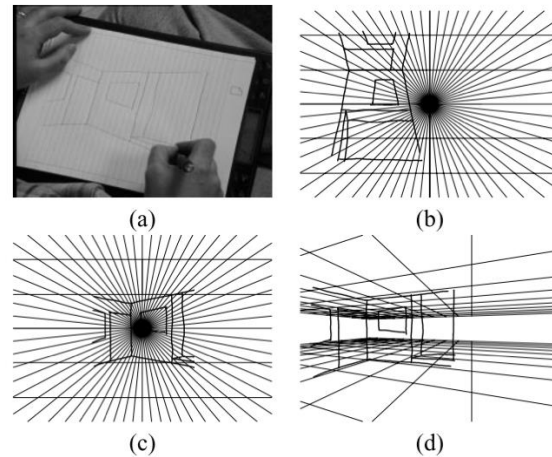


Figure 3.8: The process of using the perspective sketch tools Tolba proposed. (a) Person start from a freehand sketch. (b) Import the sketch lines into the tool, (c) Translate and rotate the drawing to fit the vanishing points, (d) Manual or automatic tools provided by the system will fit the sketch lines and relocate these lines into the proper location in 3D so that the user can change the views

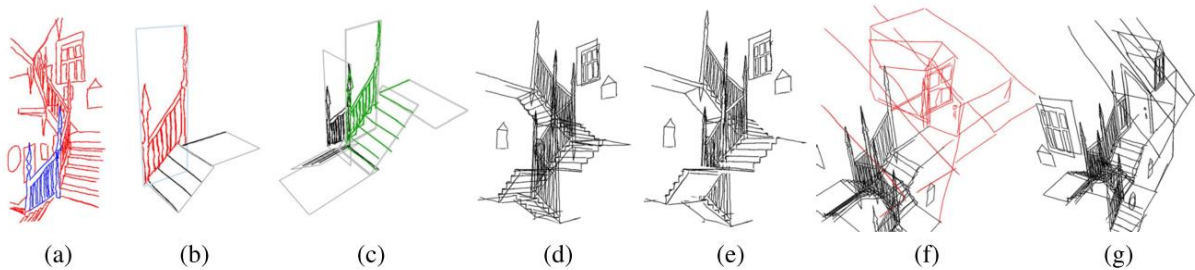


Figure 3.9: Dorsey's "The Mental Canvas" prototype helps designers to fuse their perspective drawings into a 3D drawing

Having recognizing this problem, Schweikardt and Gross created a program that directly translates the 2D perspective drawings into 3D solid geometries automatically by using the Huffman-Clowes labelling scheme (Schweikardt & Gross, 2000). By identifying a corner that projects toward and recedes away from the viewer as “+++”, and “---” respectively, it is possible to provide sufficient information for constructing a 3D solid geometry. Dorsey et al. then created an interactive prototype, “The Mental Canvas” (TMC), that enables designers to specify drawing planes for design sketches. As shown in Figure 3.9, the system then organizes and fuses the intersections of these sketched lines together in a 3D sketch environment (Dorsey, Xu, Smedresman, Rushmeier, & McMillan, 2007). By using two optimization-based algorithms for 3D reconstruction and finite element analysis, in Figure 3.10, Masry et al. developed an algorithm that can reconstruct 2D sketch lines directly into 3D

geometries (Masry & Lipson, 2007). However, these tools only aim to support the creation of 3D lines and geometries when users need it, instead of trying to infer 3D geometries interactively from the sketch in real time. Thus, users often need to carefully sketch their perspective lines for accurate recognition. The systems are also limited to recognizing simple sketch lines without any over-sketching on the existing lines.

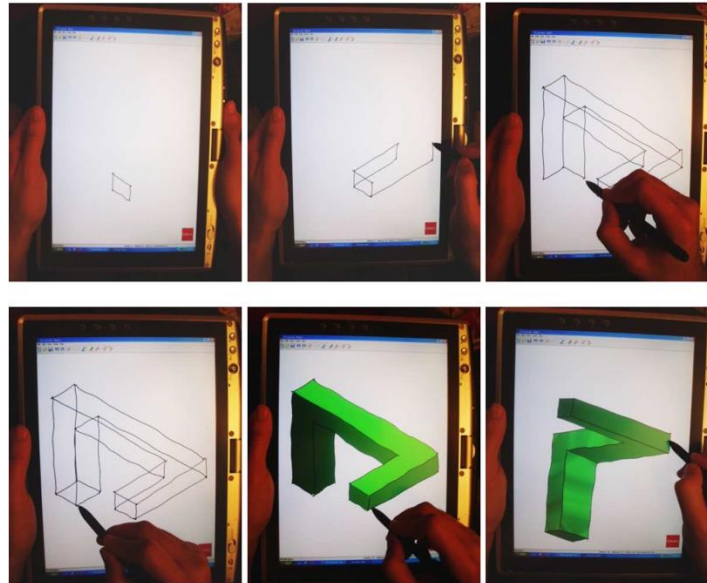


Figure 3.10: The system analyzes the positions and connections of the sketched lines, and then it optimizes the 3D projections of sketch lines for finite element analysis to determine the mesh geometries (Masry & Lipson, 2007).

In recent years, sketch interaction researchers have turned their attention to determining how to utilize traditional perspective sketch techniques for converting sketch lines to 3D geometries. These systems usually provide assistive representations that serve the purposes of communicating the system's interpretations of the user's behaviors as well as adding constraints for recognizing the user's inputs in real time. For instance, designers normally draw scaffold lines, construction lines, or planes as guidance when sketching in a perspective view. Sketch systems, such as ILoveSketch (Bae, Balakrishnan, & Singh, 2008, 2009) and Analytic Drawing Tool (Schmidt, Khan, Singh, & Kurtenbach, 2009) in Figure 3.11, enable users to sketch these assistant lines (Figure 3.11-a) or provide these lines interactively to facilitate computing 3D sketch lines in a natural way. With these scaffold lines, users

can draw curves naturally (Figure 3.11-b) with over-sketching techniques to construct meaningful 3D geometries (Figure 3.11-c).

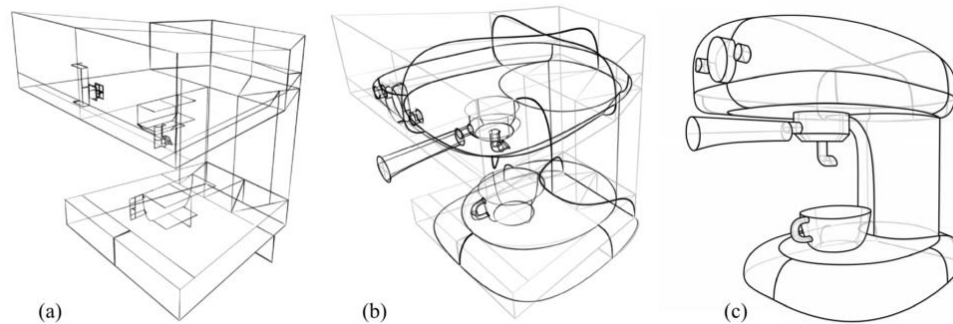


Figure 3.11: Analytic drawing tool enables users to draw 3D scaffold lines (a) that help the system locate the user's sketch lines (b) in 3D space accurately (c) (Schmidt et al., 2009).

These types of prototypes usually provide many convenient methods for rapid idea creation. For instance, EveryBodyLovesSketch provides the “mirroring feature” so that users do not need to draw both sides of the mirrored object redundantly (Bae, Balakrishnan, & Singh, 2009). Analytic Drawing Tool infers constraints for setting 3D curve locations similar to what SKETCH offers, but with more subtle recognitions that can resolve ambiguous conditions, such as curve line intersections in 3D spaces and constrained parts of the sketch strokes on a plane instead of entire strokes (Schmidt, Khan, Singh, & Kurtenbach, 2009). These tools also provide similar features for designers to form desired sketch lines by combining multiple sketch lines into one when designers draw repeatedly within a short time window. However, they do not decompose a sketch stroke into multiple lines and curve segments for inferring users' drawing intentions.

Chen et al. created a system that analyzes the intersections of sketch lines to infer 3D positions (Chen, Kang, Xu, Dorsey, & Shum, 2008). They categorized the types of the intersections (Figure 3.12-left) and observed the possible 3D positions of the lines connecting with the intersections for the methods of 3D position inferences (Figure 3.12-right). However, the types of 3D shapes that this prototype can detect are limited.

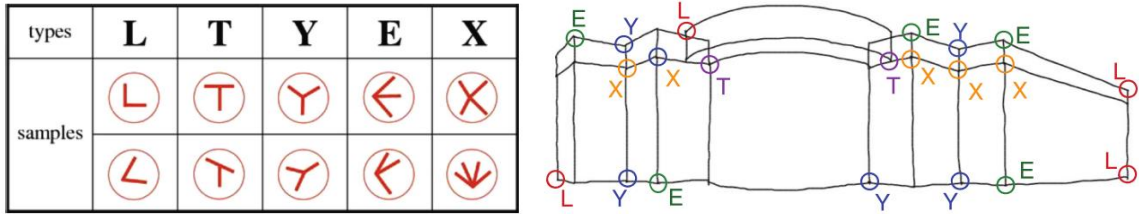


Figure 3.12: Left: the type of intersections Chen’s prototype is using. Right: the user needs to sketch the perspective views carefully for the system to reconstruct the 3D positions (Chen et al. 2008).

In addition to analyzing the types of interactions, the most recent prototype, True2Form, infers 3D curved lines from 2D sketch strokes automatically by leveraging insights from human perception through surveys using crowdsourcing techniques (Xu et al., 2014). They determined that humans tend to perceive perspective sketches with the following four regularities: (1) Orthogonality, (2) Parallelism, (3) Symmetry, and (4) Applicability, based on preset rules from observations. As shown in Figure 3.13, True2Form intelligently selects the intersections for evaluating 3D locations by using these four regularities as passes to optimize the results and then computes the curvatures in 3D space. These prototypes not only provide an intuitive way of creating 3D drawings from 2D sketches, but also help designers to draw perspective views that require much more effort using traditional media, such as pen and paper. However, since the geometries produced by using these prototypes are still only line or surface models, meaning that they only contain meshes with limited topological information, the designers would still need to reconstruct the geometries and their relationships in the design process.

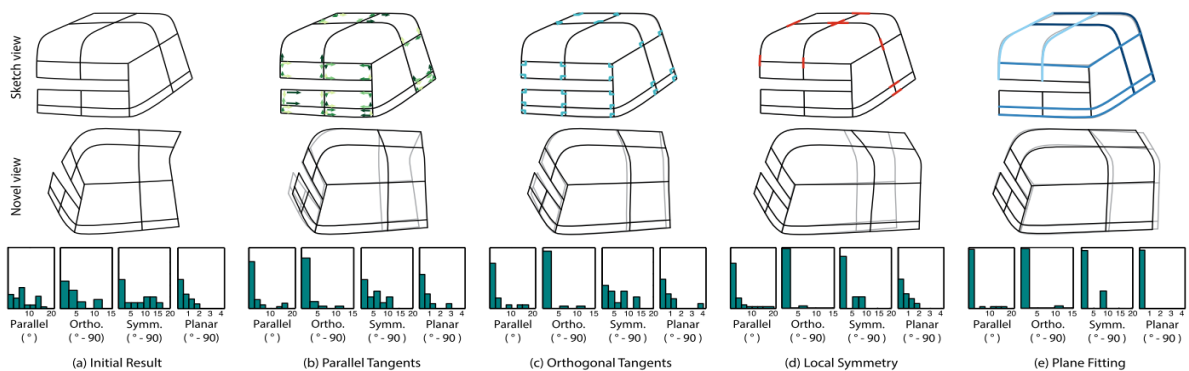


Figure 3.13: True2Form uses four regularities for picking the intersections and optimizing the curve fitting in 3D space (Xu et al. 2014).

3.3 Automation in 3D Computer Graphics

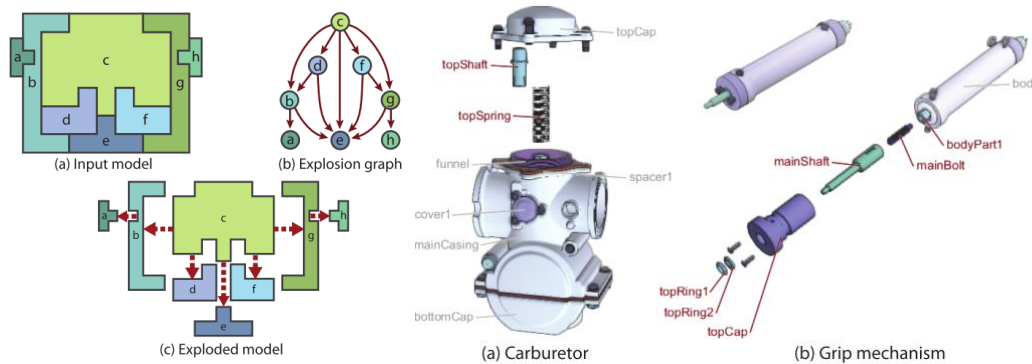


Figure 3.14: Li's system automatically analyzes the topology of different parts of a model (left), and generates an interactive exploded diagram for users (Li et al. 2008).

Geometry analysis is useful for revealing interaction opportunities, assigning constraints automatically, creating topologies, and generating new geometries without manual specifications. In the examples mentioned in the previous sections, the prototypes included some automation methods for improving usability while creating the geometries. Since interacting with 3D geometries is usually complicated and requires higher cognitive loads than 2D geometries, computer graphic researchers have focused on developing algorithms and optimization methods for assisting these interactions in the past decade. One interesting application of automation in 3D interactions, shown in Figure 3.14, is to create 3D exploded view diagrams automatically to support interactive explorations in complicated 3D mechanical models after decomposing the geometries with hierarchical topologies (Li, Agrawala, Curless, & Salesin, 2008).

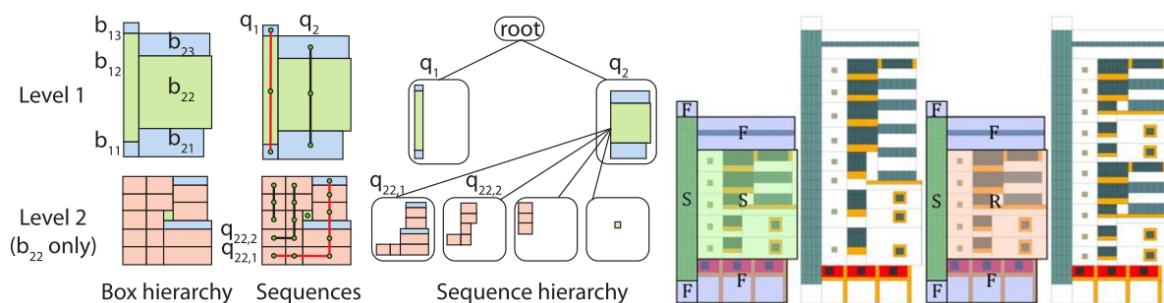


Figure 3.15: After analyzing topological structure of the building façade (left), the system knows which parts that can be scratched and squeezed while the user change the height of the building (right) (Lin et al. 2011).

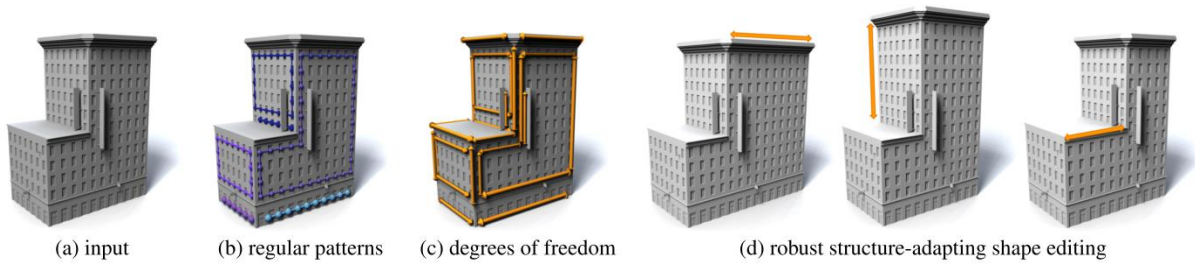


Figure 3.16: After analyzing the pattern on the façade, users can stretch the model in different directions.

Instead of decomposing the existing geometries for illustration purposes, researchers use similar methods for analyzing the topological relationships of geometries, and then preserve them for simplifying geometry generation while stretching and squeezing the façade on an architectural model (shown in Figure 3.15) (Lin et al., 2011) and furniture models (Zheng, Fu, Cohen-Or, Au, & Tai, 2011). In addition, Bokeloh et al. used a similar shape analysis approach to automate pattern generations by building an algebraic model in which the shapes in the pattern conform to a set of automatically generated linear constraints for more flexible manipulation of building models (Bokeloh, Wand, Seidel, & Koltun, 2012). In Figure 3.16, while the users interact with the building façade by dragging the handles, the system generates windows to fit the existing pattern and fill up the empty spaces. These systems typically require users to initially create geometries or patterns so that they can “analyze” them.

Recently, researchers have argued that a system should be able to analyze the topological relationships intelligently without users manually specifying them in 3D free-form models. Thus, Fish et al. created a meta-representation of topologies from analyzing 3D triangle mesh models to enable geometry manipulations (Fish et al., 2014). Mitra et al. used a slightly different approach to achieve direct manipulations on geometries (Mitra et al., 2014). These two systems automatically analyze meshes and categorize different groups of meshes into different parts of a model. For instance, Mitra’s system calculates the probability density from geometrical relations represented as

histograms in Figure 3.17 that show possible ways in which shapes can be changed. These histograms serve as “meta-representations” for users to manipulate and produce alternatives of a 3D model.

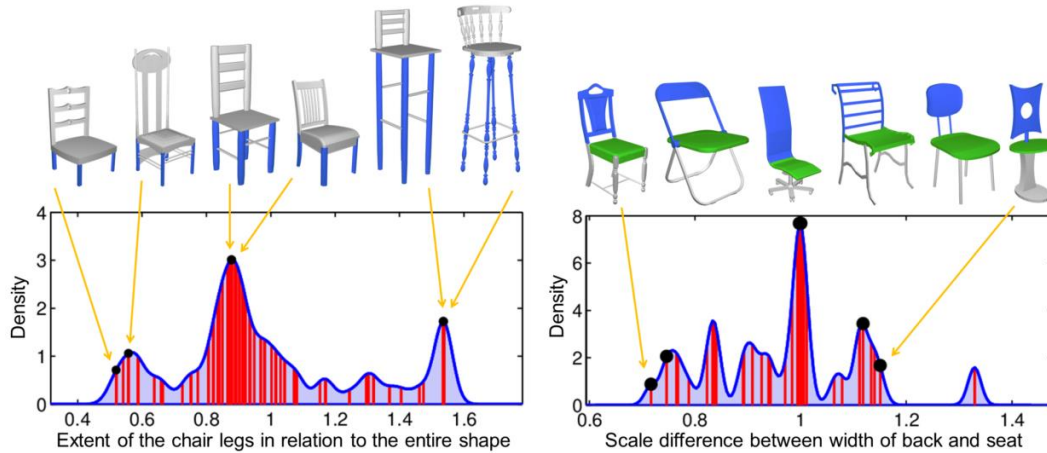


Figure 3.17: Through analyze the geometry, the system gets the topology and compute the probability density of the shapes. The users can change the density to manipulate the geometry (Fish et al. 2014)

These prototypes usually take existing geometries and models as a starting point. Through shape analysis, the systems generate internal representations of the analyzed results. The users can then interact with these outcomes directly or through meta-representations, such as in the lower part of Figure 3.17. However, these prototypes usually do not address the issues of how users create geometries. Interactions during the design process are also not the focus of these systems. Nevertheless, we should be able to apply some of the automation approaches from these studies to support the interactions in our use cases.

3.4 Creating Constraints and Rules through Sketch Interactions

The previous section introduced prototypes that employ ways of inferring topologies and constraints so that the systems can automate complicated tasks, such as pattern generation, geometry illustration, and free-form manipulations. Prior to using these programs, users typically need to first create geometries for the system to analyze. In this dissertation, we focus on how to create designs in the early stage of design. Some prior works have focused on helping designers to

explore constraints in this design process. In his SketchPad system, Ivan Sutherland recognized the importance of establishing constraints between geometries, and he employed these constraint settings in the prototype, such as same length and parallel segment constraints (Sutherland, 1963). These constraints allow the drawing to maintain the designer's expressed intent, such as maintaining perpendicular or parallel relationships between two line segments. In his dissertation, Gross developed a constraint management system to help architects to explore various constraints during the design process (Gross, 1990). Later, Anderl et al. provided a framework that described the relationships among geometric constraints, engineering constraints, parameters, and the history of parameters in a product model (Anderl & Mendgen, 1995). According to this framework, Anderl et al. designed and built an example application that used a rule-based approach with UI buttons and widgets to help mechanical designers to create geometries with topologies and constraints. This is one of the early example applications that supported designers to create not only geometries, but also comprehensive topologies that describe the relationships between shapes for automating design tasks in a single 3D environment.

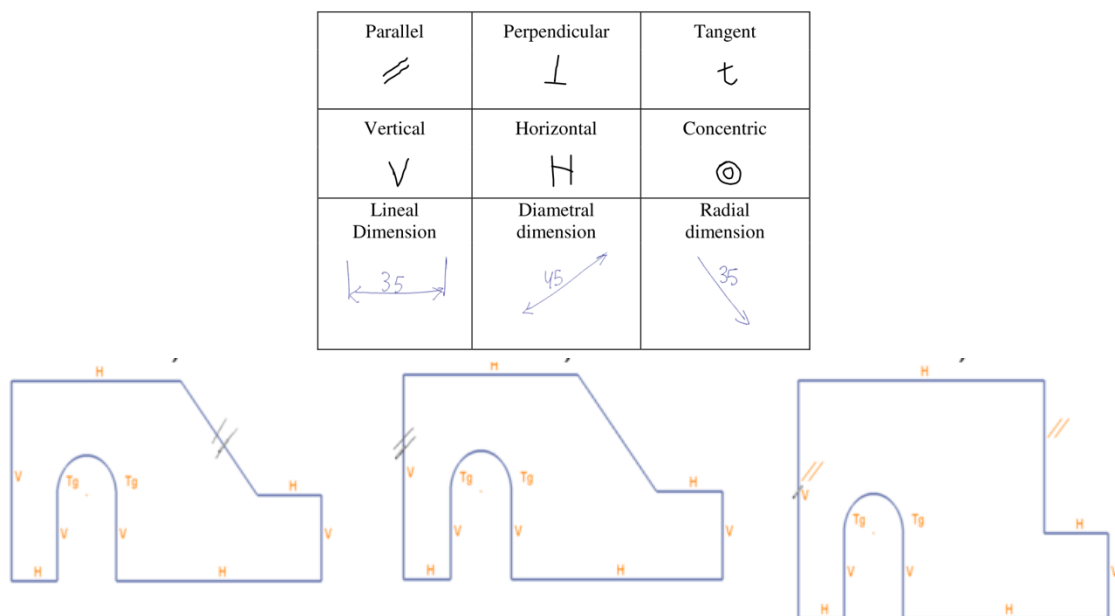


Figure 3.18: In ParSketch, users can sketch a polygon and use the simple symbols (top) as the gestures to define parallel constraints between line segments (bottom). In the bottom figure, users draw parallel gesture in the left and middle figure, the system resolves the constraint automatically in the right figure (Naya et al., 2007).

Instead of using UI buttons, researchers realized that they can use sketch gestures for executing commands and setting constraints to allow users to engage in their sketch tasks. The earliest example we found is SKETCH, built in 1996 (Zelevnik, 1996). Since then, the gestures employed in these early prototypes have been broadly used for executing commands directly to create and modify geometries, such as erasing and copying geometries. In 2007, Naya et al. created ParSketch, which employs the sketch gestures as symbols for only establishing 2D geometric constraints (Naya, Contero, Aleixos, & Company, 2007). As shown in Figure 3.18-top, users can employ conventional mathematical symbols to set up constraints between geometries, such as perpendicular, parallel, vertical, horizontal, concentric, and other annotations utilized in the design processes. Through these gestures, users can create precise engineering drawings while sketching on a canvas; they would not need to jump back to UI buttons to execute commands to accomplish the tasks.

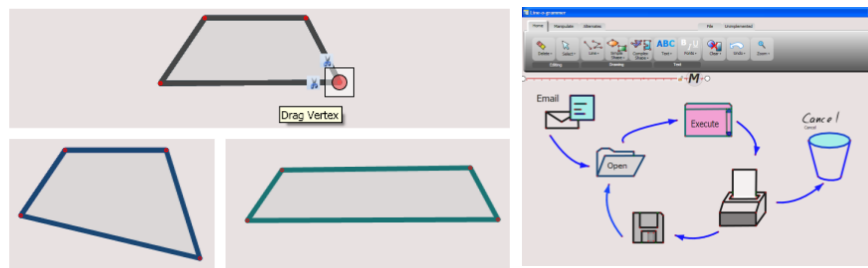


Figure 3.19: In Lineogrammer, users can quickly sketch out simple shapes and drag the vertex around (left). The system responds continuously to users while dragging. Users can also set constraints between arrows and shapes (right). The system can maintain the constraints between lines/shapes after users modify the locations of the shape.

Several researchers have further elaborated this type of interaction by adding animations during the execution of commands when sketching quick diagrams in Figure 3.19 (Zelevnik, Bragdon, Liu, & Forsberg, 2008), and creating accurate 2D drawings (Johnson, Gross, Do, & Hong, 2012). For instance, in Lineogrammer (Zelevnik et al., 2008), users can set the constraints between arrows and shapes. The system maintains the relationships while users are moving one of the shapes around continuously. In SIMI (Johnson et al. 2012), users can employ gestures to explicitly assign constraints similar to ParSketch. The system will propagate continuous transitions when it solves the constraints while users are modifying the locations at one part of the shape. These recent prototypes reduce the

need for switching between different modes when executing commands and setting constraints in one working environment.

In the domain of sketch for illustrations and animations, some researchers have created sketch prototypes for simplifying pattern generations, such as 2D texture design in Figure 3.20 (Kazi, Igarashi, Zhao, & Davis, 2012) and dynamic and interactive illustrations in Figure 3.21 (Kazi, Chevalier, Grossman, & Fitzmaurice, 2014). Unlike the works mentioned previously such as TreeSketch, users can specify their generative patterns and reuse the patterns from simple sketch lines with the help of UI widgets when setting up predefined constraints and rules. However, to the best of our knowledge, the area of capturing users' natural and continuous interactions, such as sketch inputs, to create 3D geometries with constraints, and topological and semantic information, for design automation in the early stage of design, remains unexplored.



Figure 3.20: In Vignette (a) Users can draw a pattern and a simple sketch line. (b) The system generates the entire seaweed after users lift up the pen. (c) Users can repeat the same interaction to generate multiple seaweeds (Kazi et al. 2012).

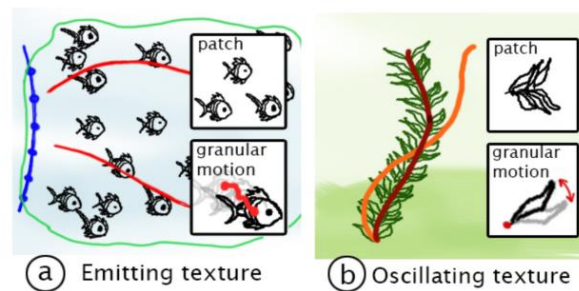


Figure 3.21: In Kitty, users can specify the ways how illustration object move with the assistant of UI buttons (Kazi et al. 2014).

3.5 The Right Interactions

In this chapter, we examined previous research works related to sketch interactions and 3D geometry creation. More importantly, as discussed in Chapter 2, the real challenge of using current

CAD tools in the early stage of design is due to the complicated interactions required to create geometrical and topological information. It is difficult for designers to quickly offload their tasks through embodied interactions using current CAD tools.

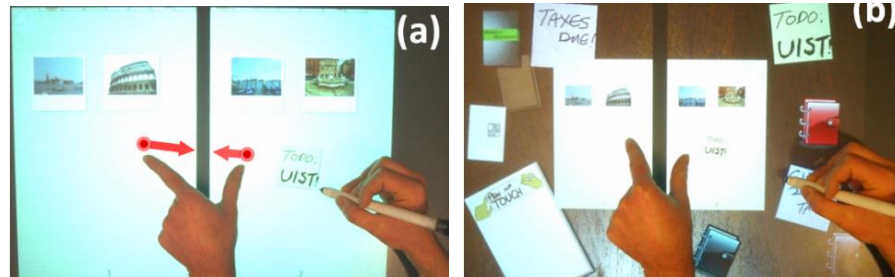


Figure 3.22: The combination of multi-touch and pen interactions create a rich interaction environment. The users can communicate complicated commands through the sets of pen and multi-touch gestures. (Hinckley et al., 2010)

HCI researchers have tried to solve this problem by employing continuous interactions that facilitate direct and rapid manipulations of virtual objects on computer screens. For instance, in Figure 3.22, Hinckley et al. employed the combination of pen and multi-touch gestures for users to communicate their intents and accomplish different commands rapidly (Hinckley et al., 2010). In their design of interactions, the same combination of interaction signals, such as single touch and stylus movements, will be interpreted differently by the system in different contexts with different virtual objects. For instance, users can hold and cut a photo into two pieces by placing one finger on the photo and performing a stroke on it with the stylus, similar to what we would do when cutting a paper. However, using one finger to hold onto a polyline while drawing with the stylus would straighten the line. The combinations of pen and multi-touch interactions also create a rich range of gestures that users can remember without a substantial learning curve since different interaction modalities can be stored in separate chunks of short-term memory.

In the domain of geometry creation, the most related prototype we have seen employs different multi-touch gestures for rapid shape manipulations in a 3D animation environment in Eden (Kin et al., 2011) and simple solid modeling applications (Kang, Kim, Suzuki, & Han, 2015). As shown in Figure 3.23-left, Eden employs many different multi-touch gestures for 3D shape manipulations from

predefined objects, e.g. scaling, translations, and rotations. However, users cannot create and design comprehensive objects in their prototypes. As shown in Figure 3.23-right, even though Kang's prototype enables the capability of creating geometries, users will not be able to edit the shapes as flexibly as the sketch prototypes mentioned before. Users still need to engage other tools for specifying topological or semantic information.

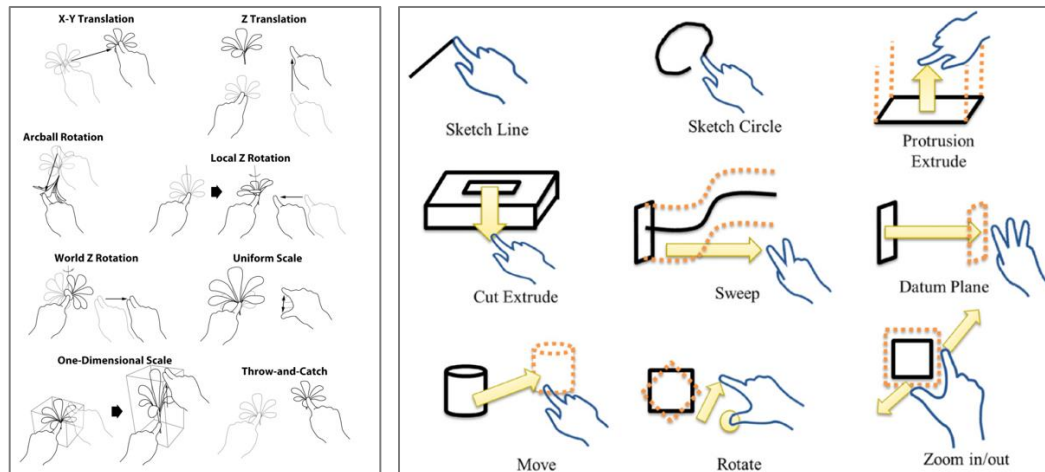


Figure 3.23: The multi-touch gestures employed by Eden (left) (Kin et al., 2011) and Kang's prototype (right) (Kang et al. 2015).

3.6 Conclusion

In the field of architectural design, a gap still exists in these novel interaction prototypes concerning what current CAD (BIM) tools can offer. As mentioned in Chapter 2, these modern BIM tools not only help us to build geometries but also topologies, properties, and parameters of geometries, which offer huge advantages for design automation when we modify and edit designs over time. As shown in Table 3.1 and mentioned previously, because of the variety of benefits provided by sketch interactions, researchers have developed prototypes that utilize the power of sketch inputs for designers during their various design activities. These prototypes shed new light on how researchers can improve interactions between internal (mind) and external (world) representations during the early design process, especially for improving low-cost cognitive offload, supporting an iterative design process, and enhancing embodied thinking. Applying these new

interface modalities enables designers to utilize more intuitive ways of translating user behaviors into computational geometries, e.g. over-sketching, line editing, sketching 2D to 3D geometries, indicating gestures *in situ*, and performing multi-touch manipulations.

However, the tools mentioned in this chapter did not fully support the features that a CAD program provides, such as design automation, interoperability, generative geometries, and simulations. On the other hand, modern CAD tools usually help designers with these aspects in the design process. However, they often employ complicated commands through different sets of UI components. Thus, we need to carefully research and design the interactions that can assist designers to engage in their tasks in the design process.

In this chapter, as summarized in Table 3.1, we reviewed tools that enhance users' capability, help to accomplish difficult tasks such as sketching in 3D, and also simplify complicated processes. However, to the best of our knowledge, we have not seen studies or prototypes that account for all of the criteria that we established in Chapter 2. In the next chapter, we will describe the interaction design of our proposed prototype using a persona and a scenario to describe one method of fulfilling the criteria identified in Chapter 2 and 3.

Table 3.1: The summary of different sketch prototypes.

Criteria	ECN/SILK	SKETCH /Chateau	Teddy /TreeSketch	The Mental Canvas	Marsy's 3D prototype	ILoveSketch /True2Form
Low Cost and Rapid Cognitive Offloading	Supported	Supported	Only for specific tasks	Only for perspective line drawings	Only for simple 3D models	Supported
Iterative Design Process	Editing at 2D object level	Editing at 3D object level	Editing at 3D object level	Unknown	Editing at 3D line level	Editing at 2D line level
Embodied Thinking Process	Results are generated after sketching an object	Partially supported. Some results are generated after performing gestures	Results are generated after sketching a line	Results are generated after sketching a line	Results are generated after sketching an object	Supported. Some results generated during sketch, other results generated after sketching several lines
Constraint Assignment	Through sketching symbols	Through gestures or automatically inferred	Very limited, preset by system logic	Not supported	Not supported	Limited, Automatically inferred

Table 3.1 continued

Construct Generative Rules	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Design Automation	Through automatically recognizing drawing intents	Not supported	Supported, but very specific type of geometries	Not supported	Not supported	Not supported
Interoperability in Design Process	Limited	Meshes	Meshes	Only lines	Meshes	Meshes
3D Geometry Creation	Not supported	Supported , through sketch gestures	Supported , through automatic inference	Supported , but only 3D lines	Supported , inferred from careful freehand sketch	Supported , through the assistant lines from system such as construction lines
Simulation	Supported , design related simulations	Not supported	Not supported	Not supported	Supported , structure analysis	Not supported

Chapter IV. Use Scenario of SolidSketch

This chapter introduces a new interaction paradigm for creating 3D solid and parametric models by describing a persona, an architect Peter, and explaining how he uses sketch and multi-touch interactions to accomplish early design tasks. As shown in Figure 4.1, SolidSketch provides only a few buttons, which enables Peter to focus on his design tasks while using this program. In SolidSketch, Peter usually employs multi-touch and sketch interactions to create geometries and execute commands right on the canvas where he explores his designs. Figure 4.2 shows the main commands and functions provided by multi-touch and sketch interactions in SolidSketch. This chapter will describe how Peter smoothly generates early design concepts and converts them into meaningful 3D models with parametric, topological, and basic semantic information in a short time using these interactions.

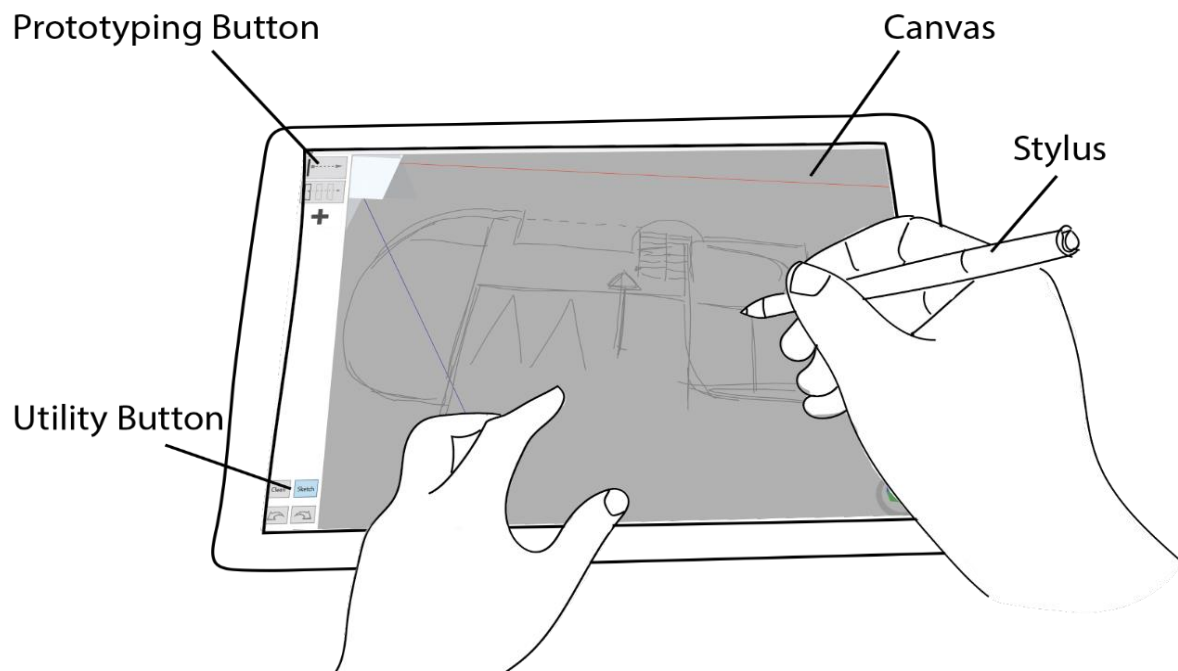







Figure 4.1: Peter uses SolidSketch to sketch out his initial design ideas and build rough models that contains building information. He mainly interacts with the 3D canvas with few clicks on left side of the UI for changing the prototyping mode in Prototyping Buttons, undo/redo, and turn off the sketch recognition for freehand sketch mode in utility buttons.

Stylus Gestures

Gesture	Name	Function
	dot	Select and Unselect
	lasso	Select line Connect segments Create axis widget
	right angle	Make right angle
	tick	Make same lengths
	scratch	Boolean subtraction Make holes Delete objects

Multi-touch Interactions


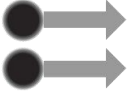



Gesture	Function
	Orbit View Move Shape Adjust Intervals in Array
	Pan View
	Zoom in
	Adjust Elements in Array Clone
	Rotate Shape

Figure 4.2: Stylus gestures (left) and multi-touch interactions (right) that are employed in SolidSketch. The dots in the stylus gestures and multi-touch interactions represent the starting points of the gestures, while the arrows represent the ending locations.

4.1 Preliminary Design

Peter is an architect in an architectural firm who begins designing a house located in a residential area using SolidSketch. After clicking on the “Freehand Sketch” button in the UI, he first studies the rough plans in the freehand sketch mode by sketching out his ideas similar to what he would do with a regular pen and paper, as shown in Figure 4.4-left. He can also use conventional multi-touch gestures as shown in the right side of Figure 4.2 to navigate around in the three-dimensional spaces, e.g. 1) a pinching gesture zooms the view in and out; 2) two-finger dragging pans the 3D camera; and 3) one-finger movements orbit the 3D camera.

He then uses the stylus to perform a lasso gesture (Figure 4.2) to create an axis widget (Figure 4.3) at the location where he wants to study the sections. He hovers on the XZ surface of the widget to highlight the surface and taps on it to switch the drawing plane to align with that surface. With this generic way of selecting different surfaces in 3D space, he can study the sections and perspective

views efficiently, as shown in Figure 4.4. He quickly sketches out the initial concepts and is now ready to design the building with 3D geometries.

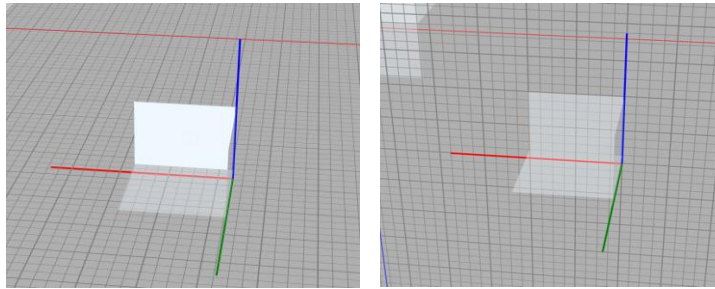


Figure 4.3: The axis widget. Different colors represent different axes: 1) red: X axis; 2) green: Y axis; and 3) blue: Z axis. Peter can click on the white highlighted area between the axes to switch the drawing grid planes. In this case, he clicks on XZ plane, the highlighted plane in the left figure, to switch to the vertical drawing plane.

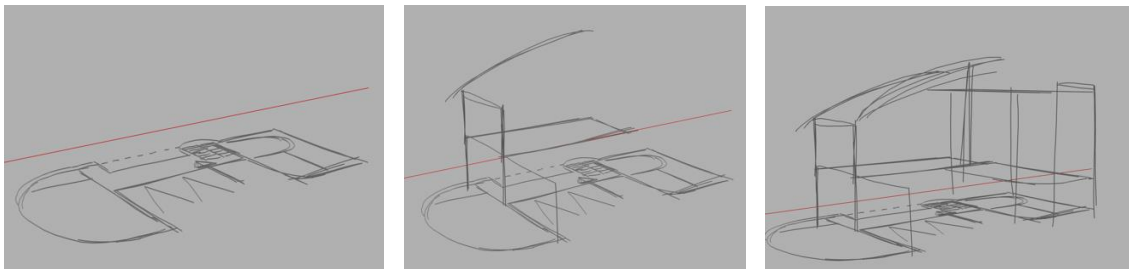


Figure 4.4: Peter can sketch out his initial ideas of the design quickly in freehand sketch mode. He can jump between sketch planes and create section and perspective views quickly by using gesture and the axis widget.

4.2 Building Templates with Rules and Parameters

Once he has a rough picture of how he wants the building to be laid out (Figure 4.4-right), he realizes that he needs a certain type of wall with particular width and height in the majority of the building. He then starts to define a wall with customized thickness and height. He enters the ‘template composition mode’ by hitting the Plus button on the left panel of the UI. Once in this mode, he starts to use sketch and multi-touch interactions to specify a custom wall template. This template consists of the dimensions and constraints of the geometries created by Peter during the template creation process.

After he clicks the Plus button to enter the template composition mode, the system will display an orange guideline (Figure 4.5-1). Then, he draws a line segment that defines the width of the desired wall, as shown in Figure 4.5-2. Next, in Figure 4.5-3, after clicking on the “extending

dimension” tool shown in the upper left corner of Figure 4.1, he traces the orange line to signify to the system that he wants to use it as the directional path for extending the width segment. The system offsets the sketch line and creates a stripe shape in real time while Peter is sketching. Then, he extrudes the parametrically defined profile created in steps 1-3 to yield the wall geometry in Figure 4.5-4.

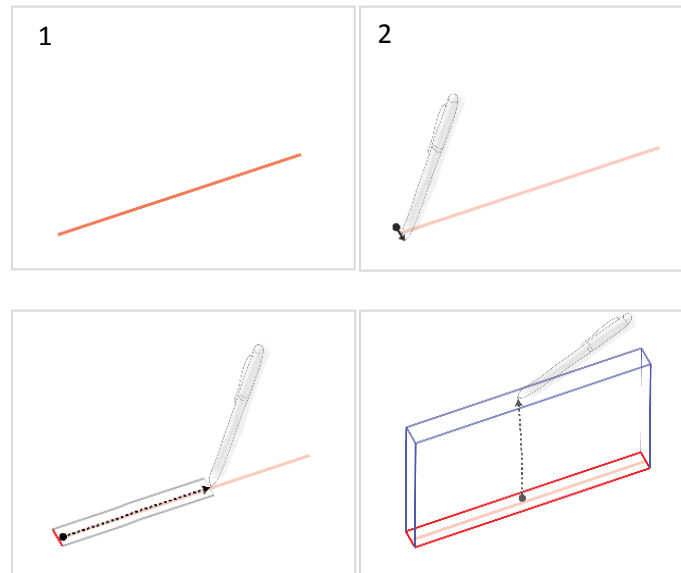


Figure 4.5: The steps to creating a simple wall template with two operations

After he clicks the Plus button again, the system shows a window in which Peter can sketch an icon and specify a wall type for the template shown in Figure 4.6. Finally, the icon appears as a new option on the UI panel on the left side of the SolidSketch interface. Once these steps are completed, he can select that recorded template by clicking on the newly created icon. After clicking on his custom wall button, he can draw a line on the canvas, and the system will automatically populate a custom wall in real time with the width and height dimensions defined in the template composition process.

Once Peter has created this wall template, he can sketch it quickly without the burden of thinking about how to execute a series of commands for specifying starting and ending points, or curvatures of a wall. The system continually generates geometries from Peter’s stylus tip based on the

constraints and parameters captured during the template composition procedure. Figure 4.7 shows how he can trace his concept sketch to implement some of his design intents in three-dimensional spaces. When Peter uses the stylus to draw the wall, the system continuously generates the geometry in real time so that he can adjust his sketch lines to adapt to the emerging nuances of the design.

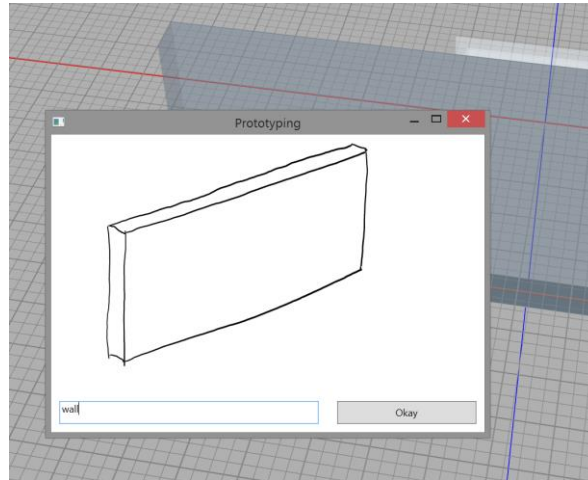


Figure 4.6: Left: After hitting the plus button again, Peter can draw the icon and assign the type of template object they are creating. Right: The system will store the rules and parameters and create an icon on the left panel of the UI.

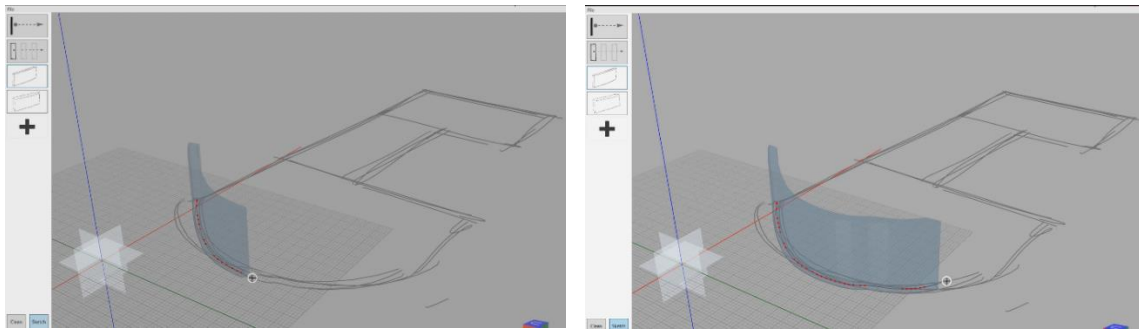


Figure 4.7: The system generates a wall from template in real-time while Peter is moving his stylus on the canvas

4.3 Extending Dimension Tool

When he finds a wall that is not ideal for the surrounding spaces, he can either erase it by using the scratch gesture, as shown in Figure 4.2, and redraw it, or extend the existing sketch lines attached to the wall to modify it quickly. For instance, after a wall is drawn using the wall template, Peter can simply sketch a line near the end points of the existing line that control the wall geometry to modify

this existing wall. While sketching this modification line, the system automatically detects the surrounding contexts and uses the raw sketch points to visualize the geometries in real time, illustrated in Figure 4.8.

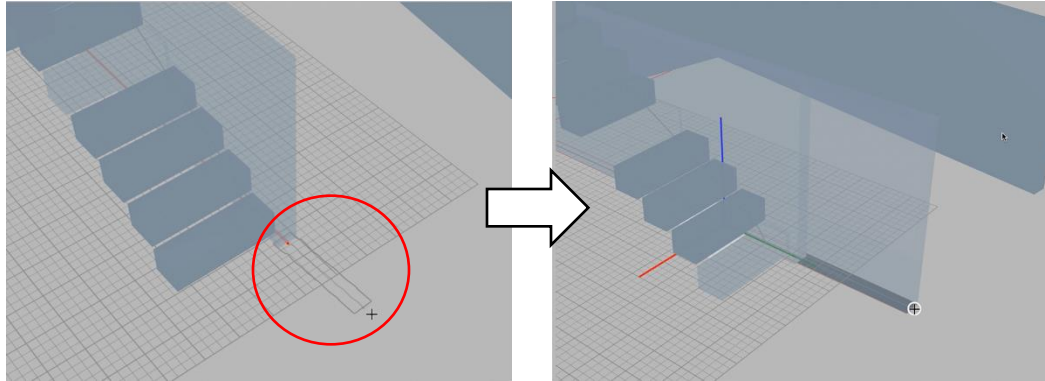


Figure 4.8: Peter sketches a new line at the end of the profile of a wall. The system automatically generates the extended profile and the wall geometry that depends on the profile

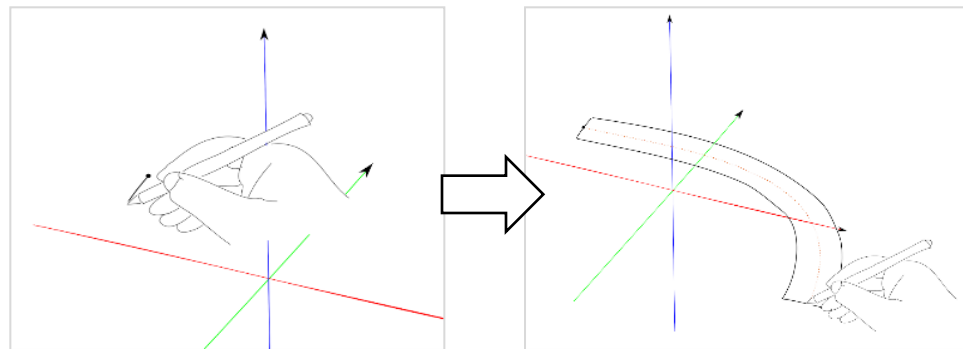


Figure 4.9: Peter extends a one-dimensional line to a 2D surface through a sketch interaction. Left: Peter draws a line segment and selects it. Right: He extends the segment into a 2D profile while holding the stylus button to draw a line in the “extending dimension” tool.

Based on his recent wall creation, he discovers that the staircase and the adjacent walls in between the living room and dining room may cause some circulation problems. He rapidly adjusts his design by redrawing the adjacent walls a few times to find the perfect location and curvature of the wall that can fulfill his design intent. To create a custom wall in between the dining room and kitchen, he draws a short segment to represent the width of the wall, and uses the “extending dimension” tool to create the wall profile, illustrated in Figure 4.9. During the interaction, he can observe the wall profile shape generated by the system in real time while moving the stylus to understand whether this custom wall will affect the surrounding designs. Then, he extrudes the

profile to the desired height to create the wall geometry using the same extending dimension tool. As depicted in Figure 4.10, Peter can also use this tool to create columns in the house after he draws the profile shape of the column. Similar to the previous example, the system also takes real-time input points to generate the solid shapes while Peter is moving the stylus.

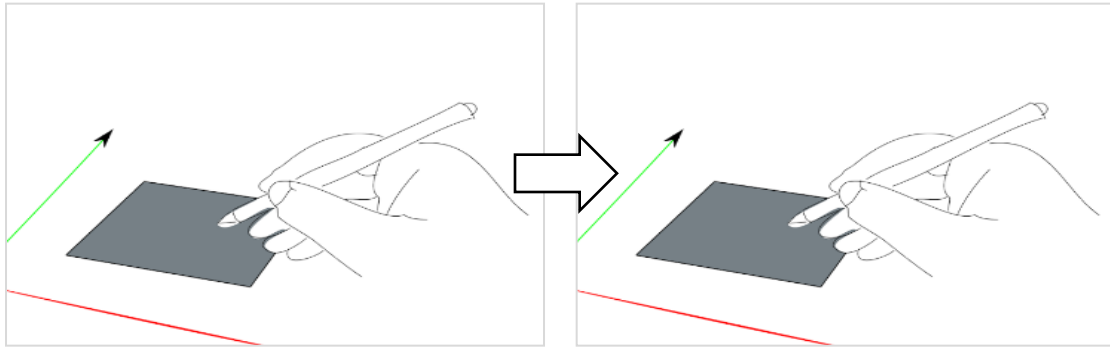


Figure 4.10: Extending a 2D shape to a 3D solid model. Left: Peter draws a 2D profile and selects it. Right: Peter extrudes the 2D profile into a solid shape while holding the stylus button to draw a line in the “extending dimension” tool.

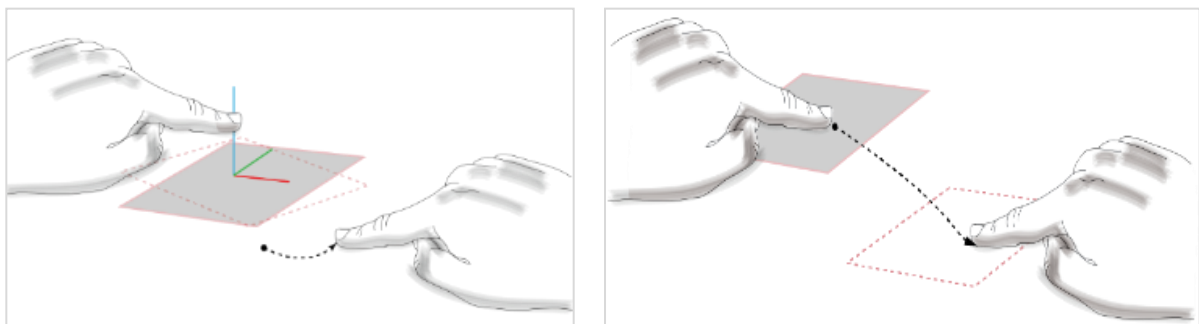


Figure 4.11: Basic multi-touch gesture in SolidSketch. Left: Peter employs multi-touch to rotate a selected object. Right: Peter moves and copies the existing shapes with two fingers.

4.4 Manipulating Existing Geometry with Multi-Touch Interactions

Peter can use multi-touch interactions to adjust the location and orientation of existing walls. By touching a shape, he “selects” it, activating the assistant axis widget, shown in Figure 4.11-left. Then, he can either drag the geometry to move it, or manipulate the axis widget to rotate the geometry. In Figure 4.11-left, Peter uses his left index finger to hold the Z axis and right index finger to drag and rotate around the Z axis. The user can also rotate around the X or Y axis in the same way while holding the X or Y axis, respectively. To move the shape only in the X, Y, or Z direction, Peter can

touch on one of the axes and drag it in the desired direction. While holding the target shape with one finger, Peter can drag with another to instantly copy the geometry and move it to a desired location in one action, as depicted in Figure 4.11-right.

4.5 Array Creation Tool

After laying out the locations of the walls, Peter then tries to create a staircase. He starts from creating the first step of the staircase. To create a precise rectangle, he uses the right angle gestures (as shown in Figure 4.2) on the corners of the profile shape he just drew. The system responds with animation to adjust the corners to 90 degrees. Similarly, he can also use a tick gesture to set two sketch edges to the same length. When he wishes to connect two edges to create a corner, he can perform a lasso gesture. After finishing the profile of the step, he extrudes it upwards to create the solid geometry for the step. Once the geometry of the first step is created, he then sketches a path on the wall while the array prototype is selected. The system generates a series of shapes along with the specified path. Figure 4.12 shows a similar case in which Peter draws a curved path for the system to create a series of shapes in real time while he is moving the stylus.

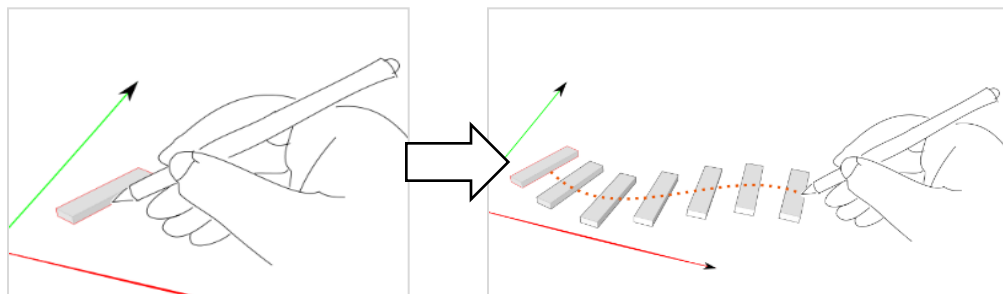


Figure 4.12: Creating an array from a shape. Left: Peter creates a shape and selects it (either 3D or 2D). Right: Peter selects the shape and draws a path to create an array when the array creation tool is selected.

Since the system knows that the objects are cloned from the selected shape, Peter can easily update the shapes in the array after he modifies the first shape. Because the default distances between the steps are too large, he decides to adjust the interval and number of the steps in the array. To adjust the interval, he touches and moves one of the elements in the array; the elements in

the array will move along with the drawn path. To adjust the number of shapes in the array, he holds the first shape in the array with one finger, and moves another shape with another finger closer to the first shape, as shown in Figure 4.13. The system then adds new shapes when there is sufficient room at the end of the path.

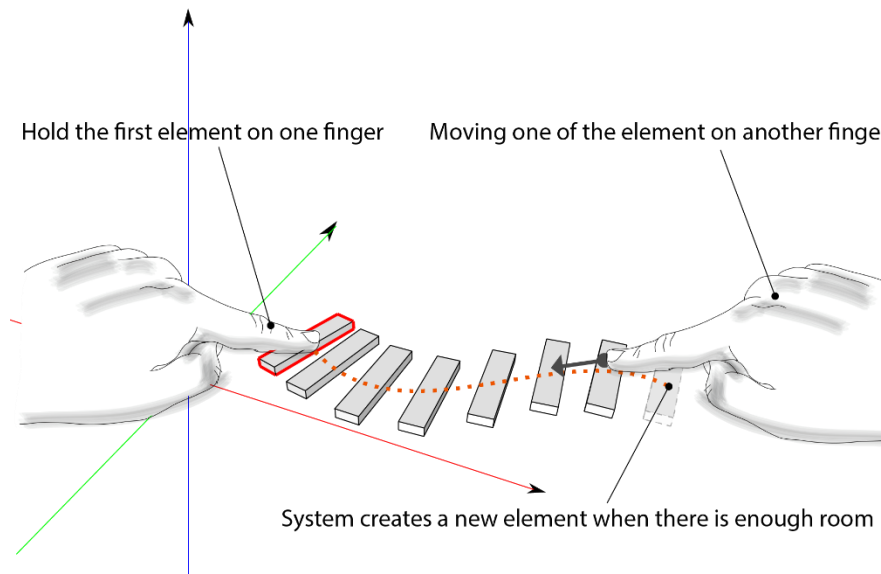


Figure 4.13: Peter adjusts the number of elements on an existing array.

4.6 Editing Through Stylus

After finishing the first floor, Peter uses the “lasso gesture” on top of a wall to move the drawing plane to the second floor, similar to what he was doing during freehand sketch. Creating a free-form polygonal slab is easy: he simply sketches out the free-form polygonal profile for the slab of the second floor. With a few over-sketch lines on the profile, he can adjust it to meet his specific design needs. The system tries to automatically merge the existing shape with the newly sketched lines. Figure 4.14 shows that he can add a curved balcony easily with one simple sketch. Similarly, creating a hole in the slab for the staircase is simple. In Figure 4.15, Peter draws a closed polygon directly inside the slab polygon to create a hole in it. While he is drawing within an existing shape, the system will make the existing shape transparent so that Peter is able to adjust the locations of the creation

lines to correspond to the objects behind it. After he finishes the profile of the slab, he can use the *Extending Profile* tool to extend the profile into a 3D solid shape.

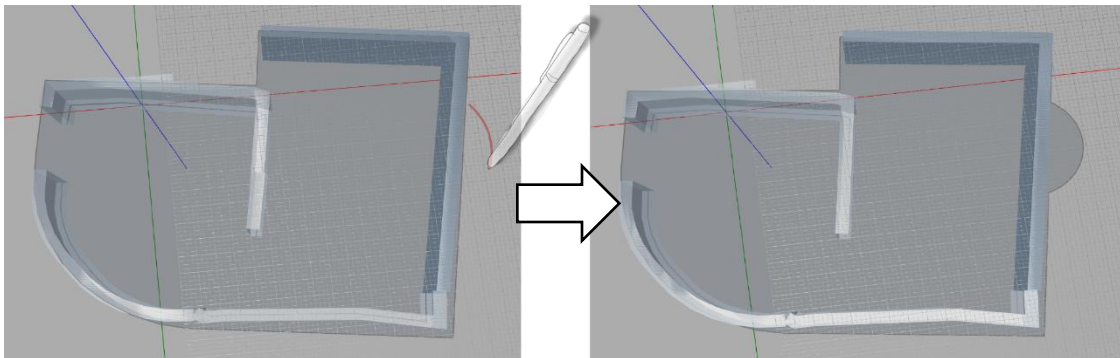


Figure 4.14: By drawing a curved line near to the slab profile, Peter can quickly create a balcony and experiment with its size and shape.

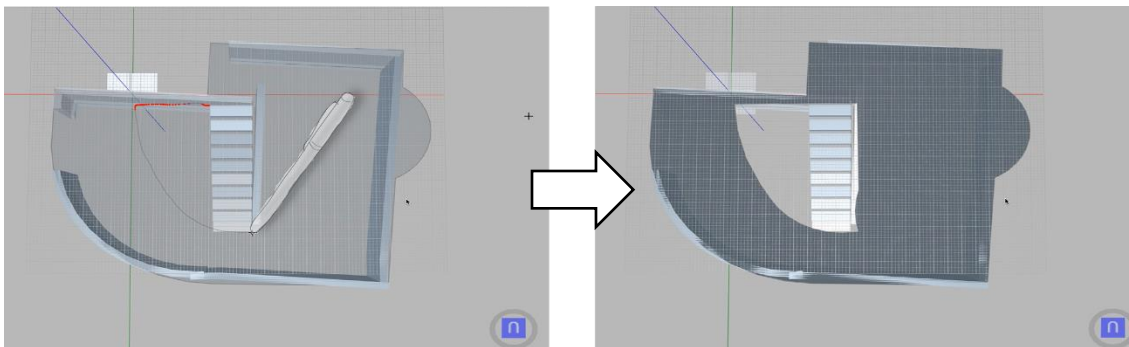


Figure 4.15: By drawing a closed polygon inside of the slab profile, Peter can create a hole for stair access.

4.7 Making Holes and Boolean Subtraction on Solid Shapes

Once the slab is finished, Peter uses another wall template with a different height to draw the walls directly on the slab of the second floor. After finishing the walls, he starts to decorate his design with columns and creates some windows to enhance the view in the main bedroom. Peter can apply two types of subtraction commands in SolidSketch to achieve these goals: (1) subtraction between two 3D shapes, and (2) subtraction between a 2D and a 3D shape. Figure 4.16 A and B show how he can create stylish columns for his design by positioning a solid object inside another and then performing a scratch gesture to delete the inside object, thereby creating a subtraction in the target shape. To create windows, he uses the second method. He first draws the shape directly where he wants it, then selects the wall where he is making the opening for the window, and finally performs a

scratch gesture on the newly created shape to make a hole for the window. Figure 4.16 C and D illustrate 2D and 3D shape subtraction where he draws a 2D shape on a wall and performs a scratch gesture on the 2D shape to create a circular opening in the wall.

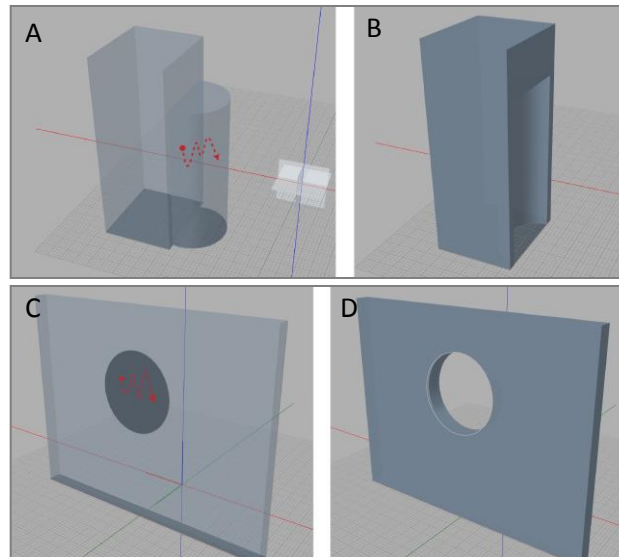


Figure 4.16: Peter can employ a scratch gesture to accomplish Boolean subtraction while selecting a base geometry. (A) The user selects a bigger cube and scratches the smaller cylinder to create the geometry in (B). (C) Similarly, Peter can perform a scratch gesture on a 2D shape attached to a solid object to create a hole in the object (D).

4.8 Transit to Next Stage of Design

Figure 4.17 shows the building that Peter created in 30 minutes. He then exports the model into the IFC format that describes the geometrical information as well as the relationships between geometries and their basic semantics. Later, he uses this exported file in Revit, a famous building information modeling program, to add detailed information, such as materials and detailed properties of the building elements. Since the exported IFC file retains the relationships between the geometries, he can modify the parameters and adjust the design's details quickly in Revit. SolidSketch closes the gap in his workflow during the early stage of his design process. He no longer needs to explore his design ideas and rebuild the entire model on different platforms. Thus, he is able to keep his original design intents throughout the entire design lifecycle.

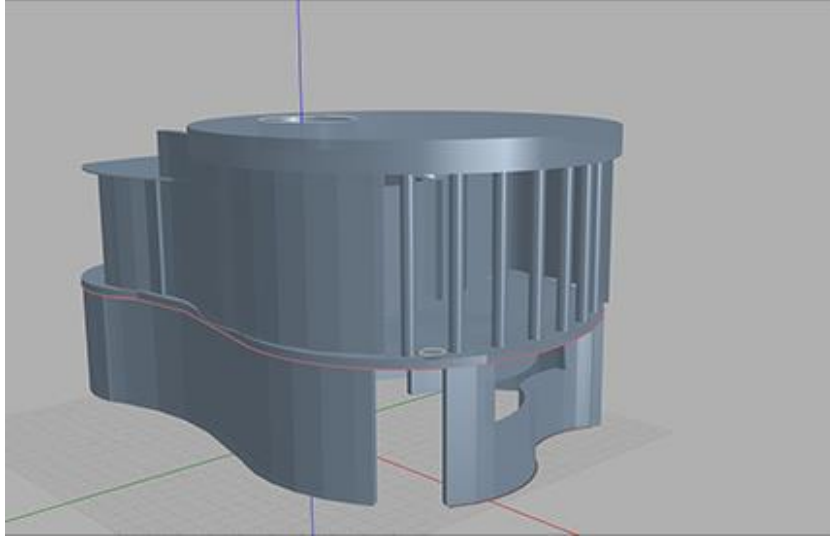


Figure 4.17: The simple house building Peter would create in 30 minutes

4.9 Summary

As shown in the previous example, SolidSketch observes Peter's sketch and multi-touch inputs and reacts in real time based on an interpreted context constructed from the user's intention recognized through continuous interactions. He can use these interactions to create not only parametrical geometries but also the information describing the topology used in the digital design tool. The system takes the continuous finger and stylus inputs, and subsequently transforms those data into meaningful information that is continuously represented to Peter for the next continuous interactions. Through these interactions on a tablet, the system minimizes the burden of executing a series of commands during the design process.

Figure 4.1 illustrates the system, which consists of a side panel with several buttons on the left side of the screen and a main canvas on which Peter uses his fingers and a stylus pen to interact and create geometries with parametric information. When using the system to create the model, he rarely had to click on the buttons in the side panel. The stylus enables detailed controls within the sketch-based interactions. Additionally, multi-touch interactions facilitate rapid, hands-on manipulation of system commands such as those related to basic navigation (e.g., pan, orbit, and

zoom) and those related to object manipulations (e.g., move and rotate shapes). By analyzing the source of the input data (i.e., pen vs. fingers) and the surrounding context (e.g., nearby geometrical relations), the system recognizes diverse intentions from the sketch and multi-touch interactions. Peter was able to sketch lines naturally for the design drawing and shape creations with stylus gestures and multi-touch gestures to assign and modify geometries and their relationships directly, without having to think about which buttons or commands to execute at the time. To maintain the flow of interactions, Peter can sketch on top of existing lines or shapes to modify the original geometry. As sketch-based refinements are created, the system propagates changes throughout the model in real time based on the sketched revisions. In the next chapter, we will explain the details of how these interactions are implemented in SolidSketch.

Chapter V. System Details

In the previous chapter, we described a scenario in which Peter used SolidSketch as a tool for the early stage in his design process. By using the system, he was able to smoothly create a house and export it as a building information model for the next stages of the design. To achieve this, the system needs to recognize and respond to interactions in real time while he is moving his fingers or stylus on the SolidSketch canvas of the tablet. The system also needs to map these interaction inputs into the internal data structure so that it can export the data into a format that can be read by other programs. In this chapter, we will introduce the system architecture and examine the implementation of SolidSketch that fulfills both online (real-time) and offline recognitions and responses. In the next chapter, we will focus on the data mapping processes. This chapter comprises four main topics: (1) system architecture, (2) online recognition and response, (3) offline sketch disambiguation, and (4) mapping recognition results to a semantic model.

5.1 System Architecture

The SolidSketch system is built on top of Windows Presentation Foundation (WPF) and is mostly written in C# and C++. Figure 5.1 shows the overall system architecture that consists of five layers: (1) Interface (light blue boxes), (2) View (yellow boxes), (3) General Controller (blue box), (4) Parametric Model (deep blue box), and (5) Geometry Engine (brown box). The arrows between the boxes represent the information flow. For instance, the Interface layer sends input information to the General Controller to analyze the user's intentions. The General Controller converts the input information to temporary geometries during the user's interactions and visualizes it on the View layer directly. It also stores this information into the Parametric Model layer after the interaction session is over. The 2D and 3D representations, e.g. triangle meshes, in the view layer are bound to the Parametric Model to enable real-time responses when the user is changing the parameters in the

Parametric Model. Except in the Geometry Engine, we implemented the entire system to achieve online/offline recognitions, real-time geometry generations, and then to create a parametric model according to the user's interactions. When a user requests to export IFC models for other programs to use, the system will map the data stored in the Parametric Model layer and create an IFC model, which will be described in detail in the next chapter.

As shown in the light blue boxes on the top left portion of Figure 5.1, we have three different forms of input in the Interface layer: (1) UI button clicks, (2) multi-touch inputs, and (3) stylus inputs. To reduce the burden of switching between modes, such as the actions required in traditional CAD tools through clicking on UI buttons, we only implemented a minimal number of UI elements in the system, including two preset prototype buttons (extending dimension and creating array) for creating 3D geometries, and a few utility buttons for undo/redo purposes. Indicated by the arrows pointing down from the boxes in the Interface layer, most commands are continuously analyzed by the General Controller layer. To produce the most immediate responses, the movements of the stylus on the canvas will directly generate real-time inking on the canvas. In addition, these stylus points will be analyzed in the General Controller and Parametric Model for generating more complicated responses in real time. After the stylus is lifted up from the drawing canvas, the system will remove the temporary ink and replace it by off-line recognition results on the canvas.

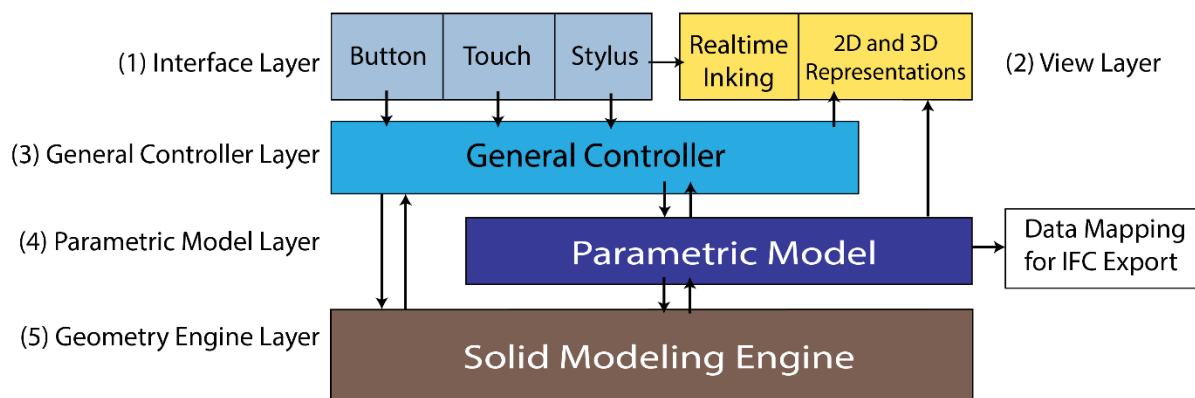


Figure 5.1: The overall system architecture for SolidSketch

The General Controller contains the main interaction logic for recognizing the user's intentions in real time (online) as well as immediately after they finish an action (off-line). Figure 5.2 shows the overall pipeline for detecting a user's intentions in the General Controller. The first step in this pipeline is real-time processing. When the user's fingers or stylus touch the canvas, the system starts to process that input information and produce responses continuously based on the surrounding information, e.g. parametric models and geometries. If users select a prototype template for drawing, such as the wall template that Peter used in the previous chapter, the system will use those parameters and constraints for generating solid models from the Solid Modeling Engine on the locations and orientations recognized based on this surrounding information.

The second step in this process is to disambiguate the user's intention. After the stylus or finger lift-up events are detected, the system starts to analyze the data of this interaction session in order to understand their intentions, which will be used for determining whether it needs to execute commands, or create or edit geometries in post-processing. In the following sections, we will discuss how the system distinguishes commands and creates geometries based on different contexts.

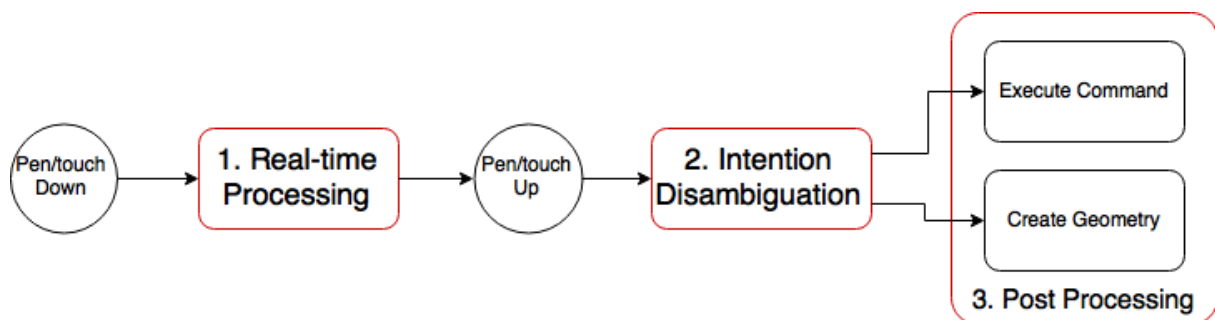


Figure 5.2: The overall sketch/multi-touch analysis pipeline for detecting intentions to execute commands or related actions

5.2 The General Controller Layer

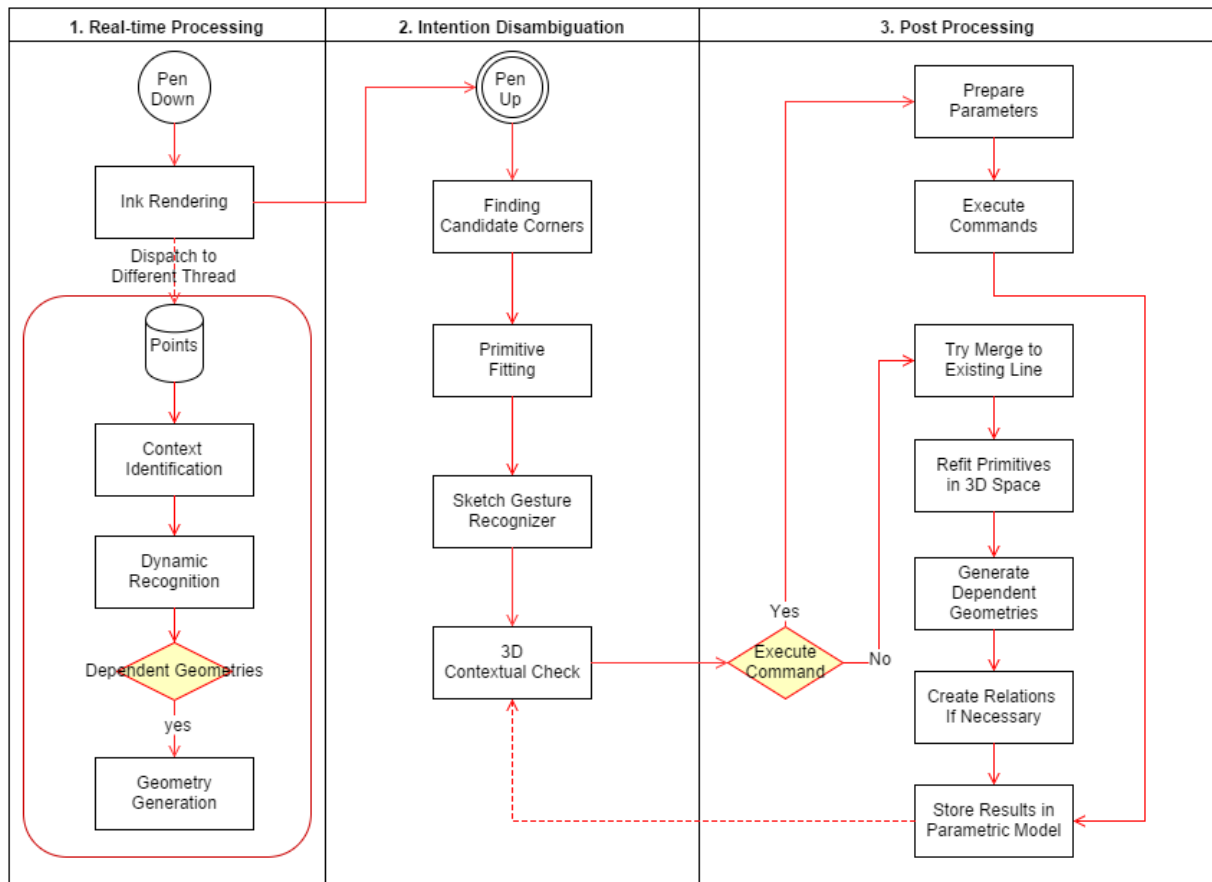


Figure 5.3: The detail sketch recognition process for SolidSketch

As shown in Figure 5.2, the General Controller has three main parts: (1) real-time processing, (2) intention disambiguation, and (3) post-processing. Figure 5.3 shows a detailed process in each part of the sketch input. Finger touch inputs share a similar pipeline as Figure 5.3. However, the pipeline of touch inputs is much simpler, as the system ignores the first three blocks in the intention disambiguation section for finding the corners, primitive fitting, and recognizing the stylus gestures. Nevertheless, the system does need help from a 3D contextual check in order to identify the three-dimensional context of the objects the user is touching. In this checking procedure, the system checks all possible touched objects and the relationships among other objects. In this chapter, we will first describe the real-time processing and intention disambiguation, and then explain what

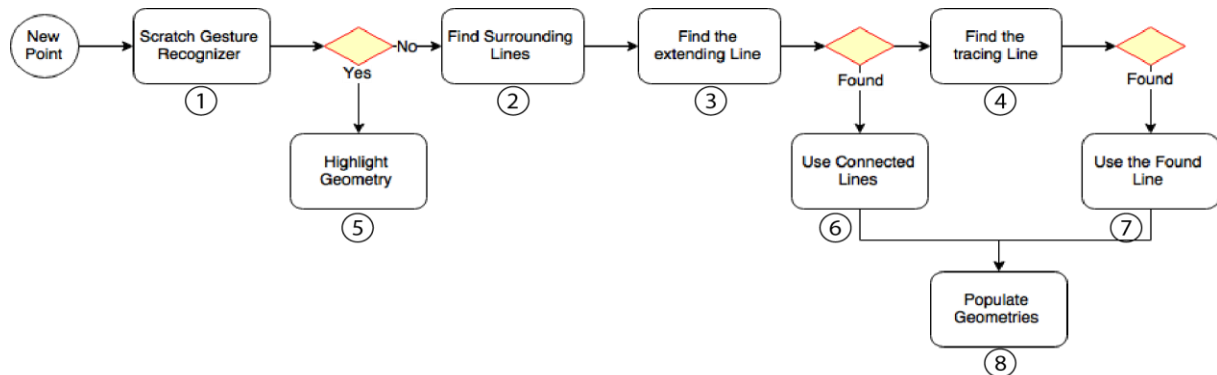
commands to execute from the analysis of intention disambiguation. The geometry generation process will be discussed in the next chapter relating to parametric objects.

5.3 Real-time Processing

The first part of Figure 5.3 illustrates the real-time processing that occurs after users place the stylus on the canvas in the program. This process includes: (1) real-time ink rendering; (2) context identification; (3) dynamic recognition; and (4) temporary geometry generation. The purpose of real-time processing is to generate continuous and embodied feedback during interactions. Ink rendering only requires minimal computational resources to reflect the user's actions in real time. Thus, as mentioned, the system directly renders the temporary ink while the user is moving the stylus without any recognitions. While the user is moving his stylus, the system is rapidly generating too many 2D raw points that would slow down the responsiveness. To avoid the problem, this process reduces the sampling rate and stores it in a queue to be dispatched to a different thread when generating geometries in real time.

In this real-time processing thread, indicated in the red box of the part 1 in Figure 5.3, the system first identifies the contexts and collects the related parameters for dynamic recognition and geometry generation. In real-time processing, the contextual information includes the following three types of information: (1) location, (2) lines around stylus points, and (3) the template that is selected. As mentioned in the previous chapter, the system provides two possible drawing locations: (1) on a drawing grid plane, and (2) on a surface of a solid shape. In this real-time processing, the system simply gets the surface, $S = \min(\text{Distance}(\text{Pt}, \text{Si}))$, where Pt is the point dequeuing from the stored 2D raw points projected on the view camera in the 3D space, and Si indicates the candidate drawing surfaces in the view. Next, the system checks whether there are any existing lines near the starting point of the current sketch stroke on the drawing surface. Finally, if the user selects a

template, e.g. wall template, in the UI, it will also be taken into consideration for the next step in real-time processing – dynamic recognition. The pseudocode in Figure 5.4 shows the logic of the dynamic recognition.



```

1  # received a new point in every five raw point
2  Function NewPointReceived (newpoint):
3      firstPoint = rawPoints.first()
4      # add the new point to a global variable
5      rawPoints.push(newpoint)
6
7      # using $1 Recognizer with trained scratch gestures
8      recResult = Recognizer(rawPoints)
9
10     if recResult is ScratchGesture
11         highlight_hittingGeometry()
12     else
13         # find all the possible lines
14         surroundingLines = FindSurroundingLines(rawPoints)
15         # find the existing line where one of the end point is close to
16         # the first point of the drawing line
17         {mindist, line} = min(distance(firstPoint, surroundingLines))
18
19         if mindist < m # where m is an arbitrary preset distance
20             existingline = line
21         else
22             existingline = FindTracingLine(surroundingLines)
23
24         if existingline is not NULL
25             # This method takes the parameters and constraints of
26             # the template that links to the existing line, and
27             # populate the temporary geometries from the new raw points
28             PopulateDependentGeometries(existingline, rawPoints)
  
```

Figure 5.4: The logic flow (top) and the pseudo code (bottom) for dynamic recognition

In Figure 5.4, the process of “dynamic recognition” determines that the user’s input is either: (1) scratching gestures, (2) extending from an existing line, or (3) tracing an existing line. Instead of recognizing all of the stylus gestures mentioned in Figure 4.2 in the real-time processing step, the

system only tries to recognize whether the user is performing the scratch gesture or not. This enables the system to decide whether it should highlight the geometry that is about to be deleted in real time. To speed up the real-time gesture recognition process, we employed \$1 Recognizer (Wobbrock, Wilson, & Li, 2007) and trained it with preset scratching gesture samples collected from two researchers, shown in Figure 5.5, in the scratch gesture recognizer.

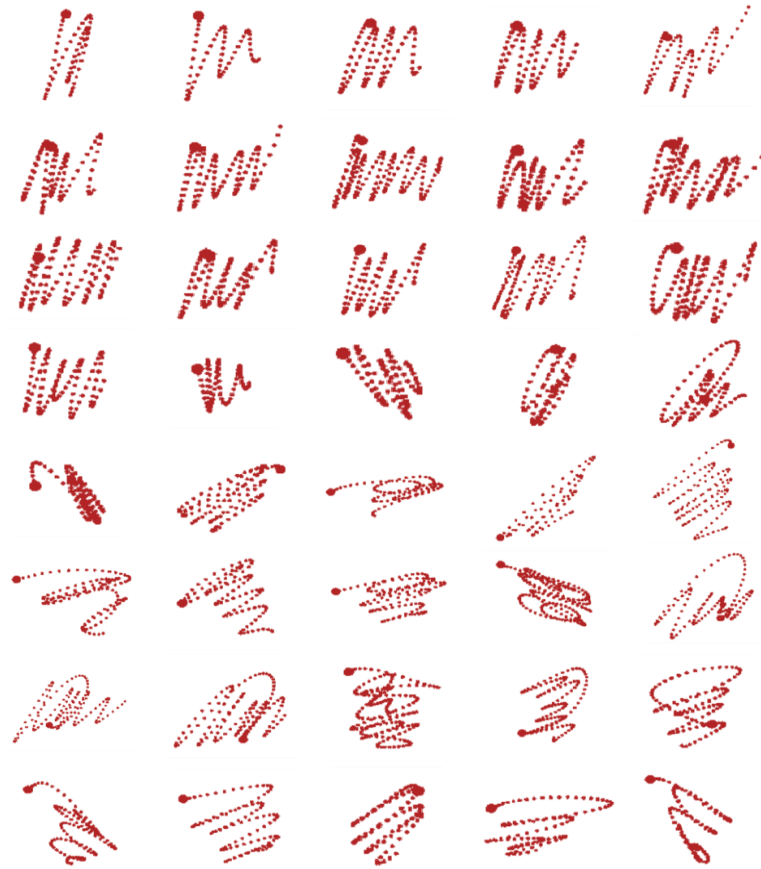


Figure 5.5. Preset scratch gesture samples

If the system assesses that the current sketch stroke is not a scratch gesture, it starts to identify the surrounding lines that are on the same surface on which the user is currently drawing. Since the system maintains the parametric relationships in the Parametric Modeling layer, it is able to determine the possible candidate lines through the surface identified in the “context identification” box described previously. In the top Figure 5.4, the “find the extending line” box shows how the

system finds that the starting point of the current sketching line is close to an ending point of an existing line. In that case, the system will extend that existing line and populate the dependent geometries based on the stored parameters and constraints, such as the example shown in Figure 4.8. When the system detects that the user is trying to trace an existing line, it will walk through the same process as “find the extending line” procedure. To populate these dependent geometries, the system first needs to run a weighted topological sort on all of the dependent tasks and determine the order for computing the triangle meshes of different geometries in these tasks. The details of weighted topological sorting will be explained in Chapter 6.5.2. Once a task is completed, the system will send the meshes to 3D space, render them, and provide the geometrical information as an input to the next task for further geometry generation. However, it is possible that not every geometry can be rendered in real time with complicated tasks. Since the geometry generation is done in a separate thread, users can at least see some geometries generated on their pen tips while moving them, which helps to adjust their design decisions on the fly. We will discuss the algorithm and the process of how the system schedules the tasks in detail in the next chapter.

5.4 Sketch Segmentations and Primitive Fitting

As shown in the second part of Figure 5.3, the intention disambiguation consists of four processes: (1) identifying candidate corners, (2) primitive fitting, (3) sketch gesture recognizer, and (4) 3D contextual check. Since finding candidate corners and primitive fitting are the fundamental steps for understanding the users’ sketch intentions, we will start the discussion here.

An accurate sketch segmentation algorithm can enable the system to recognize the precise sketch lines from freehand sketches. Sketch segmentation and primitive fitting remain topics of active research. Considerable progress has recently led to the development of reliable methods (Baran, Lehtinen, & Popović, 2010; Pu & Gur, 2009). Pu and Gur proposed a greedy algorithm for

precisely identifying a minimum set of segmentations with the help of radial basis functions (Pu & Gur, 2009). After comparing the performances and the flexibility of adjusting different parameters for granularities of curve fitting, we adopt and modify the line segmentation and primitive fitting algorithms developed by Baran et al. (2010). In our adaptation of this algorithm, it first identifies the candidate corners from a sketched input stroke. Since there are usually too many raw points captured by the device, we carefully down sample the number of points to accelerate the processing speed. To find the candidate corners, every five points are selected in these sample points from the previous step to check if they can be fitted into one or two line segments. If there is a high confidence of fitting these five points into two line segments, the system selects one of the points as a candidate corner. For instance, Figure 5.6-left shows that the set of five points is not able to be fitted into one line segment, while another set of five points can be fitted into one line segment shown in Figure 5.6-right. Thus, in the example of Figure 5.6-left, the system selects the third point as a candidate corner.

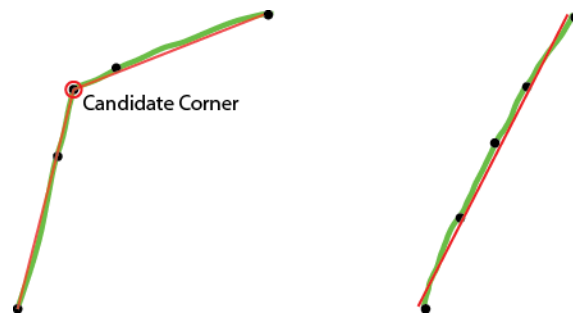


Figure 5.6: Green lines are the original stroke. Red lines are the pre-fit segments. Black dots are the sampled raw points. The line fitting algorithm is not able to fit the stroke into one line segment on the left while it has high confidence of fitting the stroke into one line segment on the right.

Then, it fits the target stroke to multiple segments by using the points in between any possible combination of the candidate corners. Each segment will be treated as a node and assigned a weight using the distance between the original raw points and the segment. In this process, the segments with weights that exceed the preset tolerance are discarded. Connecting all of the remaining nodes

together will form a graph used to identify the shortest path from the starting node to the ending node for the primitive fitting results.

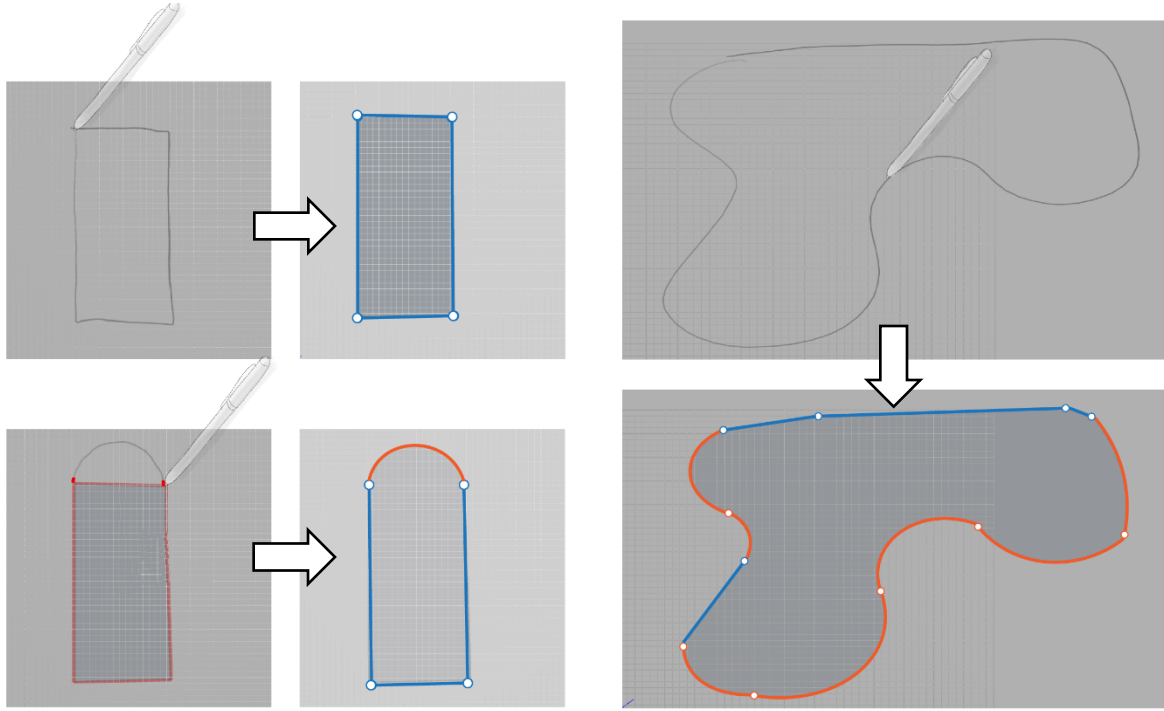


Figure 5.7: The results of sketch segmentation and primitive fitting. Blue segments are straight line segments. Orange segments are arc segments.

As shown in Figure 5.7, in our use case, the system only has to fit the target stroke into arcs and straight lines. There is a well-known procedure to fit raw points into a straight line in between two candidate points, as shown below:

$$\bar{X} = \frac{\sum_{k=i}^j x_k}{j-i} \quad \text{and} \quad \bar{Y} = \frac{\sum_{k=i}^j y_k}{j-i} \quad (5.1)$$

$$m = \frac{\sum_{k=i}^j (X_k - \bar{X})(Y_k - \bar{Y})}{\sum_{k=i}^j (X_k - \bar{X})^2} \quad (5.2)$$

where i is the index of the current starting candidate point, and j is the index of the current ending candidate point. \bar{X} and \bar{Y} are the average values of all of the X and Y values in this segment, respectively, and m is the average direction of this segment, which will be the slope of the current fitted line. The value m is calculated by summing the products of the delta of the current X and Y

values and their average values, and then dividing by the sum of the squares of the delta between the X value and its average value. To get this line segment, the system selects the candidate starting point and ending point, and then projects them to this fitted line. For arcs, the method provided by Baran et al. is applied (Baran, Lehtinen, & Popović, 2010) by lifting the points to a paraboloid in 3D space, and finding the ellipse that determines the circle in 2D space that can fit to those points.

Figure 5.8 describes the logic of how the system identifies candidate corners and fits the raw points into primitives.

```

1  Function StrokeSegmentation (raw_Points):
2      # candidate_corners has the list of the points
3      # that's possible to be corners
4      candidate_corners = FindCandidateCorners(rawpoints)
5      # candidate segment will contain an array of points
6      # in between the candidate corners
7      candidate_segment = []
8
9      # a corner has a pointer to the possible segments it connects to
10     for corner in candidate_corners
11         current_index = raw_Points.indexOf(corner)
12
13         # it tries to take every possible corner combinations and fit them into
14         # lines or arcs
15         while current_index < raw_Points.length
16             point = raw_Points[current_index]
17             if candidate_corners contains point and candidate_segment is not empty
18                 candidate_segment.push(point)
19                 # FitPointsIntoLine and FitPointsIntoArc will create line/arc
20                 # with the max distance between the raw points and the line/arc
21                 # and then build the node-link structure between candidate
22                 # corners and the fitted line/arc
23                 FitPointsIntoLine(candidate_segment)
24                 FitPointsIntoArc(candidate_segment)
25             else
26                 candidate_segment.push(point)
27
28             current_index++
29
30     # take the first corner from the candidate_corners
31     # and do a graph search with the max distance between
32     # line/arc and the raw points as the weight
33     # to find the shortest path to the end of the corners
34     # (e.g. Dijkstra Algorithm)
35     first_corner = candidate_corners.first()
36     return FindShortestPath(first_corner)

```

Figure 5.8: The pseudo code for finding candidate corners and primitive fitting

5.5 Symbolic Gesture Recognition

The system does not rely on a specialized gesture recognition engine or machine-learning approach for the final decision of whether the drawing stroke is a gesture. Instead, the system takes the results from line segmentations and determines the sketch gestures by checking the compositions of these segmented lines to help constrain the gestures in the recognition procedure. This algorithm recognizes the gesture with the highest confidence first and the gestures with lower confidences later. To the best of our knowledge, we have not previously seen a sketch gesture recognition method using this approach, which has yielded sufficient results in terms of accuracy and efficiency.

```
1  # Primitives are the results from sketch segmentation. It only contains straight line segment and arc segment
2  # Stylus speed is the speed of moving the stylus by the users
3  Function RecognizeSketchGesture(primitives, stylusSpeed):
4      # get the bounding box of this primitives
5      boundingBox = GetBoundingBox(primitives)
6      isSmallGesture = boundingBox.width < SmallGestureThreshold and boundingBox.height < SmallGestureThreshold
7      distance = GetDistance(primitives.firstPoint, primitives.lastPoint)
8
9      if count of primitives is 1 and count of raw points in first primitive < 3
10         return Dot Gesture
11     else if count of primitives is 1 and isSmallGesture and first segment of primitives is Straight Line
12         return Tick Gesture
13     else if count of primitives is 2 and isSmallGesture and distance > ClosingStrokeThreshold
14         return Right Angle Gesture
15     else if count of primitives is greater than 2 and isSmallGesture and distance < ClosingStrokeThreshold
16         return Lasso Gesture
17     else if count of primitives is greater than 4 and stylusSpeed > ScratchGestureSpeedThreshold
18         return Scratch Gesture
19     else
20         return Drawing Stroke
```

Figure 5.9: The pseudo code for recognizing sketch gestures

As shown in Figure 5.9, the procedure for categorizing the segmented stroke into sketch gestures is as follows: (1) dot gesture, (2) tick gesture, (3) right angle gesture, (4) lasso gesture, and (5) scratch gesture. If the system is not able to categorize the stroke into the above gestures, it will then create a sketch stroke for it. For instance, if the stroke captured by the canvas only contains a few raw points ($n < 3$) within a small range, the system will classify this stroke as a dot gesture. When the line segmentation routine returns just one small line segment, the system can safely assume with a high probability that the user was attempting to perform a tick gesture. If two small line segments have at

least one straight line, the system will categorize the sketch stroke as a right angle gesture. In addition, when there are at least two small arc segments with a short distance between the starting and ending points, the stroke will be recognized as a lasso gesture. For the scratch gesture, even if the system has marked the current strokes as a scratch gesture in the real-time processing step, it still inputs the raw points into this sketch recognition engine to verify the result. In other words, the system requires more information to verify whether the user has performed a gesture. In this case, if a stroke with more than four segments is drawn with the stylus moving faster than a certain threshold, it will be recognized as a scratch gesture. Figure 5.10 illustrates the example strokes for the five sketch gestures, the segmentation results from the algorithms we described previously, and the corresponding recognition rules for each gesture.





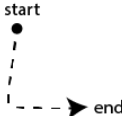

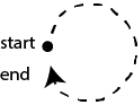

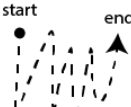

Sketch Stroke	Segmentation Result	Recognition Rule
		Dot: Few raw points within a small segment
		Tick: One small line segment
		Right Angle: Two small segments (at least one line segment)
		Lasso: At least two small arc segments with the distance of the end points in a threshold
		Scratch: More than four segments (large or small) within a time threshold

Figure 5.10: The example sketch strokes, the segmentation results, and the rules for each gesture. In the segmentation result column, the blue segments represent the straight line while the orange line segments represent the arcs.

5.6 Checking the Contexts

In SolidSketch, the system takes different contextual information into consideration to disambiguate the user's intention. This information includes: (1) three-dimensional spatial relationships, (2) parametrical relationships, and (3) what prototype template the user has selected. This information is used in online recognition, offline gesture recognition, and sketch plane decision. In the real-time processing section, we described how the contextual information can assist in determining whether the user is tracing an existing line, extending an existing line, or simply creating a new sketch line. In real-time processing, the system takes the plane the user is drawing on and searches for possible existing lines as the main resource for contextual information. In this section, we will discuss how the system further filters the sketch information in 3D to decide whether the user is performing a gesture on a surface of an object or the grid plane, or simply creating new geometries with a new sketching line.

The algorithm that infers the user's intent is a preset decision tree. Figure 5.11 shows the specific information the system needs for deciding what commands it should execute. At this point in the sketch disambiguation process, the system already has the results of sketch gesture recognition, e.g. whether the user is likely performing one of the five possible sketch gestures. However, the system does not know the exact location at which the user is drawing in 3D space and what constraints from the objects it needs to take into consideration. Thus, the system uses the results of sketch recognitions (green boxes in Figure 5.11), object types (gray boxes in Figure 5.11), and additional contextual information, including pre-selection, parametric relationships, and constraints (blue boxes in Figure 5.10) to decide the user's intentions for executing commands (orange boxes in Figure 5.11).

Similar to the real-time processing method, the system tries to gather all the possible objects involved in this interaction at the beginning of checking the context. The potential objects and locations are shown as gray boxes in Figure 5.11. Since the system enables the potential of

interacting with the objects behind solid shapes through visualizing them transparently, the system needs to consider all of the objects and locations that are hit by the ray shooting from the 2D raw points on the screen in the 3D space.

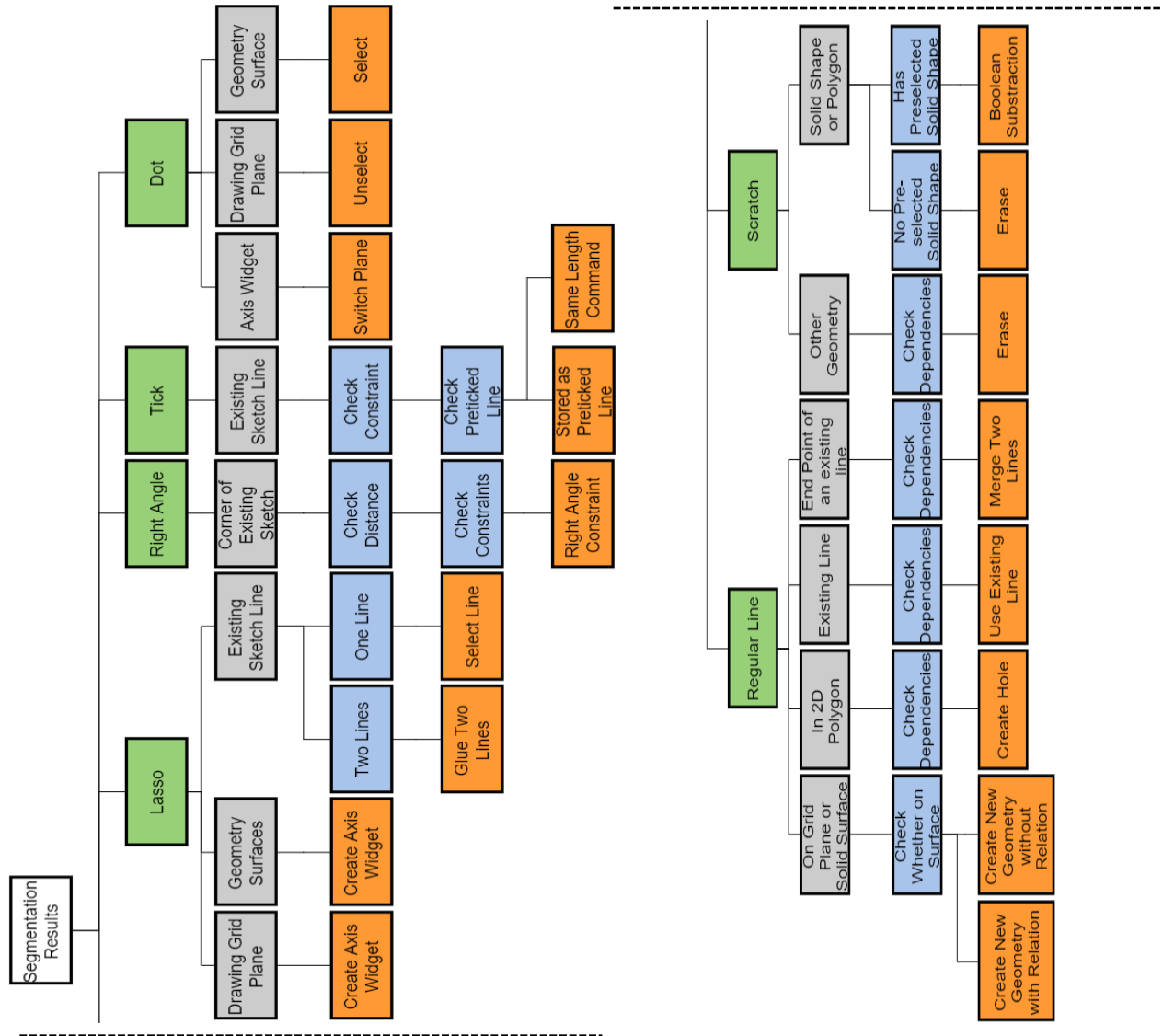


Figure 5.11: The interaction and contextual information used in the procedure of checking the contexts. Green Boxes: the recognition results from the previous steps, including the potential gestures and segmented primitives. Gray Boxes: the possible objects that are hit by the ray shooting from the raw points of the new stroke on the screen coordinate system into the 3D space. Blue Boxes: additional information that needs to be taken into account, including geometrical dependencies, preselected objects, and previous objects that are performed in the same gesture. Orange Boxes: commands executed by the system.

For instance, when performing the right angle gesture, the system will first search for probable locations intersected by the orange dashed line in Figure 5.12-right, including (1) 2D polygon shapes

on the solid geometry, (2) surface of the solid geometry, and (3) the base profile for the solid geometry. According to Figure 5.11, we know that the right angle gesture can only be performed on existing lines, which filters out the case (2). The system then checks the distances between the location of the gesture and the potential target in the 2D coordinates on the hit surfaces and selects the one with the minimum distance. Finally, it checks whether the constraints on the targeted corner allow the new right angle constraint to be set. These constraints are considered as additional contexts as shown in blue boxes in Figure 5.11. If the condition is true, then the system will add the right angle constraint and animate the shape with related geometries to fit the new constraint.

If the system is not able to classify the sketch gestures into commands with the contextual information, it will create a regular line, instead. This entire process will check not only gestures but also the new sketch lines to determine if any modifications need to be made to the existing sketch lines. If modifications must be made, it will use the algorithm described in the following chapter to modify the existing lines and populate the changes to the dependent geometries.

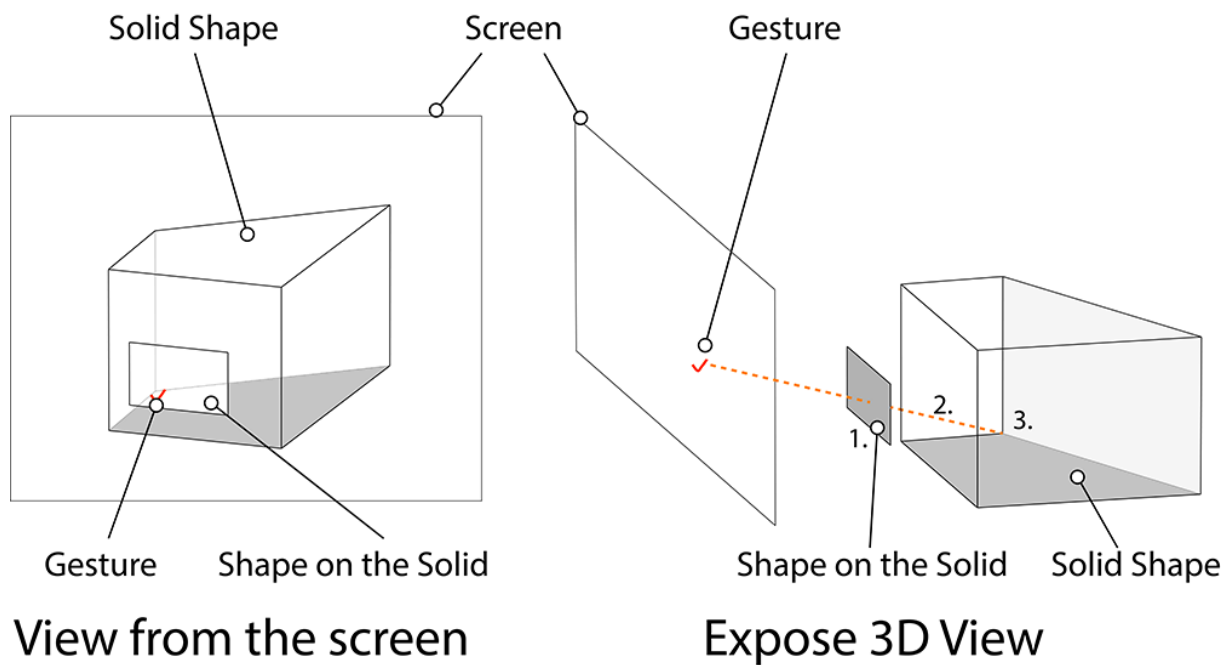


Figure 5.12: Performing a right-angle gesture on an edge of the base shape behind a solid shape. The system needs to go through each possible object to determine whether the sketched gesture applies to this particular screen location.

Chapter VI. Data Mapping and Internal Data Structure

In Chapter 2, we introduced Concept Design BIM 2010 (CDB10) that is used for exchanging design data among different programs in the early stage of the design process. As mentioned, to generate these data, users need to work with complicated UI settings and command execution procedures in traditional CAD programs. Instead of using the conventional method of generating this type of data, we described a scenario in Chapter 4 in which Peter fluidly used the stylus and multi-touch to create and explore designs. In Chapter 5, we explained the system architecture and how it infers the user's intentions for executing commands in SolidSketch. In this chapter, we will discuss how these commands can be translated to a machine-readable data structure that can then be exported for the next stages in the design process described at the end of Chapter 4. In other words, this chapter will explain the last piece of the General Controller layer – geometry creation – and the data structure we are using in the Parametric Model layer with the data mapping for IFC export.

6.1 The Parametric Objects

In the post-processing component of the General Controller layer, the system takes the results generated from the intention disambiguation process and decides whether it should execute commands or generate geometry. In SolidSketch, there are three types of parametric objects: 1D nodes, 2D lines, and 3D solid shapes. All types of parametric objects are a subclass of a general parametric object. As shown in Figure 6.1, this general parametric object contains the properties that a parametric object needs to have, e.g. basic semantic properties, and constraint relations to other parametric objects. The detail relationships between different parametric objects will be described in the following sections. In general, 3D solid shapes are directly or indirectly controlled by 2D lines. Users can employ the stylus or multi-touch interactions to change the 2D lines. When changes are

made to these lines, the dependent parametric objects will be updated based on the constraint and parametric relations.

We will discuss how parametric 2D lines are structured from sketch strokes and tied to the meshes that are visualized in the viewport. Then, we expand the discussion to the 3D geometries and their relationships in the next sections.

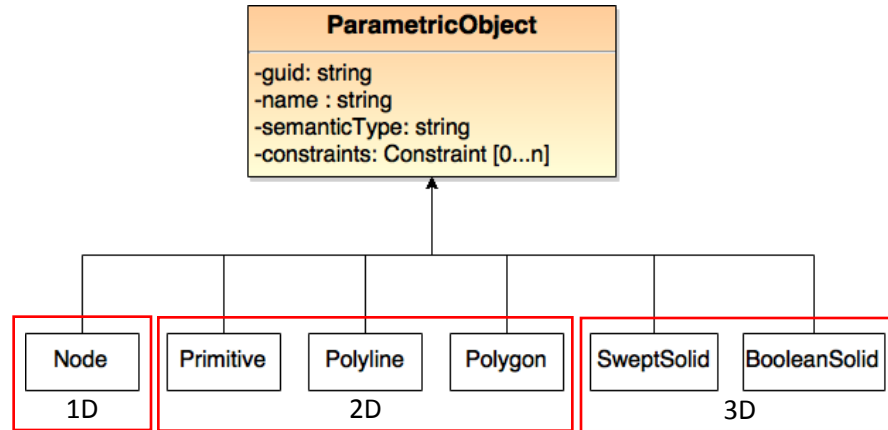


Figure 6.1: The general parametric object and the major parametric object types used in SolidSketch

6.2 Two-Dimensional Geometries

6.2.1 Polyline and Polygon Shape

As discussed in the previous chapter, after sketch recognition is complete, the sketch stroke will contain the 2D segmentation primitives, including lines and arcs, along with the transformation information describing the plane on which the stroke is situated. As shown in Figure 6.2, the system maintains links between this set of straight lines and arcs to the sketch raw points in the *Primitive* object. This set of primitives is linked with nodes for describing the relationships between themselves. The system will decide whether the end points of sketched polylines are close enough to form a polygon shape. To decide whether it should be a closed polygon, it simply calculates whether the Euclidean distance, $D = \text{Distance}(\text{Pt}_{\text{start}}, \text{Pt}_{\text{end}})$ where Pt_{start} and Pt_{end} from the start and end points of the raw stroke points respectively, is smaller than 15 pixels or not. If it is a closed polygon, the system will create a *Polygon* object in addition to the *Polyline* object as shown at the far left of

Figure 6.2. This *Polygon* object takes the responsibilities of generating meshes filling the polygon area, as shown in Figure 6.3-right. As described in Peter’s scenario, he drew a closed polygon inside of a polygon shape to directly create a hole for stair access. This closed polygon is also stored in the *Polygon* object so that it can be taken into consideration for generating meshes with a hole.

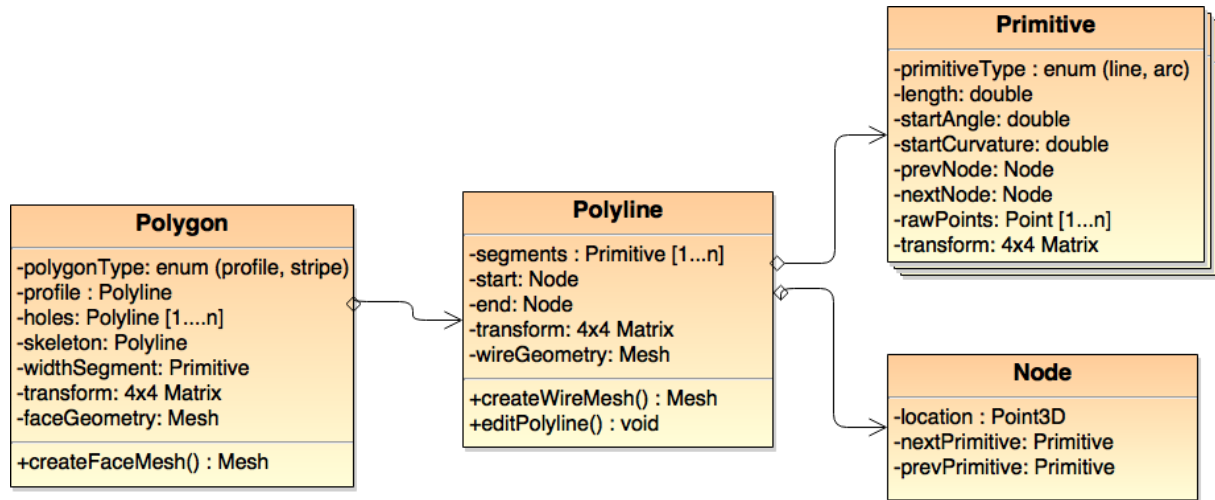


Figure 6.2: The UML diagram for major 2D objects used in SolidSketch

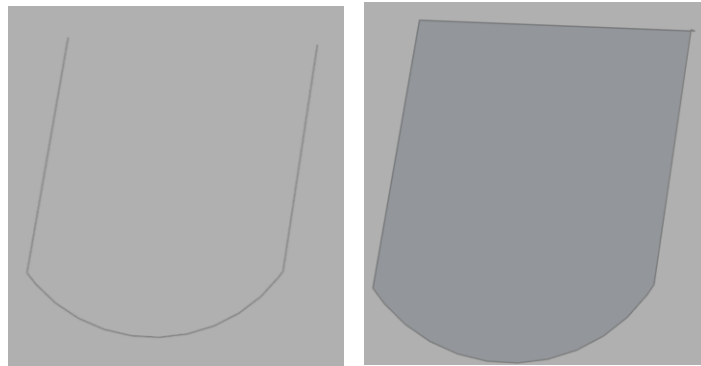


Figure 6.3: Polyline versus Polygon. The system creates meshes to fill up the polygon.

These parametric polyline and polygon objects are bound to the meshes of the actual lines (called *wires* in the system) and shape geometries (*meshes*) represented in the view canvas. When any changes are made to these parametric objects, such as changing the dimensions and line editing, the system will notify the parametric object to update the geometries immediately. Also, the system always associates polyline objects to their closed polygon objects, if there are any, to maintain this

connection for future modifications. Every update from the new sketch strokes will cascade changes in the related polyline and polygon objects immediately.

6.2.2 Handling Sketch Editing

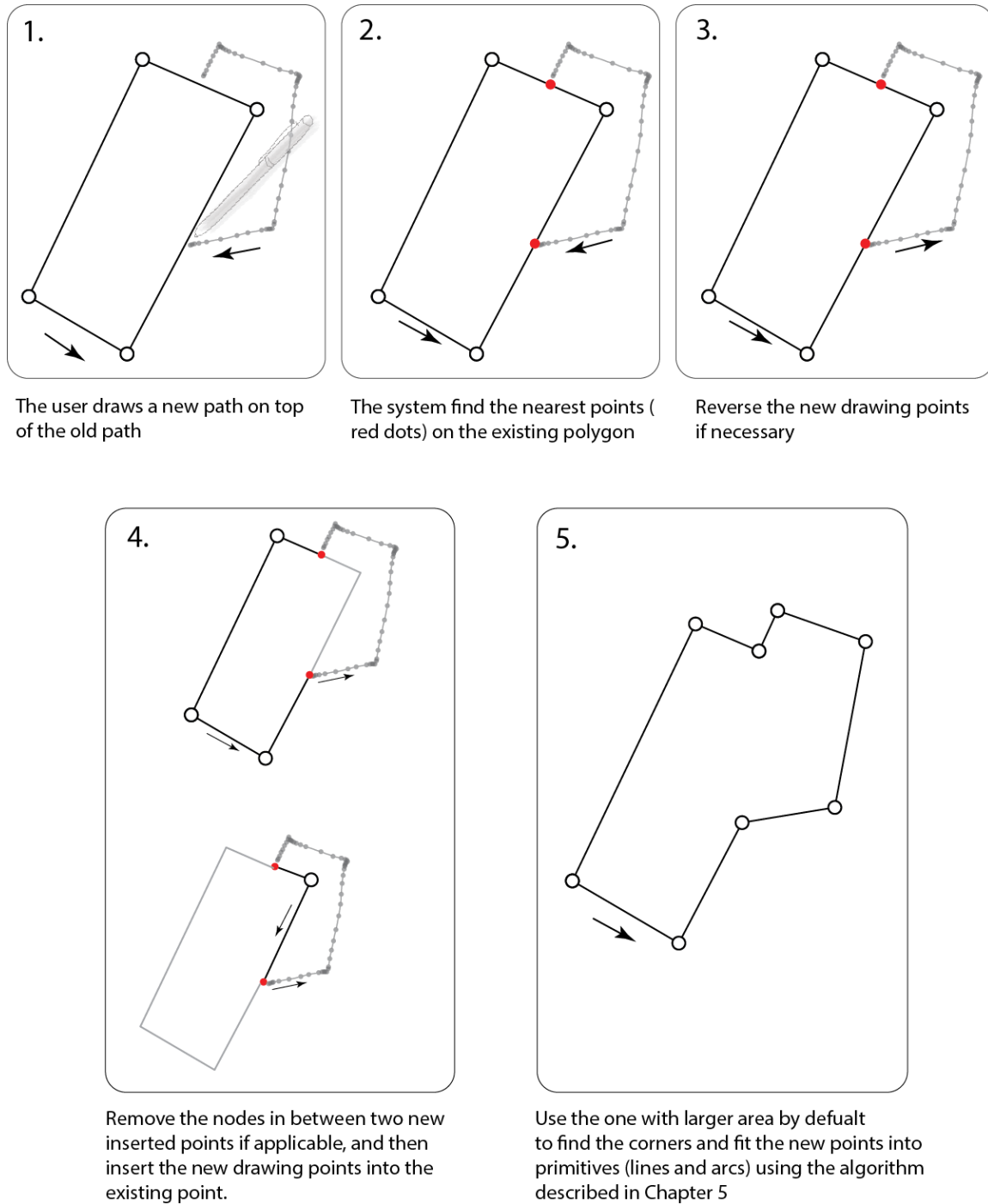


Figure 6.4: The sketch editing algorithm taking the larger shape of the two candidate shapes

Many over-sketch algorithms have been developed for various purposes. The algorithm used by ILoveSketch helps users to form curves (Bae, Balakrishnan, & Singh, 2008). In our case, we wish to increase the capability of editing existing shapes to support design explorations, including polylines and polygons, and maintain as many of the relationships among the existing lines as possible.

The algorithms described in the previous chapter have already suggested the potential surfaces in 3D space for merging new strokes. The system searches the existing lines on these surfaces to test whether the new stroke can be merged into the existing lines or not. In this procedure, the system selects the target polyline individually from these existing lines to test it. The algorithm for merging the new stroke to the target polyline consists of four parts depicted in Figure 6.4: (1) identify potential points on the target polyline for merging (Figure 6.4-2); (2) reverse the order of points in the new line, if necessary (Figure 6.4-3); (3) remove the selected points in the target polyline and insert the new points (Figure 6.4-4); and (4) compare the two candidate point sets and select the one with a larger area, and then use the algorithm described in the previous chapter to fit it into primitives.

Before the system can run the merging process, it needs to transform the raw points from the new stroke on the screen coordinate system into the same space of the merging target. To do this, the algorithm applies the related transformation T_r to the raw points on the new stroke.

$$T_r = T_g \times T_c \quad (6.1)$$

T_g is the 4x4 transformation matrix describing the location and orientation relationships between the target polyline coordinate system and the global coordinate system, and T_c is the transformation matrix between the global coordinate system and the 2D screen space of the view camera on which the user draws the new strokes. After the system transforms the new stroke into the same space as the target polyline, it starts to identify potential points for merging. In this step, the system runs through all of the line and arc segments on the target polyline to identify the nearest points to the

starting and ending points of the new stroke. To acquire the nearest point on the target line, we use the following equations:

$$Pt_{nearest} = \text{Min}(\text{Distnace}(Pt_{proj1}, Pt_{start}), \dots, \text{Distnace}(Pt_{projn}, Pt_{start})) \quad (6.2)$$

$$Pt_{projn} = \begin{cases} Pt_A + V_{proj}, & \text{if } Pt_A + V_{proj} \text{ is between } Pt_A \text{ and } Pt_B \\ \text{Null}, & \text{Otherwise} \end{cases} \quad (6.3)$$

$$V_{proj} = \text{Normalize}(V_{AB}) \text{Length}(V_{Astart}) \cos(\text{Angle}(V_{AB}, V_{Astart})) \quad (6.4)$$

$$V_{AB} \in V_{segments} \quad (6.5)$$

where $V_{segments}$ is the vector from each segment in the target line, Pt_A and Pt_B are the two end points in each $V_{segments}$, Pt_{start} is the starting point of the new stroke, and V_{Astart} is the vector from Pt_A to Pt_{start} . The two nearest points are shown as the red dots in Figure 6.4-2. We apply the same equations to the end point of the new stroke to get the nearest point on the line segments. As shown in the pseudocode of Figure 6.5, “nearStart” and “nearEnd” are the products of the equations shown above. If the target shape is a closed polygon, it requires both ends of the new line to touch the target shape; if the target shape is a regular polyline, however, the new line only needs to touch at least one end of the existing line.

After the system identifies the two nearest points from the starting and ending points on the target line, it removes the points in the target line in between Pt_A and Pt_B , and inserts the points from the new stroke into it. As shown in Figure 6.4-4, there are two possible candidate sets of points when using a different order of the target points. The system compares the area enclosed by the two candidate points and selects the one with a larger area than the merged point set. Finally, the system tries to use the algorithms introduced in the previous chapter with the merged point set for determining the new corners and primitives.

```

1  Function TryMergeExistingPolylineAndNewStrokes(existingPolyline, newStrokePts):
2      # get the 4x4 transformation matrix from the existing polyline
3      matrix = existingPolyline.transformationMatrix
4      # transform the new stroke to the same space of the old strokes
5      transformNewStrokePts = matrix.transform(newStrokePts)
6
7      # Both nearStart and nearEnd points contain the index of the nearest node in the existing polyline
8      # if the distance between the start/end point on the newStroke is greater than a threshold
9      # then set nearStart/nearEnd to null
10     {nearStart, nearEnd} = FindNearestPointsOnExistingLine(existingPolyline, transformNewStrokePts)
11
12     if nearStart is null and nearEnd is null
13         return null
14
15     if existingPolyline is closed
16         if nearStart.index > nearEnd.index
17             transformNewStrokePts.Reverse()
18             switch(nearStart, nearEnd)
19
20         # initialize two candidate points
21         set candidatePoints1 to List of Point
22         set candidatePoints2 to List of Point
23
24         # Add the following list of points into candidate 1
25         candidatePoints1.AddRange(existingPolyline.Range(0, nearStart.index))
26         candidatePoints1.Add(nearStart)
27         candidatePoints1.AddRange(transformNewStrokePts)
28         candidatePoints1.Add(nearEnd)
29         candidatePoints1.AddRange(existingPolyline.Range(nearEnd.index, existingPolyline.length))
30
31         # Add the following list of points into candidate 2
32         candidatePoints2.AddRange(existingPolyline.Range(nearStart.index, nearEnd.index))
33         candidatePoints2.Reverse()
34         candidatePoints2.Add(nearStart)
35         candidatePoints2.AddRange(transformNewStrokePts)
36         candidatePoints2.Add(nearEnd)
37
38         area1 = getArea(candidatePoints1)
39         area2 = getArea(candidatePoints2)
40         if(area1 > area2)
41             return candidatePoints1
42         else
43             return candidatePoints2
44     else if existingPolyline is not closed
45
46         connectionPoint = FindNearestConnectionPoint(existingPolyline, nearStart, nearEnd)
47
48         if connectionPoint is equal to nearStart
49             # start point of the existing polyline to start point of the new stroke
50             if distance(connectionPoint, existingPolyline.StartPoint) < distance(connectionPoint, existingPolyline.EndPoint)
51                 return combinePoints(transformNewStrokePts.Reverse(), existingPolyline)
52             # end point of the existing polyline to start point of the new stroke
53             else
54                 return combinePoints(transformNewStrokePts, existingPolyline)
55         else if connectionPoint is equal to nearEnd
56             # start point of the existing polyline to the end point of the new stroke
57             if distance(connectionPoint, existingPolyline.StartPoint) < distance(connectionPoint, existingPolyline.EndPoint)
58                 return combinePoints(transformNewStrokePts, existingPolyline)
59             # end point of the existing polyline to the end point of the new stroke
60             else
61                 return combinePoints(existingPolyline, transformNewStrokePts.Reverse())
62     end Function

```

Figure 6.5: The pseudo code for merging new stroke into existing lines

6.2.3 The Stripe Shape

The stripe shape is the product of applying the “extending dimension” tool on a 2D line segment. It is a subtype of the closed polygon and thus shares the same properties. In Chapter 4, we described that Peter used the “extend dimension” tool to create a custom wall profile when the preset prototype templates did not fit his needs. When using the “extend dimension” tool on a selected line segment, the system will generate offset lines in real time. As shown in Figure 6.6-2, the system first

offsets each segment from raw sketch points with the distance $D_o = L_s/2$, where L_s is the length of the selected segment. In Figure 6.6-3, the system finds the intersection points between the offset lines. Then, the system uses the polygon clipping algorithm proposed by Vatti (Vatti, 1992) to merge (union operation) the two sets of the offset lines, which results in a closed polygon. This polygon is similar to the polygon object drawn manually by the users, except that the system will mark the polygon as having been generated by the “extend dimension” tool. Once the user’s pen is up, the system performs the same primitive fitting process to determine all of the primitives first, and then uses these primitives in this line offsetting procedure to determine the final stripe shape.

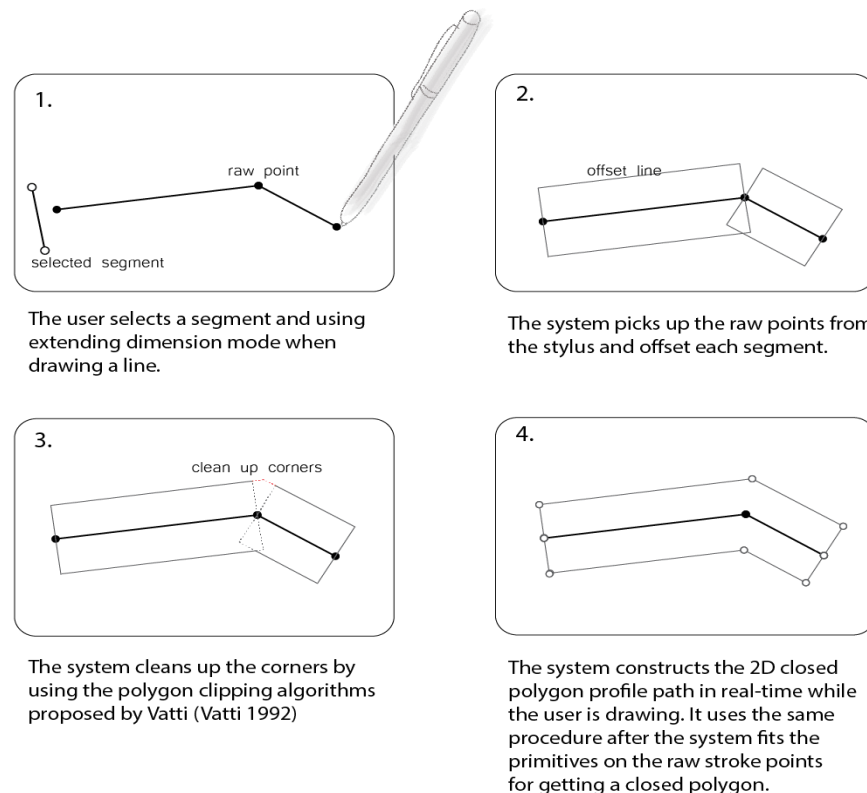


Figure 6.6: The offset algorithm consists of four steps. It runs in real-time when the user is moving their stylus to provide embodied feedback as well as after the user’s pen is up to finish the current drawing

6.3 Three-Dimensional Geometries

Solid modeling programs, such as SolidWorks or building information modeling programs, usually create 3D geometries through a set of input parameters. The system creates meshes based on these

parameters for visualizing shapes on a 3D canvas. The most basic type of solid modeling program includes a set of primitives, such as boxes, cylinders, cones, and spheres. These systems also include more advanced object types that can describe more generic shapes used in many cases. For instance, the “swept solid”, a typical solid geometry used in many of these solid modeling programs, describes a 3D solid geometry that is composed of a profile shape and a path. This swept solid is generated as the profile shape sweeping along with the path. These programs also include basic Boolean operations for users to create more complicated shapes, including union, intersection, and subtraction. In our system, we have implemented several important solid and parametric modeling objects for creating 3D shapes, such as swept solid and Boolean subtraction objects. Each object has different input parameters for generating meshes of the solid shapes. However, most of them are based on the polylines and polygons introduced earlier in this chapter.

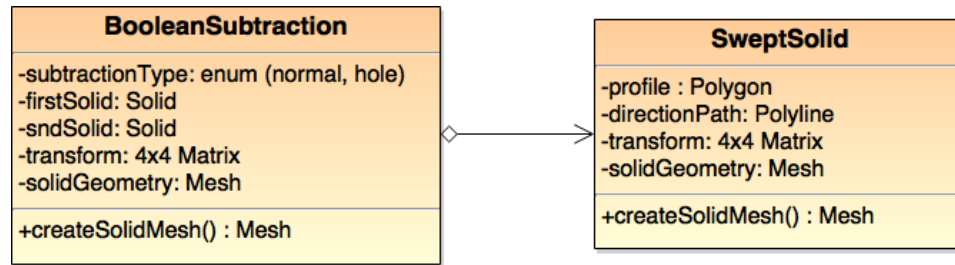


Figure 6.7: The UML diagram describing Boolean Subtraction and Swept Solid parametric objects used in the system. The profile of the SweptSolid here is linked to the Polygon parametric objects described in the previous section

For instance, the system takes the list of line and arc segments constructing a polygon, which is used as the profile shape in a swept solid as shown in Figure 6.7. Without any specification, the system extrudes the profile similar to *SketchUp* while the user moves his stylus in real time. The system uses the distance, $D_{projZ} = \text{Distance}(Pt_{projLast}, Pl_{profile})$, as the extrusion height for determining the direction path from the centroid point of the selected profile. $Pt_{projLast}$ is the projected point from the last 2D stylus raw point on the screen to the virtual plane Pl_n . This plane is made from the normal direction of the current selected polygon shape and the upward direction of

the current view camera. $Pl_{profile}$ is the plane on which the profile is situated. The system continues to update this solid shape while the user is moving his stylus by updating the $Pt_{projLast}$ in real time to determine the meshes. If the user clicks on the XZ or YZ surfaces on the axis widget on the selected profile before drawing a path, the system will use the projected points onto the XZ or YZ plane as the directional path in real time. It uses the fitted sketch polyline as the directional path after the user's pen has been lifted up. The example product of this type of swept solid is shown in Figure 6.8-right.

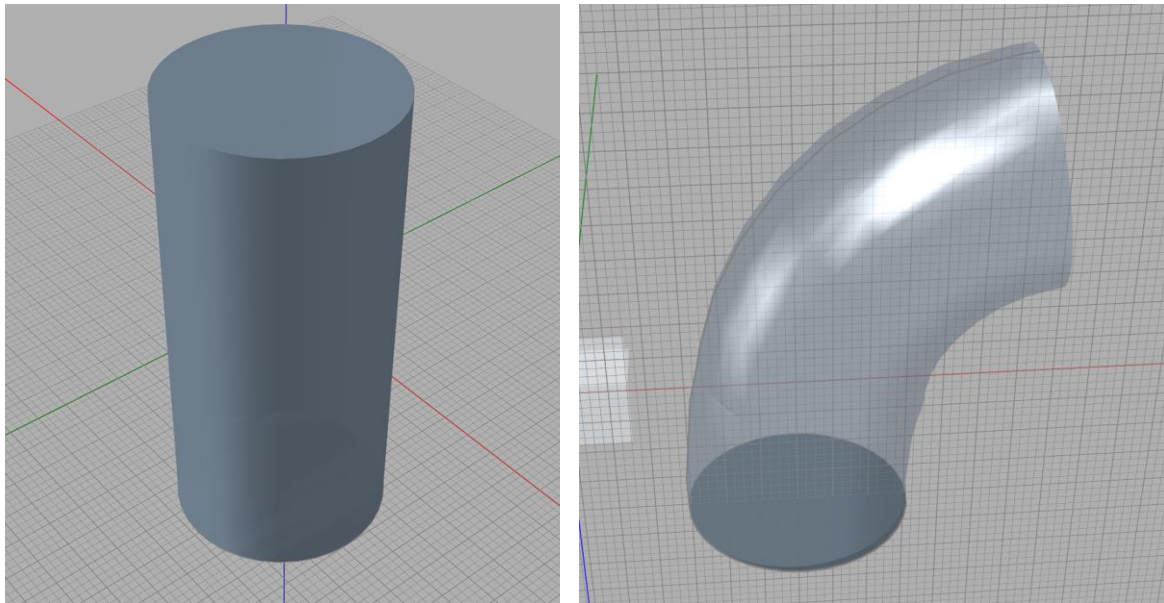


Figure 6.8: Two types of swept solid in SolidSketch. Left: without any specification, the user extrudes the swept solid in one direction without any specification of the drawing plane. Right: when the user specifies the drawing plane on the XZ or YZ planes of the extruding profile, the system tries to use the sketch polyline as the path for sweeping to a solid shape.

The Boolean subtraction object is another important solid modeling operation used in the system. In the scenario described in Chapter 4, Peter drew a closed profile shape on a surface of a wall and performed a scratch gesture to create an opening. He also used two solid shapes to perform a subtraction operation using two objects. The mechanisms behind these two different interactions are essentially the same. The major difference is that the system creates an invisible swept solid from the drawn profile shape in the first interaction example, whereas it uses the actual solid shape on the canvas for Boolean subtraction in the second interaction example.

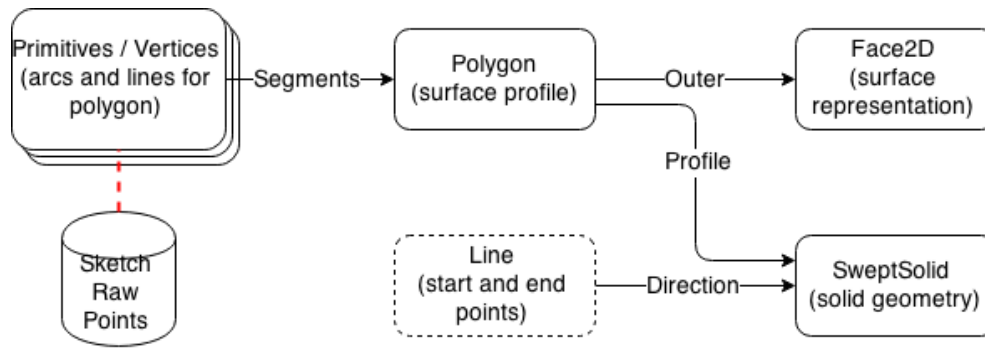


Figure 6.9: Simplified relationships created through the interactions between various parametric objects for a solid shape. Changes in the sketch raw points will propagate changes in the parametric objects in the solid box in the figure.

As shown in Figure 6.9, similar to 2D geometries described previously, the system maintains relationships between different related 2D and 3D parametric objects. For instance, if the user changes the profile of a swept solid, it will propagate the changes and recalculate the meshes for the dependent Boolean object immediately.

6.4 Constraints

In addition to creating the parametric relationships mentioned previously, users can also create constraint relationships by explicitly or implicitly assigning rules between objects during the interactions described in the previous chapters. The system implements three types of constraints: (1) node-edge relation constraints (e.g., right-angle constraints); (2) edge-object relation constraints (e.g., an array of objects on a path); and (3) object-object relation constraints (e.g., transformation constraints).

Each type of constraint contains unique parameters responsible for populating changes on the constraining objects from the interactions. The system uses a graph search algorithm, such as BFS, to populate changes from an object in all of the connected objects through parametric and constraint relations. For example, Figure 5.12 shows the setup of a right-angle constraint on a corner of the bottom profile of a solid. This action will create a constraint on the vertex (node) of that corner and drive changes in the connected edge primitives to satisfy the constraint. Since these primitives are

the fundamental components of the Polygon, Face2D, and SweptSolid objects, it will also drive changes in these objects. As the system changes these objects, it also searches for additional constraints that may be attached to the objects. If the system detects these additional constraints, it will add the existing constraints to the queue for a subsequent graph search. The example in Figure 5.12 shows constraint setting cases in which the surface of a solid shape has a 2D polygon shape attached to it. In this scenario, the right-angle constraint on the corner of the base polygon causes the 2D polygon shape to rotate on the solid to ensure that its attachment to the surface of the solid is properly maintained. Figure 6.10 shows the structure of the relationships among all of the parametric objects in the chair example illustrated in Figure 6.10-right.

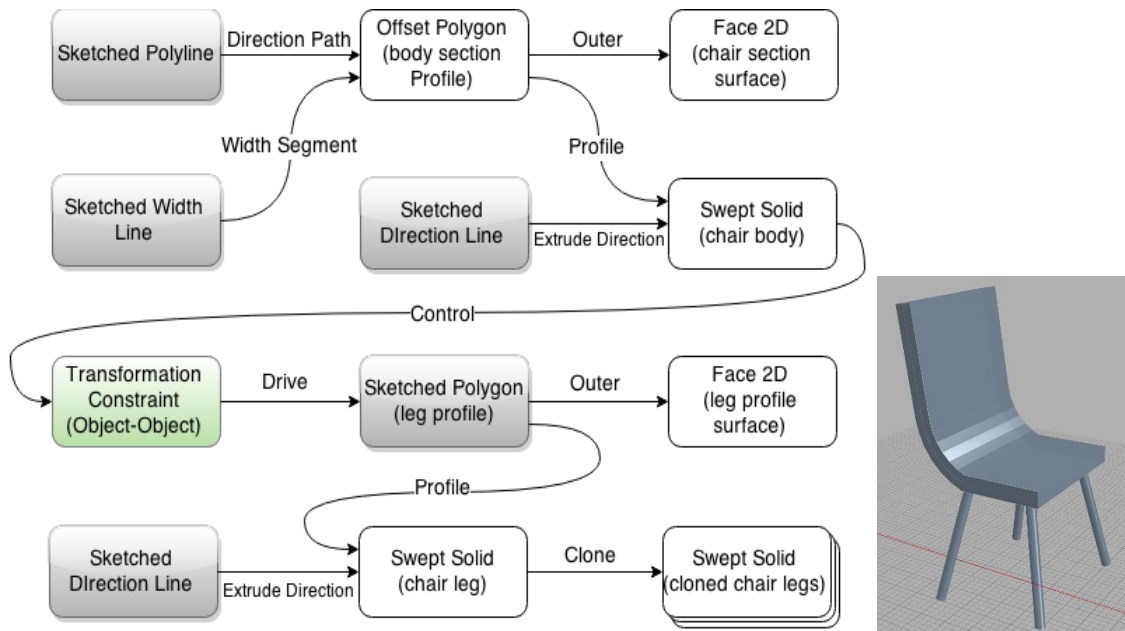


Figure 6.10: The actual parametric structure of the chair shown on the right. Gray boxes: directly created by sketch strokes; white boxes: created by a combination of strokes and interactions; green box: constraint that connects two sets of parametric objects.

6.5 Operations

6.5.1 List of Operations

For our purposes, an operation is defined as one or more actions recorded when the user is in the template composition mode. The system creates operations for describing the composition of a

template for geometry generation when the user is employing a template to create models in real time. In Chapter 4, when Peter used his stylus to trace the orange line by using the “extending dimension” tool with the line segment selected, the system created the first operation “OffsetToFace2D” as shown in Figure 6.11. This operation takes the selected line segment as the “width” parameter and the orange line as the “path” parameter for generating the stripe shape. Since a stripe shape is a type of polygon, as mentioned previously, the system takes it with another polyline as the direction path for the second operation, “extrude”, to create the swept solid. These two operations are stored in the system and ready to be used as a wall while designing a house.

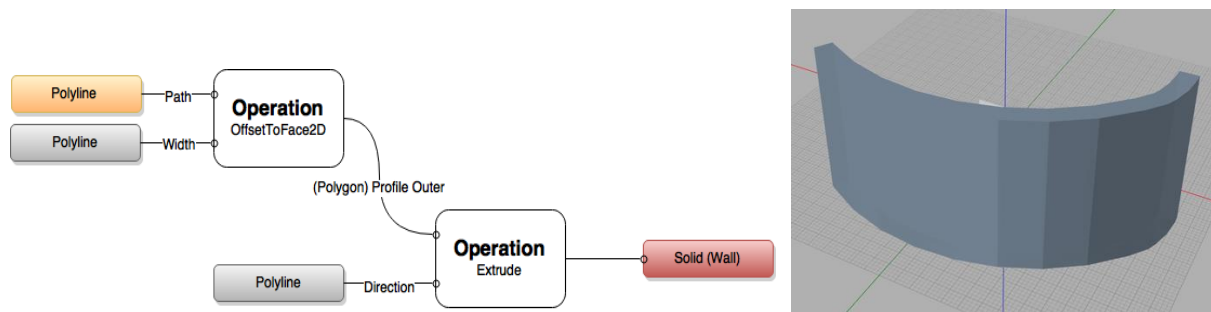


Figure 6.11: The operations (left) that composes the wall template (right). On the left, the orange box represents the sketch points from real-time sketch stroke. The gray boxes are the geometries recorded and transformed from user’s sketching line during the template composition mode. The red box is the product from the two operations showing on the right figure.

The system also takes the following operations. When a user interacts with an object, either a 2D or 3D shape, and traces a line by using the array command, the system will take the object and the line as input sources for the “*MakeArray*” operation. The system can also execute the “*CloneLine*” operation when the user employs a multi-touch gesture to clone an orange line into two lines. These non-closed lines can be used for tracing when the user is attempting to create stripe shapes or make arrays. If the user performs the subtraction command with solid shapes, the system will also create a “*Subtraction*” operation. Figure 6.12 shows a template with many operations (Figure 6.12-left) and the graph that describes how these operations are stored in the system (Figure 6.12-right).

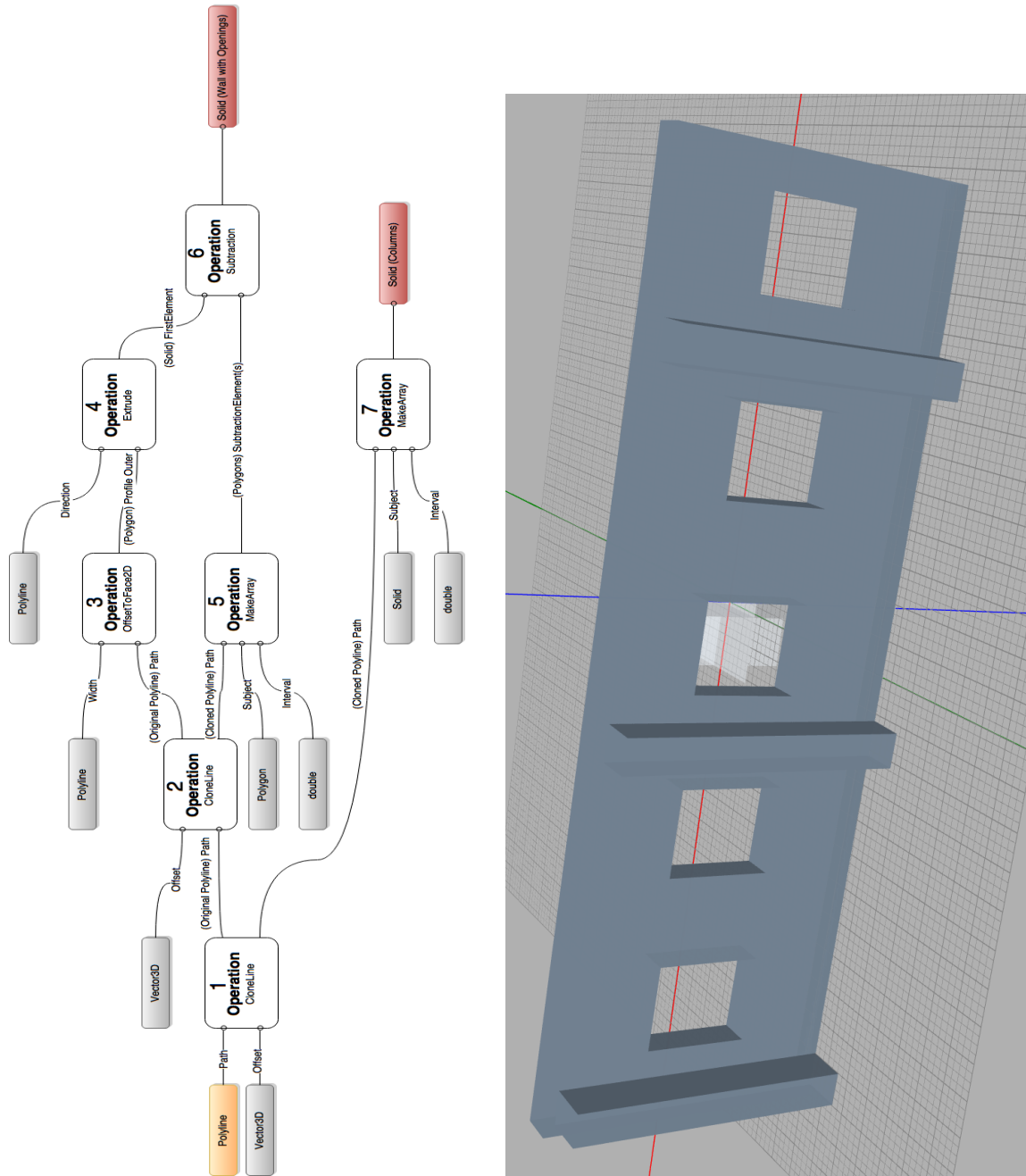


Figure 6.12: The operations (left) for a complicated template that consists of openings and columns on a wall. Gray boxes are those recorded in the template composition procedure. The orange box is the sketch points that are converted into polyline in real-time. The two red boxes represent the final geometries generated in real-time.

6.5.2 Generate Feedback in Real Time

When generating geometries in real time for embodied feedback, the system replaces the orange line that appears during the template composition process with the real-time sketch points captured by the tablet. When regenerating the meshes from each operation, the system needs to properly

schedule and optimize the order of executing operations to avoid any blocking issues when the required inputs are not yet computed from the dependent operations.

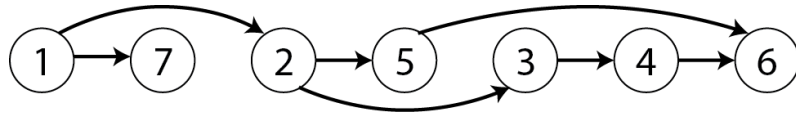


Figure 6.13: The execution order for the operations shown in 6.12-left by using a naïve topological sorting approach.

In general, since operations are a directed acyclic graph (DAG), a naïve topological order would solve the scheduling problem. For instance, Figure 6.13 shows the order of executing a series of operations in Figure 6.12-left. However, since the system renders geometries in real time, it also has to carefully choose which geometries need to be generated first in the template. The order in Figure 6.13 is generated using a naïve topological sorting algorithm in which the user would see the columns generated first (operation 7), then the geometries for creating openings are generated (operation 5), and finally the walls with openings (operation 6). Since geometry generation is computationally expensive, it is impractical for the system to generate all of these geometries in real time. In practice, the visual experience, from the user's perspective, would be a series of columns generated along with the real-time stroke inks while moving the stroke. Thus, this real-time feedback would not provide the users a smooth experience of what they are creating while using the system.

Table 6.1: The inputs, outputs, and weights for the operations used in the system.

Operation	Input(s)	Output(s)	Weight
CloneLine	1. Polyline (Path) 2. Vector3D (Direction)	1. Original Polyline 2. Cloned Polyline	5
OffsetToFace2D	1. Polyline (Path) 2. Polyline (Width)	1. Polygon	4
Extrude	1. Polygon (Profile Outer) 2. Polyline (Direction)	1. Solid	3
MakeArray	1. ParametricObject (subject) 2. Polyline (Path) 3. Double (Interval)	1. ParametricObjects	2
Subtraction	1. Solid (FirstElement) 2. Polygon(s)/Solid(s) (SubtractionElement(s))	1. Solid	1

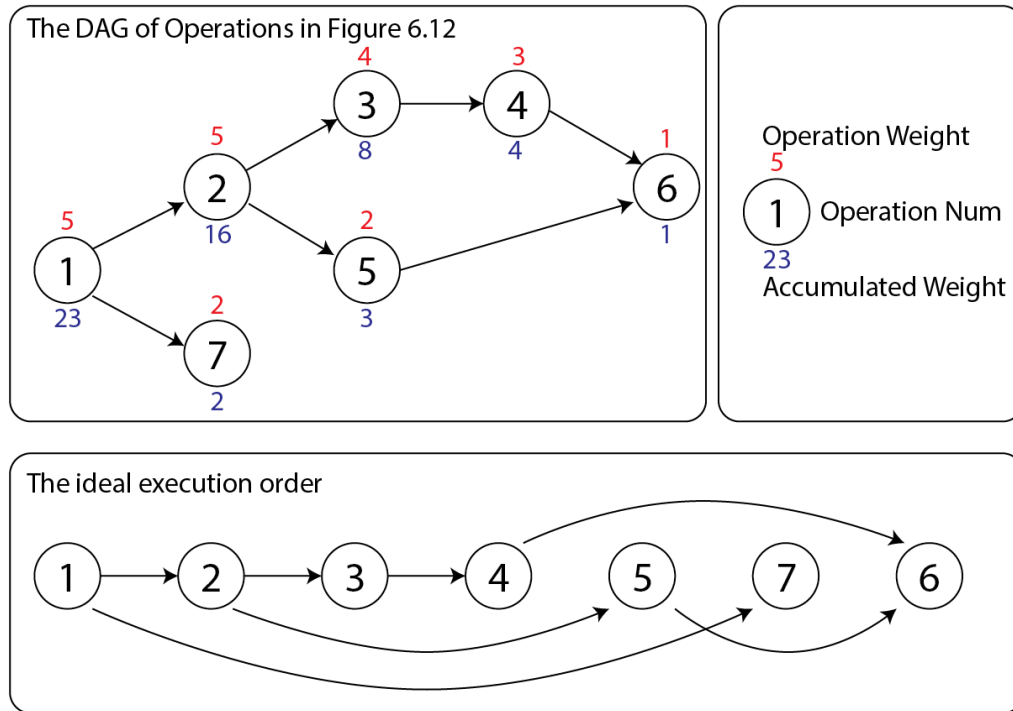


Figure 6.14: The DAG from the operations in Figure 6.12 left and its ideal execution order

Instead of naïvely using the topological order, we give each operation a different weight, as shown in Table 6.1, which influences how the system runs. To compute this execution order, the system first uses graph search to compute the accumulated weights for each operation, which is depicted as the blue numbers in Figure 6.14-top. The system then stores all of the operations into a max heap according to their weights. Next, it pops the operation with the largest weight into this heap to run first. For example, according to Figure 6.14, when the system runs to the operation 1 in Figure 6.12-left, it will choose the node with larger weight to run first, which is the operation 2 in this case. The pseudocode in Figure 6.15 shows the detailed algorithms the system uses to build the operation execution order. This procedure only runs once when the user finishes the template building procedure. Whenever the user employs the template, the system will run through the execution order to generate the geometries progressively. If the system is not able to generate any geometries in real time, it will choose the geometries with larger weights and abort the geometry generations for those with lower weights in order to maintain real-time continuous feedback.

```

1  Function FindSchedulingOrder(operations):
2      # The first operation is always the starting operations
3      startOperation = operations.first()
4
5      GetWeightsOfOperations(startOperation)
6
7      # insert operations into a heap
8      # pop the max element in the heap each time when finishing
9      # an operation will result in the ideal execution order
10     maxheap = new MaxHeap()
11     for curOpr in operations
12         Maxheap.insert(curOpr)
13
14     return maxheap
15 end Function
16
17 Function GetWeightsOfOperations(operation):
18     # InitialWeight is the first weights
19     weight = operation.InitialWeight
20
21     if operation.isVisited
22         return operation.weight
23
24     for nextOpr in operation.NextOperations
25         weight += GetWeightFromOperations(nextOpr)
26
27     operation.isVisited = True
28     operation.weight = weight
29     return weight
30 end Function
--

```

Figure 6.15: The pseudo-code for computing the order of executing the operations

6.6 Mapping to Semantic Data

In Chapter 4, once Peter finished his design, he could quickly export the model he created in SolidSketch to other Building Information Modeling programs. In Chapter 2, we mentioned the Model View Definitions (MVDs) that define various use cases for designing. We adopted and slightly modified the MVD, “Concept Design BIM 2010” (CDB10), which fits the use case we are aiming to support. As mentioned, this use case is even earlier than CDB10 in the design process. As a result, some of the data needed originally in CDB10 were removed. Figure 6-16 shows the example data structure used for a wall object. Other geometries created in the system have similar structures as the wall element. In the following sections, we will describe the mapping between our internal data structure to the following concepts by using the concepts in IFCWall as an example: “*Shape Representations*”, “*Root Attributes*”, “*Generic Definition*”, “*Generic Object Placement*”, “*Generic Voiding*”, “*Generic Connectivity*”, and “*Generic Containment*”.

6.6.1 Shape Representation Concepts

When the user creates a wall template, SolidSketch records the parametric relationships for the geometries that compose a wall. Once the user draws a wall using the template, the system uses the sketch line to generate a wall parametrically as discussed earlier in this chapter. These parametrically generated geometries and their relationship information will be stored in the concepts in the category of *"Shape Representation"* in Figure 6-16. For instance, the *Polyline* object from a sketch stroke will be described parametrically in the concept *"FootPrint"*, which consists of two pieces of critical information: (1) the composition of the polyline in the type of *"IFCCompositionCurve"* that represents the central path of the wall; and (2) the position of this polyline related to the generated wall profile polygon, which is always at the *center* of the stripe profile polygon. The actual geometries of the wall are described parametrically in the concept *"Body"* in *"Shape Representation"*. In SolidSketch, the swept solid is used for constructing a major part of the 3D solid model, which means that the system can easily map related information in the internal data structure to the concept *"Swept Solid"*.

6.6.2 Root Attributes

In Root Attributes, SolidSketch stores information related to the concepts *"GUID"*, *"BIM Object Owner History"*, and *"Name"*. This information is generated immediately when the user is employing a wall template to create a new wall. For instance, *"GUID"* and *"Name"* are directly mapped from the attributes with the same name from the *"ParametricObject"* in the system as shown in Figure 6.1. In the system, the *"GUID"* attributes of *"ParametricObject"* is a set to automatically and randomly generate a 22-character string. During interactions, the system will maintain this same string during the modification and editing process. Similar to *"guid"*, the *"name"* attribute in the parametric object is also automatically generated. However, the system uses a naming convention, such as *"Wall-01"*, to make the documentation more human-readable. For *"BIM Object Owner History"*, an *"IfcOwnerHistory"* that consists of author and the timestamps of the editing histories is automatically

generated when the system is started. This IFC object will be linked to each IFC object when exporting. These attributes are important for individual users, as well as design teams, to keep track of design documents between different programs. Since SolidSketch enables data generation in the early stage of design, the designer can look back at how his design intents were evolved in the early stage of the design process.

6.6.3 Generic Voiding

In the scenario mentioned in Chapter 4, when Peter created a hole in a wall for adding windows, the system created a Boolean subtraction object between the hole and the wall geometry. This Boolean subtraction object is mapped to the concept “Generic Voiding” in CDB 2010. As shown in Figure 6.17, the “*RelatingBuildingElement*” attribute in “*IfcRelVoidsElement*” points to the “*IfcElement*”, which is the wall object in this case. The “*RelatedOpeningElement*” points to “*IfcFeatureElementSubtraction*” that is mapped to the opening geometries.

6.6.4 Other Concepts

The information in “*Generic Definition*” and “*Generic Connectivity*” are mostly derived and computed from the geometric information stored in a parametric object. Since the system is not dealing with material information for the parametric object, it does not create and store the information of the concepts such as “*Wall Composite Thermal Transmittance*” or “*Generic Material Association*”.

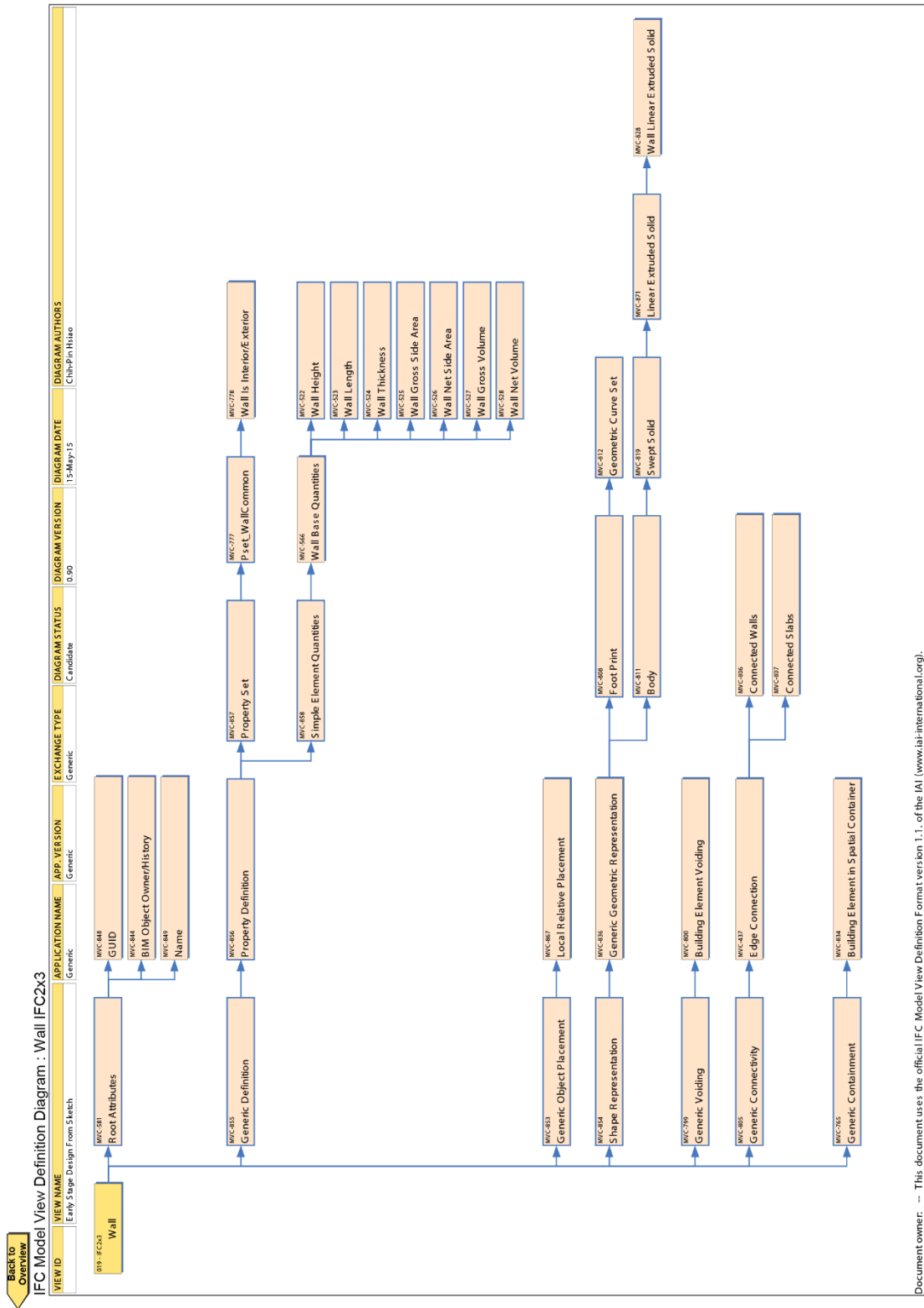


Figure 6.16: The Model View Definition Diagram for wall created by SolidSketch

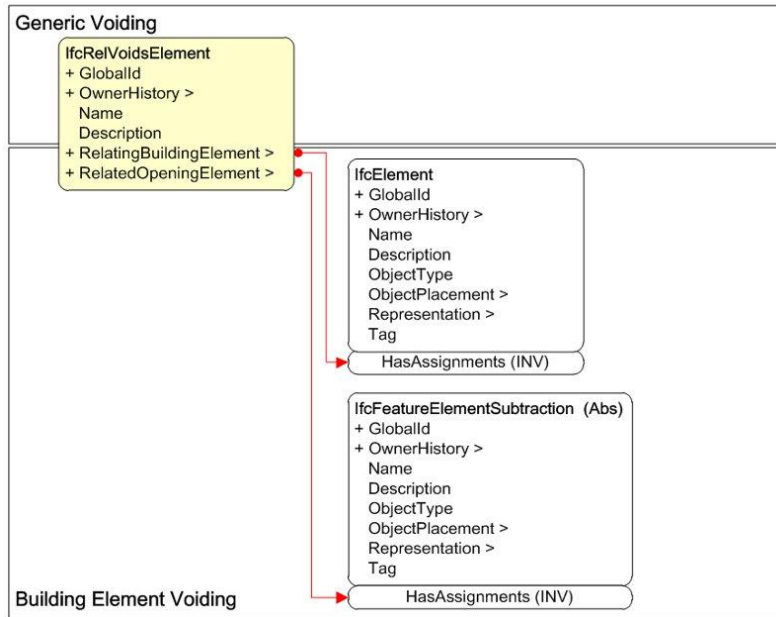


Figure 6.17: The Generic Voiding concept from CDB 2010⁴

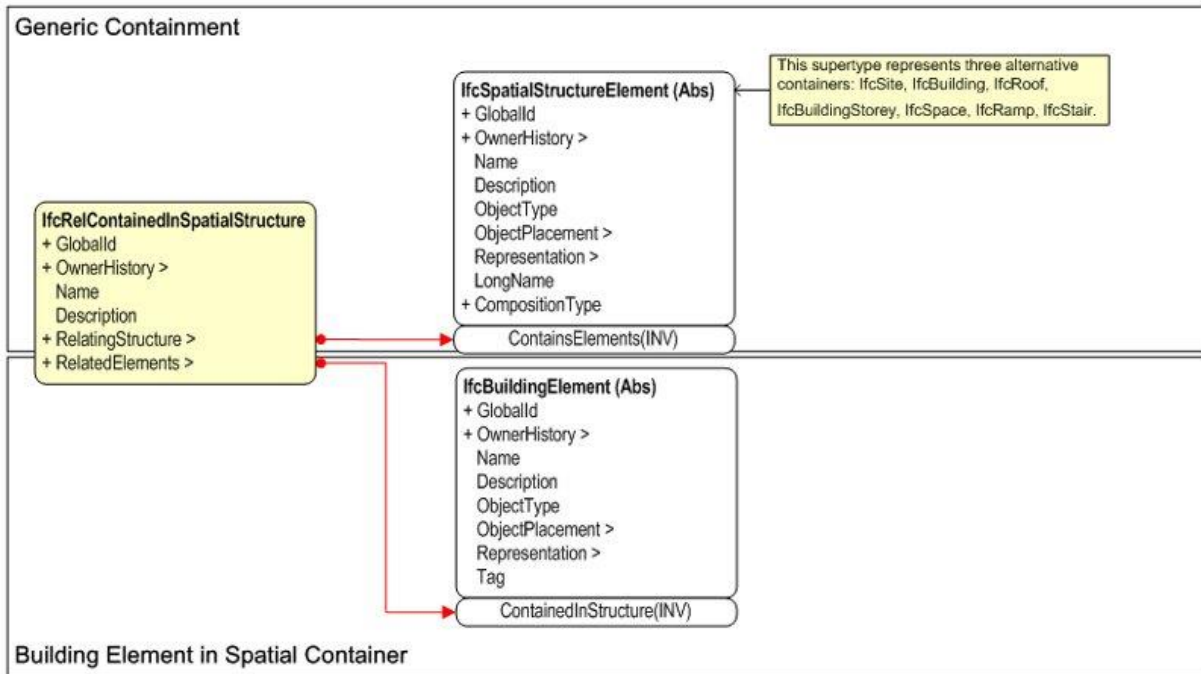


Figure 6.18: The Generic Containment concept from CDB 2010⁵

⁴ <http://www.blis-project.org/IAI-MVD/reporting/showConcept.php?CREF=MVC-800>

⁵ <http://www.blis-project.org/IAI-MVD/reporting/showConcept.php?CREF=MVC-834>

“Generic Connectivity” and *“Generic Containment”* are also generated automatically from the stored parametric objects, their relationships, and their contexts. For instance, when Peter used the wall template to create walls, the system created an *“IfcBuildingStorey”* object and an *“IfcRelContainedInSpatialStructure”* object that linked the building story and wall object together as shown in Figure 6.18. When he drew a profile on top of the wall on the first floor, the system automatically assumed that it was a slab for the second floor and thus created another *“IfcBuildingStorey”* for the second floor. It also created the same structure as the first floor for containing walls and slabs in the building story.

6.6.5 Interoperability between SolidSketch and BIM Programs

Figure 6.19 shows a building model exported from SolidSketch to two major BIM programs: Revit and Solibri Model Checker. Revit is the most popular BIM tool with approximately 70% of the current market share. It provides a platform on which various professionals in the AEC field can create design and construction drawings, collaborate, and perform simple energy or structural simulations. Solibri Model Checker is another platform for checking the integrity of the model as well as building codes. The bottom left of Figure 6.19 shows that Revit can render and create the building data exported from SolidSketch, including walls, slabs, and openings. In Figure 6.19, we demonstrate that users can adjust and edit this building data and input specifications with parameters or mouse manipulations. For instance, in Revit, the user can manipulate the handles of an object (blue arrows in Figure 6.20-left), or type in numbers to adjust the length of a wall created in SolidSketch, as shown on the left of Figure 6.20. With the building information provided by SolidSketch, Revit knows where and how to add the details of the windows and doors on the wall created in SolidSketch, as shown on the right side of Figure 6.20. By using SolidSketch and Revit, designers can flexibly create and explore designs using SolidSketch in the early stage of design, and add more details and increase precision in Revit in the next stages. However, those drawings that do not represent objects in SolidSketch will not be

exported for other BIM programs at this point, since we do not have proper methods of exchanging this data.

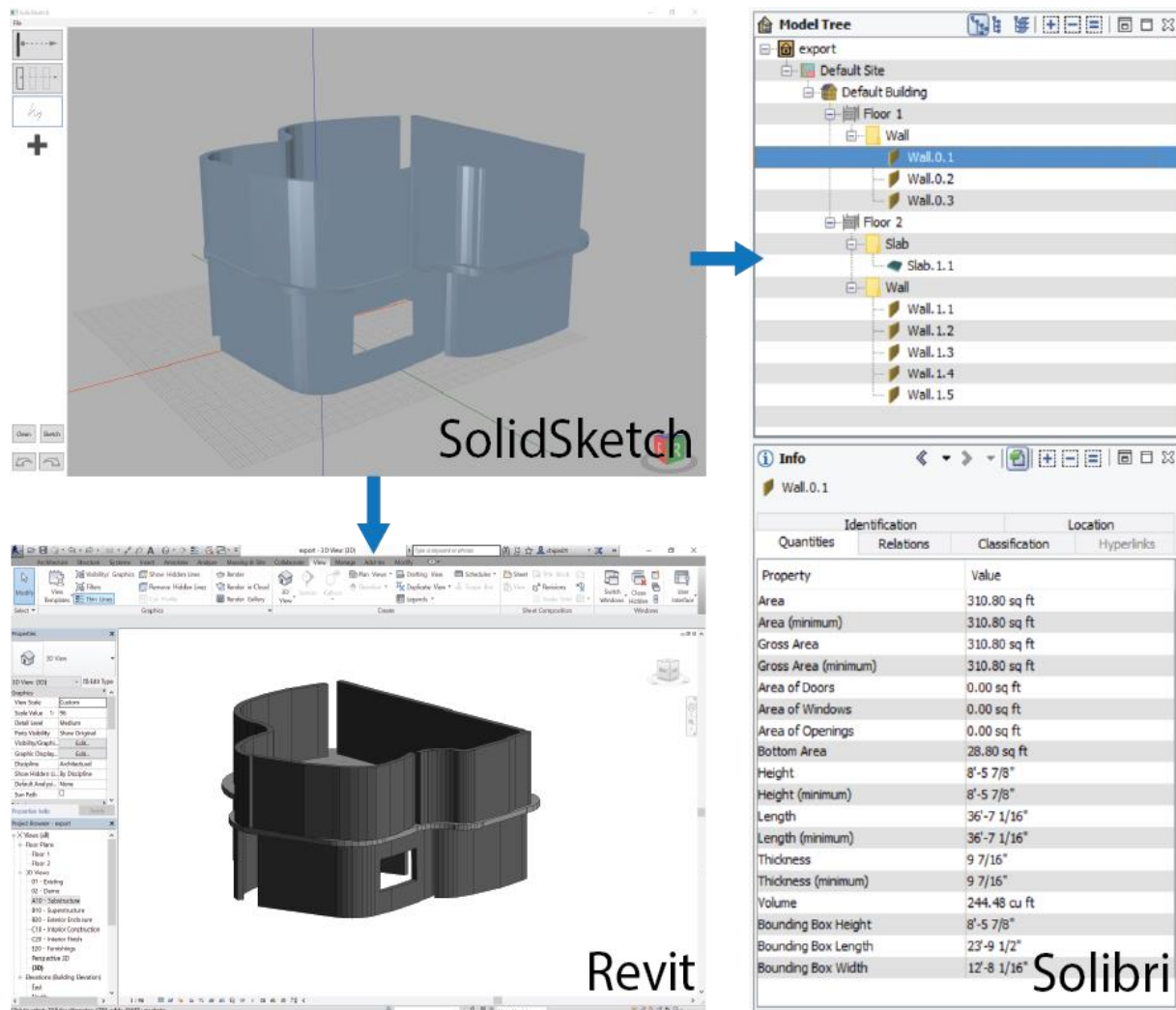


Figure 6.19: The data produced in SolidSketch can be imported into other BIM programs, such as Revit and Solibri Model Checker. The user can add more details or perform simulations in these programs.

The top right of Figure 6.19 presents the topological hierarchy between building objects created by Solibri Model Checker from the data exported by SolidSketch. It shows that SolidSketch creates two building stories with several building elements in them, such as walls and slabs. The building stories are also contained in the building within the site. The bottom right of Figure 6.19 shows that when a user selects a wall in Solibri Model Checker, s/he can see the quantitative information created by SolidSketch, such as height, length, width, bottom area, and gross area of the wall. This

information can be used for model checking against different building codes to verify the design specifications.

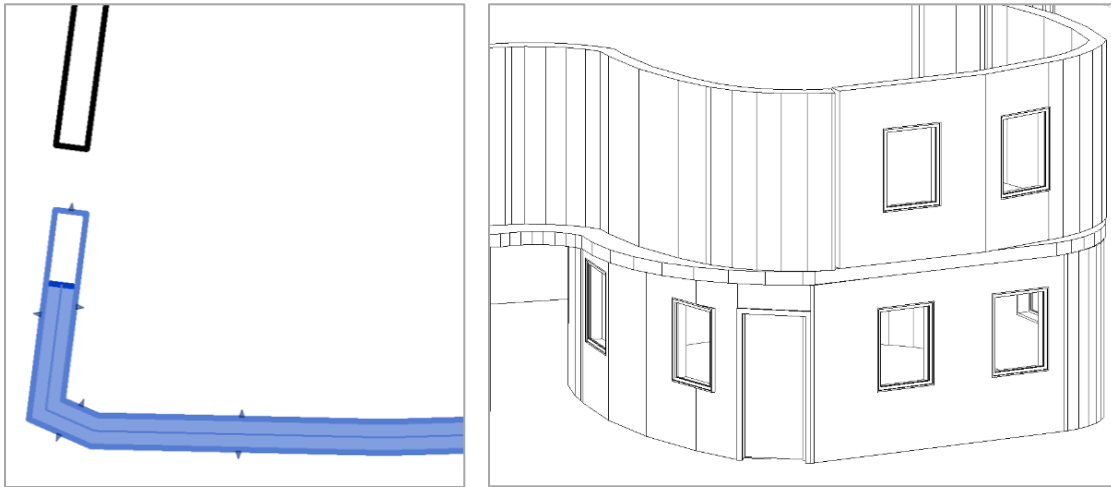


Figure 6.20: The building model created in SolidSketch can be edited in Revit for adjusting precisions and adding more details. Left: the user can adjust the lengths of the walls through grabbing the handles (blue arrows). Right: the users can add windows with more details since the system knows what to do with the specific building elements.

6.7 Conclusions

Chapter 5 and Chapter 6 have described the general architectures and recognition processes, as well as detailed algorithms and data structure, of SolidSketch. Chapter 5 starts by describing how the system discerns the users' intentions by analyzing their interactions through sketch segmentations and primitive fittings. The system then takes the segmentation results to decide whether the user is performing a special gesture or not. By checking with the contexts in the 3D scene and its parametric relationships, the system has higher confidence of whether it should create new geometries or execute commands in the current action. In this chapter, we discussed how the system maps geometries and commands into the internal data structure, and then translates the data structure to a neutral data format – the Industrial Foundation Class (IFC). We also discussed how the system schedules the operation tasks in order to optimize real-time geometry generation. Since many of these processes are automatically generated using on-line or off-line recognition and computation, users can smoothly create and focus on their design tasks, instead of focusing on how to execute

commands and manually entering detailed information. In the next chapter, we will describe a technical and human evaluation of SolidSketch.

Chapter VII. Case Study and Discussions

One of the goals of this dissertation is to translate human interactions to computer-understandable format that can then be utilized in subsequent tasks of the modern design process using different software. In Chapter 2, we discussed the issues of modern CAD programs that were not able to facilitate the rapid, iterative, and embodied thinking processes in the early stage of design. On the other hand, Chapter 3 provides a review of the prior sketch prototypes in which we reveal that there remain gaps for the early stage of design – they are not able to fully support what CAD programs are capable of producing, including sufficient geometrical information and geometrical relationships as shown in Table 3.1. In this chapter, we examine SolidSketch against the criteria in Table 2.1 and summarize the evaluations in Table 7.6 through the following three approaches:

- (1) We compare SolidSketch and typical CAD or BIM programs to ascertain which areas SolidSketch supports and which areas we can still improve in the early stage of design.
- (2) We investigate how the combination of sketch interactions with BIM capabilities affects the early stage of design, especially for the first three criteria in Table 2.1. To do this, we employ a case study with experts designing a building by using sketch and multi-touch inputs.

7.1 Comparison of Existing Programs

Table 7.1: The interaction to achieve fundamental operations in SolidSketch, sketch prototypes (e.g. SKETCH (Zelenik, 1996), IloveSketch (Bae, Balakrishnan, & Singh, 2008), and SIMI (Johnson, 2012)), and typical CAD/BIM tools. In this table, we use (T) to represent touch actions, (P) to represent stylus actions, (M) to represent mouse actions, and (K) to represent keyboard actions.

Comparison	SolidSketch	SKETCH / IloveSketch/SIMI	Typical CAD Tools
Input Device	Touch (T) + Stylus (P)	Mouse/Stylus (P)	Mouse (M) + Keyboard (K)
Create a straight line	Draw a stroke (P)	Draw a stroke (P)	(1) Click the “Line” button (M) or type in the command (K) (2) Click two points on canvas (M)
Create a curve line	Draw a stroke (P)	Draw a stroke (P)	(1) Click the “Curve” button (M) or type in the command (K) (2) Click three points on canvas (M)
Create a composite polygon with line and curve segments	Draw a stroke (P)	Draw multiple strokes (P)	(1) Click the “Polyline” button (M) or type in the command (K) (2) Click multiple points on canvas (M)

Table 7.1 continued

			<p>(3) The user may click on UI buttons (M) or hot keys (K) again to change the modes for straight or curved line segments</p> <p>(4) Click the control points on canvas to adjust the curvature (optional) (M)</p>
Replace several segments on a 2D polyline with line and/or curve segments	<p>(1) Tap on the shape (optional) (T)</p> <p>(2) Draw stroke(s) close to the modifying lines/shapes (P)</p>	<p>(1) Perform a lasso gesture on the shape (P)</p> <p>(2) Draw stroke(s) close to the modifying lines/shapes (P)</p>	<p>(1) Click on a shape (M)</p> <p>(2) Click UI buttons (e.g. trim button) (M) or type in commands (K)</p> <p>(3) Delete the modifying segments by using keyboard (K)</p> <p>(4) Click the UI button (e.g. polyline button) (M) or type in commands (K)</p> <p>(5) Click multiple points to create a new shape that connects to the existing shape on canvas (M).</p> <p>(6) The user may click on UI Button (M) or type in hot keys (K) again to change the mode for straight line segments or curved line segments</p>
Offset a polyline/polygon	<p>(1) touch the line with one finger (T)</p> <p>(2) drag the line with another finger to the desired location (T)</p>	Not Supported	<p>(1) Select the line on canvas (M)</p> <p>(2) Click the "Offset" button (M)</p> <p>(3) Specify the distance with multiple mouse clicks on canvas (M)</p>
Move a shape	(1) Touch and drag the shape (T)	(1) Use pen to drag the shape (T)	<p>(1) Click on "Move" button (M)</p> <p>(2) Move the shape by dragging the mouse (M)</p>
Extrude a profile shape	<p>(1) Tap on the shape to extrude (optional) (T)</p> <p>(2) Hold the button on the stylus to draw a stroke (P)</p>	<p>Supported in SKETCH:</p> <p>(1) Draw the profile shape (P)</p> <p>(2) Immediately after closing the profile shape, draw a straight line for extrusion (P)</p>	<p>(1) Click the "Extrude" button (M)</p> <p>(2) Click on the shape and extrude it (M)</p>
Sweep a profile shape	<p>(1) Tap on a shape (T)</p> <p>(2) Select the desired plane for sweeping (P)</p> <p>(3) Hold the button on the stylus while drawing a stroke (P)</p>	<p>Supported in SKETCH:</p> <p>(1) Draw the profile shape (P)</p> <p>(2) Immediately after closing the shape, draw a stroke for sweeping (P)</p>	<p>(1) Click "Line" button for drawing the polyline (M)</p> <p>(2) Draw the polyline on canvas as described above (M)</p> <p>(3) Click the "Sweep" button (M)</p> <p>(4) Select the profile shape (M) and the new polyline (M) as two parameters for creating the sweep</p>
Create an array of objects on a (poly)line	<p>(1) Click on "Array" button (T)</p> <p>(2) Tap on a profile shape (T)</p> <p>(2) Hold the button on the stylus to draw a stroke (P)</p>	Not Supported	<p>(1) Click "Polyline" button (M)</p> <p>(2) Draw a polyline on canvas as described before (M)</p> <p>(3) Click "ArrayPath" button (M)</p> <p>(4) Select the shape (M) and the new (poly)line (M) for two parameters to create the array of objects</p>
Create a hole	<p>(1) Draw a 2D shape on a 3D shape (P)</p> <p>(2) Perform scratch gesture (P) to delete the 2D shape while the 3D shape is selected (T)</p>	<p>Supported in SKETCH:</p> <p>(1) Draw a 2D shape on a 3D shape (M)</p> <p>(2) Extrude the 2D shape to 3D shape as described previously (M)</p> <p>(3) Delete the 3D shape to create a hole (M)</p>	<p>(1) Draw a 2D shape on a 3D shape as described previously (M)</p> <p>(2) Click "Extrude" button (M) and extrude the 2D shape on canvas (M)</p> <p>(3) Click "Boolean" button (M) for Boolean operation</p> <p>(4) Select the two 3D shapes as the parameters to create a hole (M)</p>
Assign constraints (e.g. right angle)	(1) Perform gesture at the desired location (P)	<p>Supported in SIMI:</p> <p>(1) Perform gesture at the desired location (P)</p>	<p>(1) Click on "Constraint" button (M)</p> <p>(2) Click on multiple line segments that will constitute the constraints on canvas (M)</p>
Record parameterized constraints and rules	<p>(1) Tap "+" button (T)</p> <p>(2) Use the above operations (e.g. assign constraints and offset polyline) in an order to create the simple rule sets (P+T)</p>	Not Supported	<p>(1) Use the above operations (M)</p> <p>(2) Different UI environment (e.g. Grasshopper) typically required with many UI buttons for comprehensive constraints and rule sets (many Ms)</p>
Semantics	Simple semantics from automatic inference based on	Not Supported	Select the desired objects' assembly with specific semantics prior to creation

Table 7.1 continued

	topological relationships between shapes		
BIM Workflows	Supported	Not Supported	Natively supported or through plug-ins

Table 7.1 presents the comparisons of different fundamental CAD operations between (1) SolidSketch; (2) sketch prototypes such as SKETCH (Zelznik et al., 1996), ILoveSketch (Bae et al. 2008), and SIMI (Johnson et al., 2012); and (3) typical CAD programs, including BIM programs. In the column of the sketch prototypes, we choose the simplest possible steps that users need to perform to accomplish the tasks from the listed tools. In general, we wish to create a program that enables users to offload their tasks quickly. The column of typical CAD programs shows how these CAD programs adopt GUI interactions to accomplish various tasks, such as supporting comprehensive parametric and generative modeling, e.g. Grasshopper with Rhinoceros; detailed engineer drafting, e.g. Revit Structure; and generalizing for different professions, e.g. AutoCAD. In these CAD programs, the fundamental operations of creating and editing shapes typically start from clicking on UI buttons, then clicking on the main canvas for pointing locations, and sometimes clicking additional UI buttons for further specifications.

For complicated operations, such as creating a composite polyline, users need to click on UI buttons and then the shape they are working on multiple times to assign the input parameters. For instance, if designers wish to draw a composite polyline with straight line and curve segments, they will need to (1) click the polyline tool first, (2) click on the canvas, (3) change the drawing mode from straight line to curve line through the UI buttons again, (4) click on the canvas to create control points multiple times, and (5) finally click the UI button to finish the creation. Since these systems require users to specify precise geometry and geometrical relationship information along with other details, they usually provide interoperability between different BIM programs and thus support BIM workflows easily.

On the other hand, as shown in Table 7.1, even though the programs that adopt sketch interactions simplify the ways in which we interact with the lines and geometries by using stylus or mouse gestures, they only provide limited capability of parametric modeling (if any), and produce mostly mesh models with no integration to BIM workflows. Designers who use these sketch programs later have to recreate geometries using BIM tools, which could result in losing track of their design intents, such as relationships between geometries. Finally, SolidSketch enables the possibility of 2D and 3D sketch interactions with stylus and multi-touch gestures for supporting constraint assignments on parametric models. The system can infer simple semantics, such as slabs, walls, openings, and other basic building elements, when exporting the design data for other BIM programs to use.

Table 7.2: Comparison of features supported in different 3D sketch tools.

	SKETCH (ZELEZNIK et al., 1996)	Teddy (Igarashi et al., 1999)	Masry's 3D Sketch prototype (Masry et al., 2007)	ILoveSketch (Bae et al. 2008)	True2Form (Xu et al., 2014)	SolidSketch
Direct 2D Line Editing	Not Supported	Not Supported	Not Supported	Over-Sketching	Through control points	Over-Sketching
2D Shape Editing	Not Supported	Not Supported	Not Supported	Not Supported	Through control points	Over-Sketching
Mirror Sketching	Not Supported	Not Supported	Not Supported	Supported	Not Supported	Not Supported
3D Solid Shape Editing	Through stylus gestures (hole, scale, etc.)	Through stylus gestures (cut (one stroke), bite (two strokes), etc.)	Not Supported	Not Supported	Not Supported	Through stylus gestures (hole, cut, etc.)
Shape Manipulations (Rotate, Move)	Through stylus with gestures	Not Supported	unknown	Not Supported	Not Supported	Through multi- touch gestures
Sketch to 3D	Gestures with different mouse buttons to create 3D	Blob shapes to create 3D	Rigorous perspective sketch to create limited 3D geometries	Natural sketch with gestures on scaffolding assistant lines	Natural sketch with GUI buttons and automatic inference	Natural sketch with buttons on stylus
Multi-touch	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Supported
Multi-touch Gesture	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Supported
Navigation	Through mouse gestures	Through different combinations of stylus gestures	unknown	Through stylus with physical buttons and gestures	Through GUI button and stylus	Through conventional multi-touch gestures
NURBS Surface	Not Supported	Not Supported	Not Supported	Use gestures in enclosed curves	Automatically inferred from sketch lines	Not Supported
Stylus Gesture	For creating various 3D geometries	For editing 3D geometries and navigations	Not Supported	For navigation, constraints,	Not Supported	Mainly for assigning

Table 7.2 continued

				delete, and other purposes		constraints and deleting
2D Constraints (right angle, same length, etc...)	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Through stylus gestures
3D Constraints (on surface, touch, align, etc....)	Constraints through auto inferencing or gestures when sketching	Basic constraints (e.g. on surface) through auto inferencing when sketching	Not Supported	Through gestures for surface constraints	Not Supported	Constraints through auto inferencing when sketching
Operation Behind Objects	Not Supported	Not Supported	Not Supported	Operating on surfaces	Not Supported after the 3D shape is inferred	Automatically inferred through contexts
Parametric Lines	Not Supported	Not Supported	Not Supported	Supported	Supported	Supported
Parametric 3D Shapes	Basic primitives	Not Supported	Not Supported	Not Supported	Not Supported	Solid geometries in common solid modeling programs
Relationships between 2D/3D Shapes	Not Supported	Not Supported	Not Supported	Not Supported	Not Supported	Contain, connect, and other topological relationships between shapes

Table 7.2 shows the major features supported by different sketch programs. The recent sketch to 3D research prototypes focus on creating 3D geometries automatically from 2D sketching lines, e.g. ILoveSketch (Bae et al. 2007) and True2Form (Xu et al. 2014). While these prototypes enable users to rapidly create new geometries and easily edit existing geometries through over-sketching, they miss the functionalities of inferring the geometrical relationships, which are important for interoperability in the early stages of design. The earlier prototypes, such as SKETCH (Zelznik 1996), infer many 3D constraints through many different sketch gestures, which requires some learning curves. They do not employ natural sketch techniques, such as over-sketching, to avoid disrupting designers' workflows. More importantly, they only adopt simple constraints for interaction purposes, instead of inferring the relationships between geometries for the parts that users are specifying. SolidSketch incorporates the important features of both sketch prototypes and CAD programs. It enables the creations of geometries and topologies from sketches, stylus gestures, multi-touch gestures, and automatic inferences. This information can be used for basic building information exchange and generative design purposes for subsequent stages of design. However, at this moment, SolidSketch

does not support the NURBs surfaces that are supported by ILoveSketch and True2Form. In the next section, we will examine the interoperability between SolidSketch and the most popular BIM programs.

7.2 Design of the Case Study

Table 7.3: The participants of the case study and their brief information

ID	Name	Age	Gender	Education	Sketch Skill	CAD Skill
P1	Tommy	39	Male	College	Medium	High
P2	Linda	39	Female	Master	Medium	High
P3	John	32	Male	Master	High	Medium
P4	Robert	23	Male	Master	Medium	High

We invited four architecture designers and students with advanced architecture degrees to test SolidSketch. The age range of the subjects was between 23 to 39 years old, with one female and three males. As shown in Table 7.3, the three male participants were P1 - Tommy (39), P3 - John (32), and P4 - Robert (23), while the female participant was P2 - Linda (39). Because we placed the UI panels on the left side of the system for the non-dominant hand to use, we only selected right-handed subjects when we recruited the participants. We conducted the experiments in an office space where there was no interference during the session. This experiment consisted of three parts: a preliminary questionnaire, a test with SolidSketch on a prescribed design task, and a follow-up questionnaire with a semi-structured interview session. After background information had been collected, we provided the participants with a list of features as shown in Appendix A-1. The experimenter reviewed the features one at a time and demonstrated the interactions to the participants before they tried the same feature by themselves. After the training, the subjects read the instructions of the experimental tasks. During the study, each participant was asked to design and build a model of a house within 50 minutes according to the program of a simple house. During the training and experiment session, the subjects were free to ask any questions about the function of

SolidSketch. Figure 7.1 shows the proposed site of the house. A description of the program is provided as follows:

Professor C is a new Georgia Tech professor in the School of Architecture. He just bought a piece of land located in the Midtown neighborhood and would like to hire an architect to design and build a house for starting a new life in Atlanta. The site is roughly 55 feet by 145 feet with the longer side facing north and south as shown in Figure 7.1. The east side of the site has a few beautiful trees and a pond that he would like to retain as a garden. He will need a living room area, a dining space, a kitchen, an office, and four bedrooms including a guest room.



Figure 7.1: The selected site for the task

The goal is to answer the second research question: “How does the combination of direct input and real-time feedback of sketch interactions with the power of BIM affect the early stage of design? Does it facilitate a rapid, flexible, and iterative design process?” In particular, we would like to understand how designers are using the tool, whether they have similar or different behaviors as compared with traditional tools, and why these differences exist. Also, we would like to know what we can improve in terms of software performance and interaction design.

To answer these questions, we created an add-on program to SolidSketch that can log all of the interaction data while the participants are interacting with it. There are four types of logged data: (1) timestamp, (2) interaction event name, (3) location, and (4) other information. We have the following three categories regarding the interaction event name category: (1) touch input events, e.g. “touch down”, “touch up”, and “touch move”; (2) stylus events, e.g. “stylus in range”, “stylus in air move”, “stylus down”, “stylus move”, and “stylus up”; and (3) gesture events, e.g. “tap gesture”, “scratch gesture”, “right angle gesture”, and “lasso gesture”. According to these logged events, we can derive different useful information to analyze the participants’ behaviors. For instance, we can compute the speed of movements when they use the stylus to sketch a shape or perform a gesture. We also know the count of the individual gestures they are using while completing a given task.

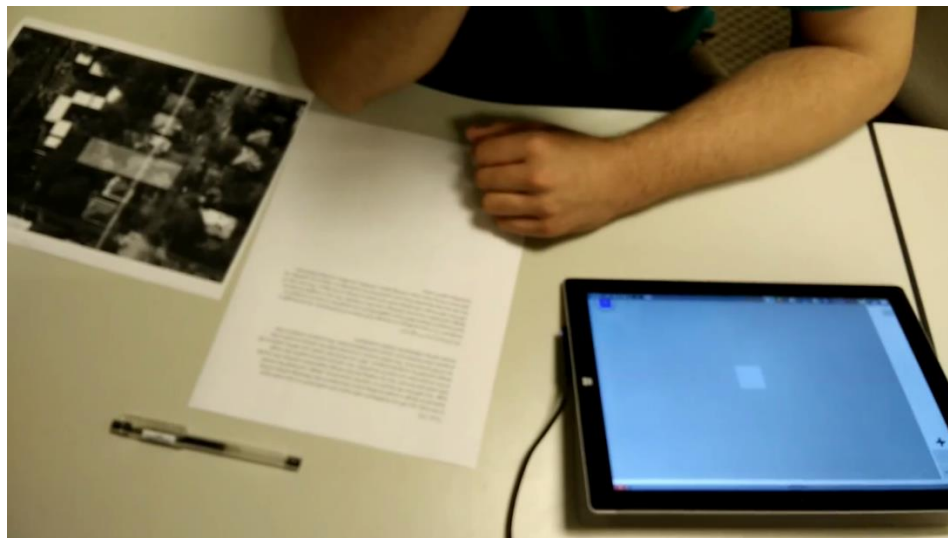


Figure 7.2: The view from the camera that records the participant’s actions during the experiment

To understand the participant’s interactions with the tool in real time, we set up a camera that recorded the entire sessions while the participants were performing the tasks. Figure 7.2 shows the view in which we targeted the camera at the participant’s hands and their workspace, including the tablet and papers with information about the architectural program. To collect interaction details, we also performed screen recording during the sessions. Using an inductive qualitative data analysis

method called Grounded Theory (Adams, Lunt, & Cairns, 2008), we are able to conceptualize the behavior patterns and understand the reason behind these behaviors using a coding scheme, which includes the basic commands that participants used to finish the tasks, and other actions we observed during the experiment sessions. This initial coding scheme included three categories: (A) stylus actions, (B) multi-touch actions, and (C) UI button presses. According to these basic codes, the experimenter iteratively added and edited codes while observing the videos when new or different actions were found. When the experimenter reviewed the videos, we marked the starting and ending time of specific actions to yield the timestamp and duration of the action. The experimenter reviewed these two types of videos several times to ensure that all of the codes were applied integrally among the participants.

Table 7.4 shows the final version of the coding scheme used in this case study of design tasks using SolidSketch. In addition to the first three categories mentioned in the previous section, we added the category D, “On Paper”, with the codes “QuickSketching” (D.a) and “Reading” (D.b). The code (D.a) represents freehand sketching activities on the provided piece of paper, while (D.b) represents actions when participants checked the given program on the paper during the design task. We also added “FalseStylusCommand” (A.g-1) and “FalseMultiTouchCommand” (B.c-1) that record the interaction errors when participants performed wrong actions during the tasks. For instance, during the task, Linda tried to pinch to zoom the view with two fingers on each hand instead of only one hand and confused the system. We coded this event with two codes: (B.c-1) and (B.a-3). In addition, we found that users moved their hands in the air to simulate the actions and locations of drawing before they actually drew the object. We coded these two events as (A.f-1) “InAirMoving.” Finally, we also observed that participants would physically rotate the tablet to make it easier for drawing. These actions are coded as “AdjustTablet” (B.a-5) as shown in Table 7.4.

Table 7.4: The final code scheme used to record the participants' actions

Action Category	Sub-Category	Code
A. Stylus	a. Freehand Sketch	1. Sketch On Tablet
	b. 2D Shape Creation	1. CreateShape
		2. AddHole
		3. OffsetProfile
	c. Shape Modification	1. ModifyShape
	d. 3D Shape Creation	1. ExtrudeShape
		2. CreateWall
		3. CreateHole
	e. Gesture	1. Delete
		2. Lasso
		3. Click
		4. RightAngle
		5. SameLength
	f. In Air Actions	1. InAirMoving
		2. Deciding
	g. Others	1. FalseStylusCommand
		2. CreateArray
B. Multi-Touch	a. Navigation	1. Orbit
		2. Pan
		3. Zoom
		4. TouchWidget
		5. AdjustTablet
	b. Shape Manipulation	1. MoveShape
		2. CopyShape
		3. RotateShape
		4. TouchShape
		5. AdjustArray
	c. Others	1. FalseMultitouchCommand
C. UI Button	a. Undo/Redo	
	b. SwitchMode	
D. On Paper	a. 2DSketching	
	b. 3DSketching	
	c. Reading	
	d. InAirMoving	

After the participants had finished the design task, we gave them a set of questions including NASA TLX (Hart & Staveland, 1988), Creativity Support Index (Cherry & Latulipe, 2014), and our own specific questions with Likert scales to understand their experience using the tool. We also conducted a semi-structured interview to acquire more qualitative data. Please refer to Appendix A-2 for details about the materials we used during the post-experiment sessions. We will report the analysis of this quantitative and qualitative information in the following sections.

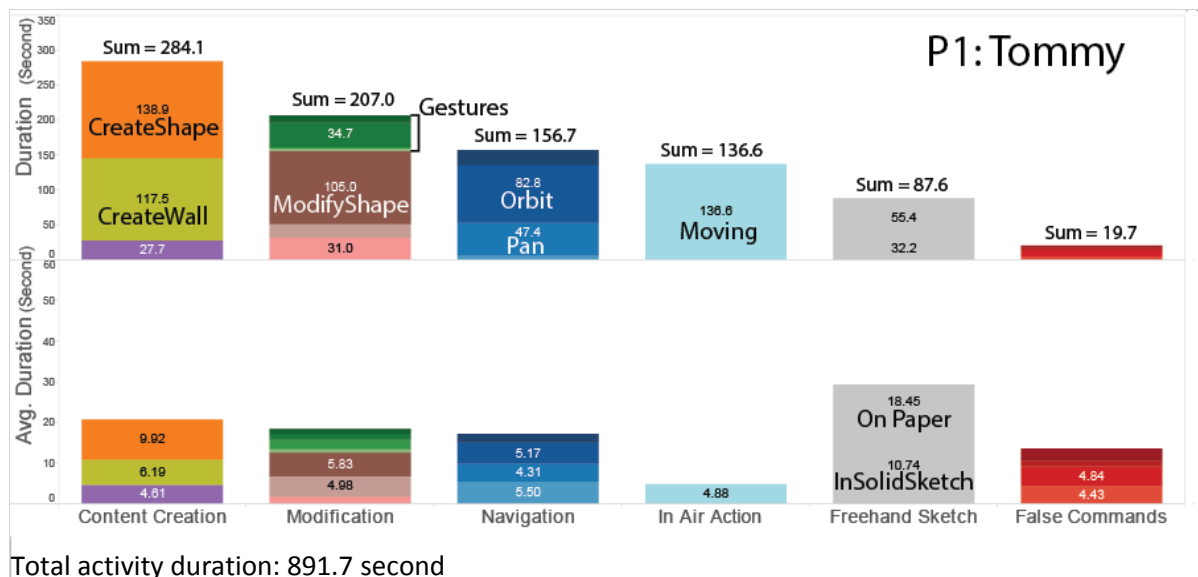
7.3 Data Analysis from Recorded Videos

In Chapters 1 and 2, we argued that the conventional CAD tools do not adequately support the early stage of design according to many previous studies. We also posited that by providing direct

and continuous feedback during interactions, users would be able to employ the tool for tasks in the early stage of design. Through coding the actions in the recorded videos, we collected 835 excerpts from the four participants. Each excerpt lasts approximately 4.1 seconds on average. As shown in Table 7.5, based on the purpose of the actions, we group the codes into six categories: “Content Creation”, “Modification”, Navigation”, “In Air Action”, “Freehand Sketch”, and “False Command”. Figure 7.3 shows the bar charts of stacking the sum and average durations of different codes according to their categories during the experiment sessions. In general, the participants spent most time on “Content Creation”, “Modification”, and “Navigations” during the experiment sessions. The first three participants also spent time on freehand sketching, while the last participants only used 4.7% of his activity duration in freehand sketching. The following sections describe the analysis of the interaction behaviors observed in the video.

Table 7.5: Six categories of activities used to categorize participants’ actions

Activity Categories	Codes
Content Creation	2D Shape Creation (A.b-*), 3D Shape Creation (A.d-*), Create Array(A.g.2)
Modification	ModifyShape (A.c-1), Stylus Gestures(A.e-*), ShapeManipulation (B.b-*), Undo/Redo (C.a)
Navigation	Navigation (B.a-*)
In Air Action	InAirActions (A.f-*), InAirMoving(D.d)
Freehand Sketch	Sketch on Tablet (A.a-1), 2D Sketch on Paper (D.a), 3D Sketch on Paper (D.b)
False Command	FalseStylusCommand (A.g-1), FalseMultiTouchCommand (B.c-1)



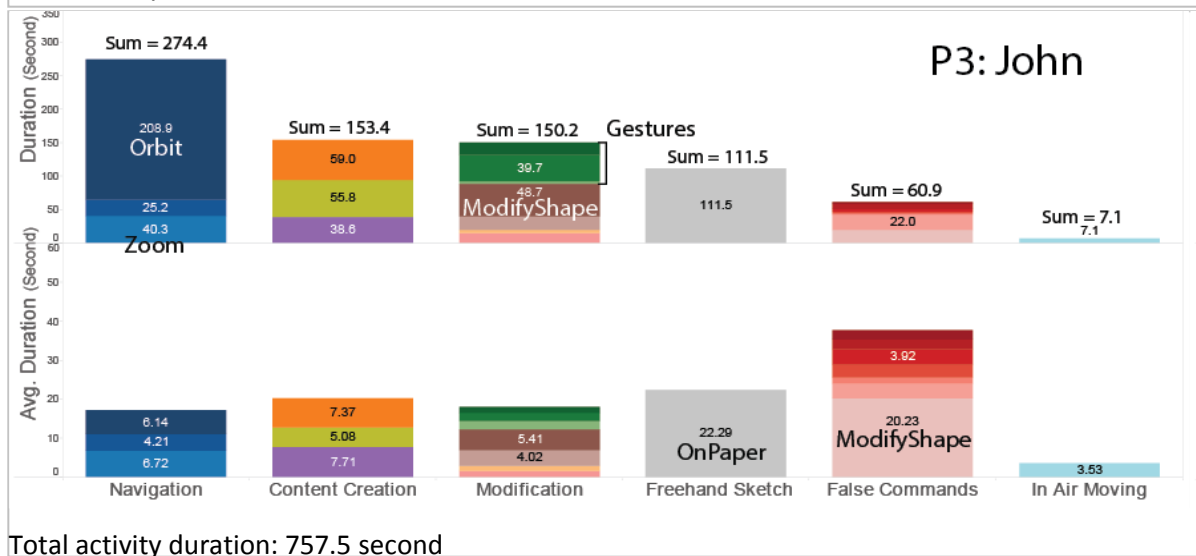
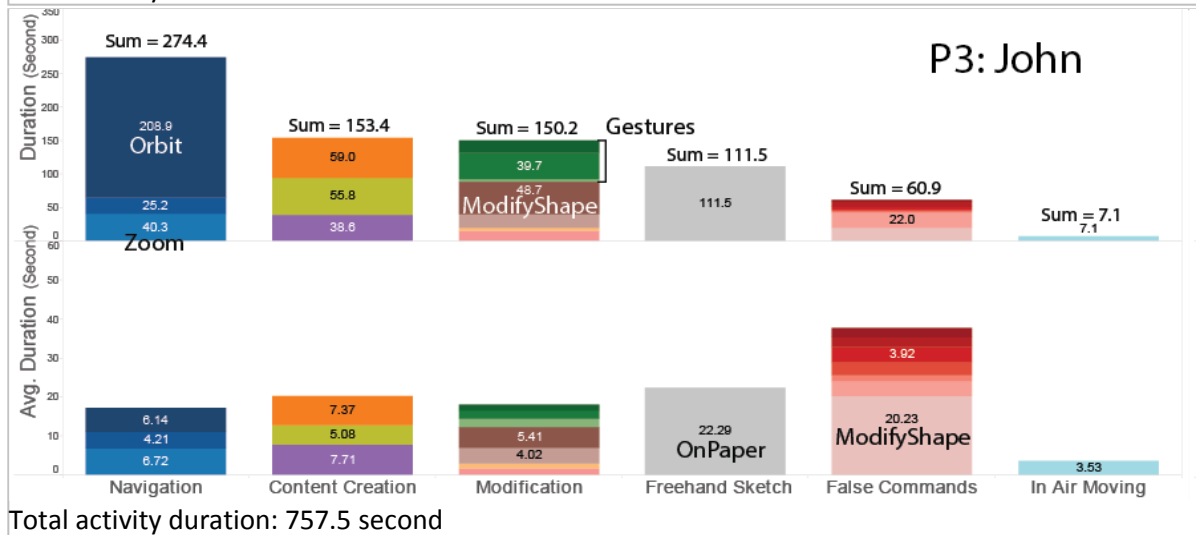
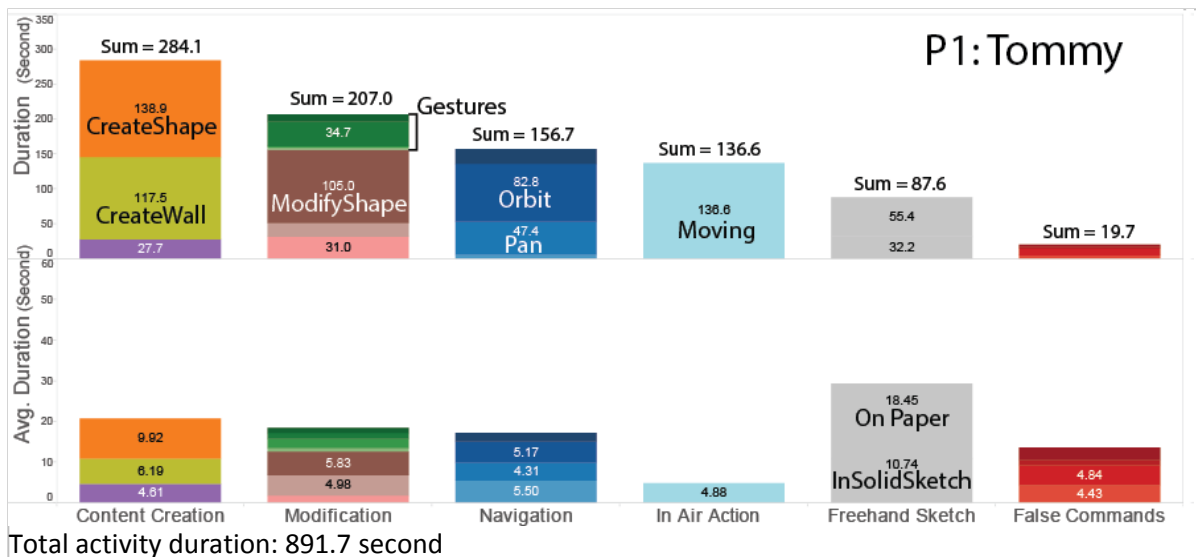


Figure 7.3: The total and average duration of the selected actions observed in the video recordings. This graph is sorted by the total duration of observed behavior category.

7.3.1 The Interaction Sequences Observed from Recorded Videos

As shown in Figure 7.3, all four participants spent large portions of time (Tommy: 17.6%, Linda: 19.8%, John: 36.2%, Robert: 39.1%) navigating around the 3D environment. Through observing the videos, we found that a typical interaction sequence starts and ends with navigation. Navigation through multi-touch interactions served a critical role throughout all of the experiment sessions. At the beginning of an interaction sequence, the participants used navigation (e.g. Pan, Zoom, Orbit, Touch Widget) to find a suitable view and opportunities for adding or modifying geometries on the desired locations. At the end of a sequence, they used navigation to inspect whether the new creations met their desired design intents. All four participants used their fingers to navigate the view to make the environment easier for content creation. During navigation, all four participants were able to use a combination of two fingers to pinch to zoom and pan the view simultaneously. However, the participants sometimes accidentally touched on the existing geometry when performing the multi-touch gestures, labeled as “False Command” in red bars in Figure 7.3, which resulted in incorrect interpretations from the system. For instance, while Linda paid attention to the right side of the canvas, she tried to use her left hand to zoom the view and accidentally touched the existing shapes on the other side. This caused the system to copy the shape instead of zooming the view. We also found that subjects sometimes rotated the tablet physically to orient the view similar to what the designers would do to a piece of paper.

After they found the perfect view for sketching, they may have started to move their hands while holding the stylus to simulate the desired sketching line or the spatial configurations in their mind. These events were coded as “In Air Moving”. These actions were observed in the first three participants (especially Tommy and Linda) but not Robert. In air movements were performed before actions, such as the “Freehand Sketch”, “Create Shape”, “Create Wall”, or “Extrude” actions. These

actions demonstrate that the sketch interactions may encourage embodied motions to simulate the lines they are going to draw in the air during design thinking.

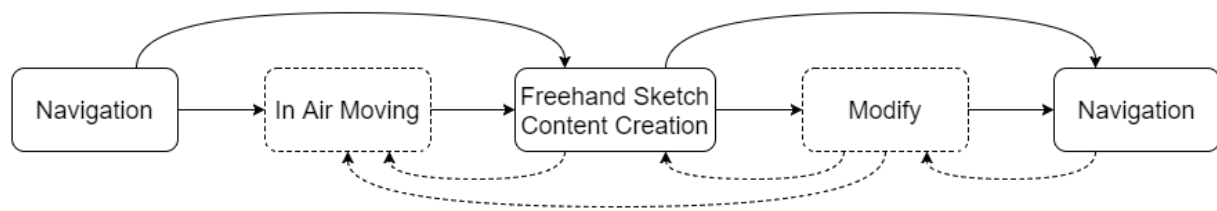


Figure 7.4: An interaction sequence starts and ends with navigation activities with different activities between navigations. Boxes with dashed outlines represent the activities that are not observed in every sequence.

Immediately after these motion simulations, as shown in Figure 7.4, we observed two main categories of activities in video coding: “Freehand Sketch”, either on paper or on the tablet, and “Content Creation”, e.g. “Create Shape” (A.2-1) and “Create Wall” (A.d-2). As shown in Figure 7.3, the “Content Creation” (Tommy: 39%, Linda: 26.0%, John: 20.2%, and Robert: 30.4%) and “Modification” (Tommy: 23.2%, Linda: 18.6%, John: 19.8%, and Robert: 22.6%) events formed approximately 40% to 50% of action durations during the experiment sessions. Linda sometimes stopped her stylus for a few seconds in the air (purple bar in Figure 7.3), coded as “Deciding” (A.f-2), before she put down the pen to create the shapes.

Robert had a different interaction pattern than the rest of the participants. He never used freehand sketching to study the design nor performed “In Air Moving” (A.f-*) to facilitate design thinking. Instead, he used navigations, e.g. zoom (B.a-3) or orbit (B.a-1), between different view angles to examine his designs while using “Create Wall” (A.d-2) for studying spatial configurations. As shown in Figure 7.5, he found that he could use the existing wall for other interior walls in a different view angle (Figure 7.5-2). Thus, he zoomed in to copy the existing wall and move to the desired location (Figure 7.5-4), and then finished the interior configuration with the wall (Figure 7.5-6). Similar behaviors were also observed in the rest of the participants after they had stopped using

freehand sketching and employed the “Create Wall” (A.d-2) or “Create Shape” (A.b-1) features to study their designs.

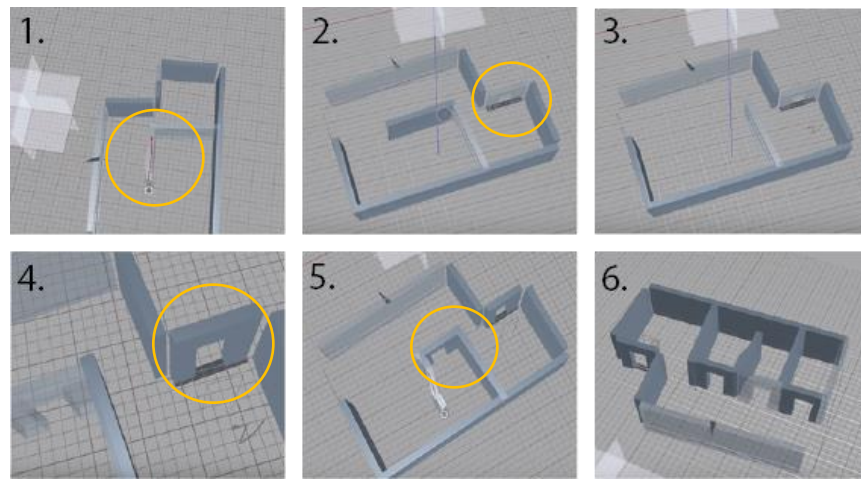


Figure 7.5: (1) Robert created an interior wall in the middle of the canvas. (2) He found out that he can use an existing wall for the wall he just created. (3) He deleted the wall he just drew. (4) He zoomed in and copied the wall. (5) He moved to the desired location and drew another interior wall. (6) He used the wall for other three rooms.

After they had created the geometry, the participants would try to delete (e.g. 7.5-2), modify, or undo the creation. The videos showed that the participants tended to use the scratch gesture to delete creations much more often if they wanted to modify the design (e.g. 7.5-2), while users tended to click on the Undo button when the system produced bad recognitions (e.g. error recognitions on scratch gestures). When the participants found that the created geometries could not satisfy their design intents, they would use “Modification” commands to change the shape. These “Modification” and “Content Creation” commands were sometimes iterative without the changes of viewpoints. Figure 7.4 illustrates these sequences in chronological order.

By analyzing the recorded videos, we found that these interaction sequences mainly served the following three purposes during the experiment: (1) studying and exploring spatial configurations, (2) studying 3D forms, and (3) constructing and editing for precise design drawings. While the first three participants started from the first purpose and then gradually learned to integrate it with “studying 3D forms”, participant 4 mixed the first two purposes together while using SolidSketch during the

experiment. From the video observations, we found different behavior patterns for each purpose during the experiments. The following sections describe the behavior patterns for each purpose.

7.3.2 Exploring Spatial Configurations

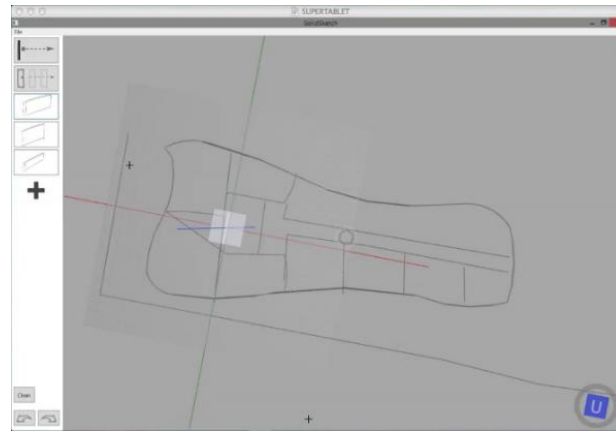


Figure 7.6: Participants 1 to 3 start the design task by quickly sketching out the floor plans for study.

Exploring spatial configurations, such as studying the locations of different rooms, is a typical task in the early stage of design. Designers gradually define design constraints through sketch. All four participants were able to use the tool for sketching 2D lines when studying their design. As shown in Figure 7.6, the first three participants would sketch floor plans to study the spatial configurations. They sometimes used “Create Shapes” (A.b-1) and “Create Wall” (A.d-2) to study the floor plans after they had stopped using freehand sketch. Since the participants could employ these tools to sketch the geometries quickly, they were able to use these geometries to study layout and circulations. Shown in Figure 7.5, Robert, in particular, used the “Create Wall” (A.d-2) tool while studying floor plans from beginning without freehand sketch. Thus, the results of the video coding are significantly different when comparing with the first three participants.

The solid red lines in Figure 7.7 represent the running totals of the activity category “Freehand Sketch”. The solid red lines from the first three participants show that “Freehand Sketch” activities were gradually diminished as they learned to use other tools from the system during the study session, while the line from the fourth participant shows approximately the same rate of using

freehand sketch. It is common for designers to start with sketches to explore design constraints and generate new design ideas. While they gradually reduced the actions of freehand sketching, they used the actions in the “Content Creation” category to study the design further, including “CreateShape” (A.b-1), “CreateWall” (A.d-2), and “ExtrudeShape” (A.d-1). As we found in the video, in addition to exploring spatial configurations (e.g. Figure 7.5), participants also used these tools for studying the 3D forms of the building, which will be described in the following section.

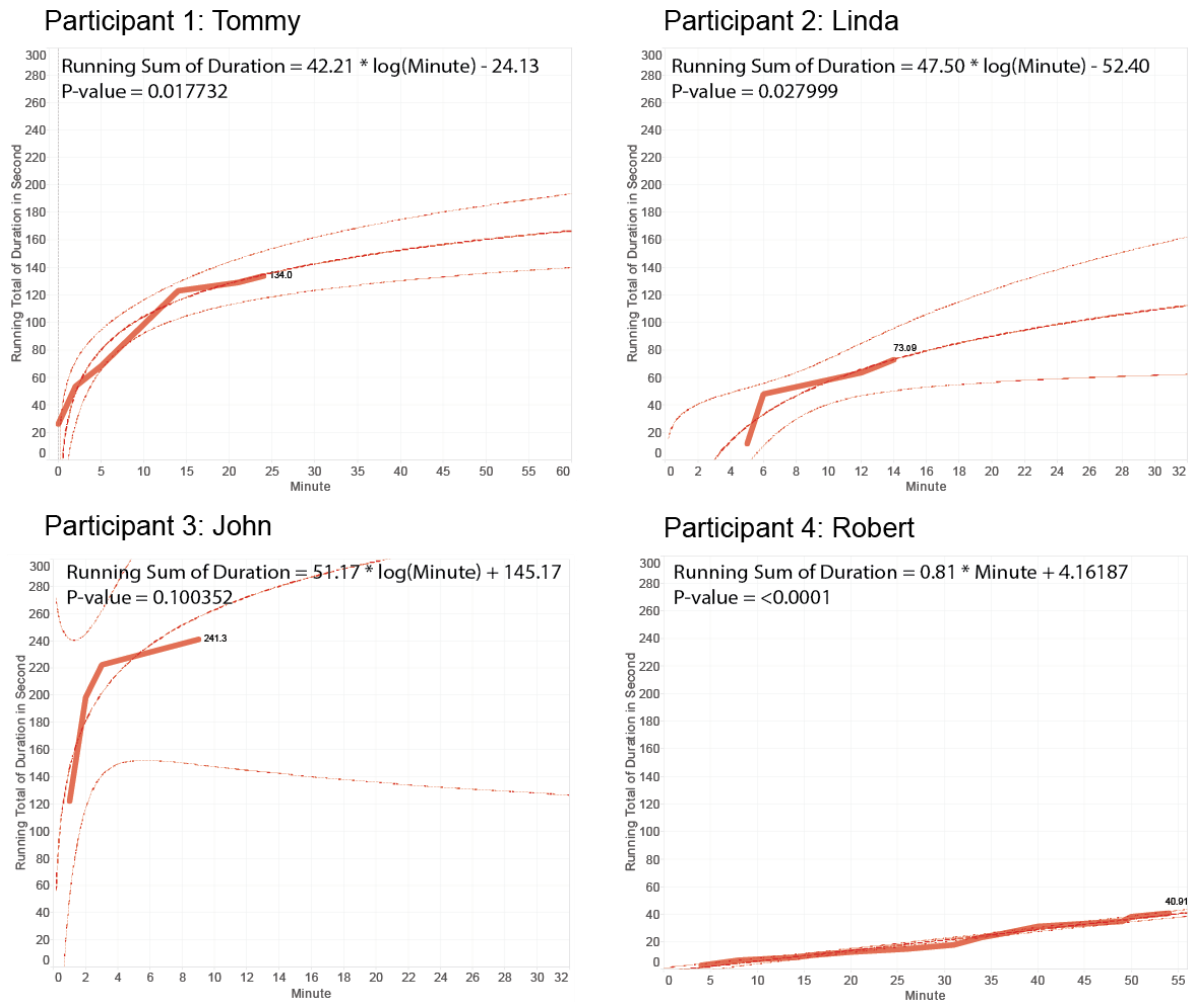


Figure 7.7: The unit of X axis is second. The unit of Y axis is minute. The solid red lines represent the actual running total of duration (in seconds) while the participants use freehand sketch to study spatial configurations during the experiment sessions. The dashed lines represent the fitted curve to the solid red lines. The equations of the dashed curve lines are on the top of each chart.

7.3.3 Studying the Forms of the Building

With the exception of participant 4, the first three participants started to study the forms of the building toward the end of their freehand sketch session, typically no later than the midpoint of the experiment sessions. While studying forms, the first three participants dramatically increased the uses of the 2D/3D shape creation tools that SolidSketch provides. For instance, as shown in Figure 7.9, the blue line and light green line of Tommy's chart indicate the increasing uses of "CreateShape" (A.b-1) and "CreateWall" (A.d-2) at around 12 minutes and 22 minutes, respectively, of his 48-minute study session. The blue line and light green line of Linda's chart indicate her increasing uses of "CreateShape" (A.b-1) and "CreateWall" (A.d-2) at around 16 minutes and 8 minutes of her 30-minute study session. The blue line and light green line of John's chart indicate the increasing uses of "CreateShape" (A.b-1) and "CreateWall" (A.d-2) at around 6 minutes and 10 minutes of his 29-minute study session.

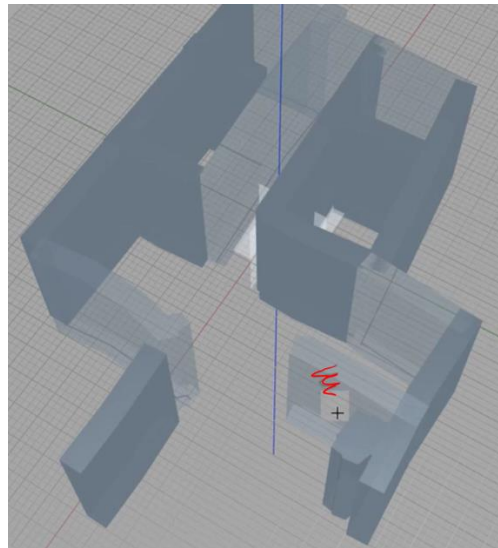
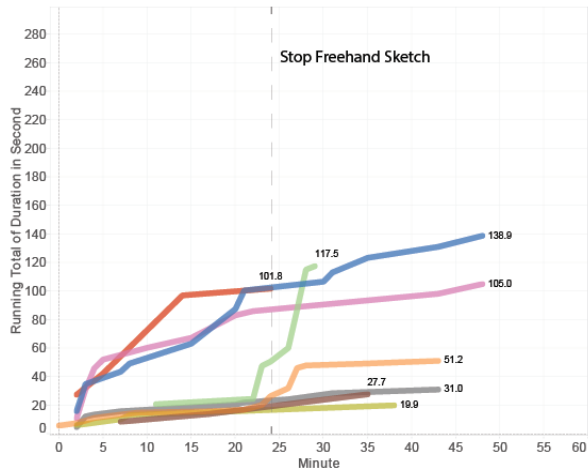


Figure 7.8: Participant 1 (Tommy) studied the floor plan again after he has done some freehand sketch at 25 minute of his study session. The red scribble line shows he tried to delete one of the wall and redraw it.

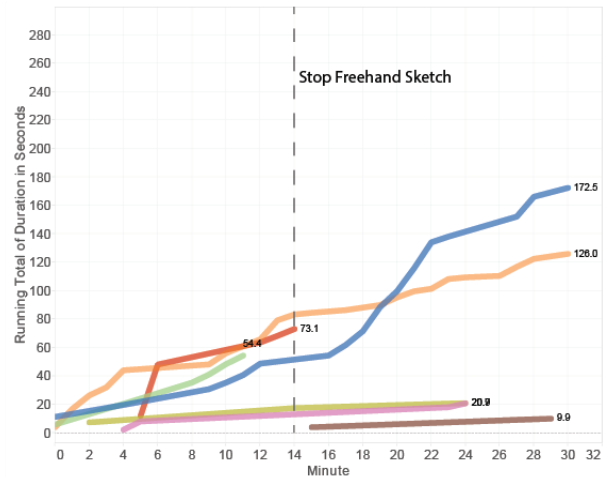
These frequent uses of geometry creation through sketch also boost the usage of employing scratching gestures for deletion or over-sketching for modifications of the geometries. For instance, as shown in the orange line of Figure 7.9, since Tommy needs to ensure that the walls are drawn at the right location and will not cause any additional design issues, he either used scratch gestures to

delete (A.e-1) and then redraw the wall (A.d-2), or used the shape modification (A.c-1) to modify the existing shapes at around 22 minutes into his study session. Figure 7.8 shows that Tommy created several walls and used the scratch gesture to delete one of the walls that was not fit into his design intents at 25 minutes into his study session.

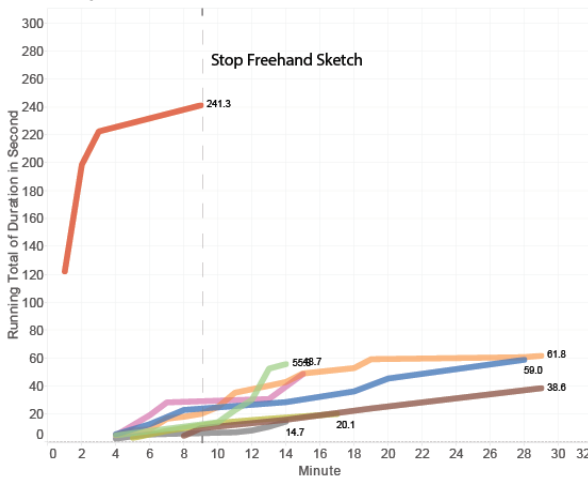
Participant 1: Tommy



Participant 2: Linda



Participant 3: John



Participant 4: Robert

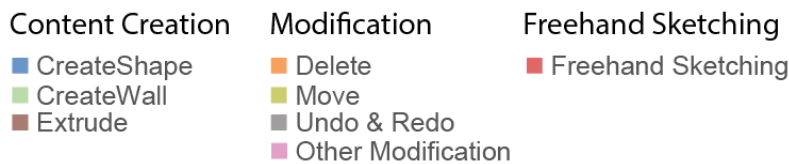
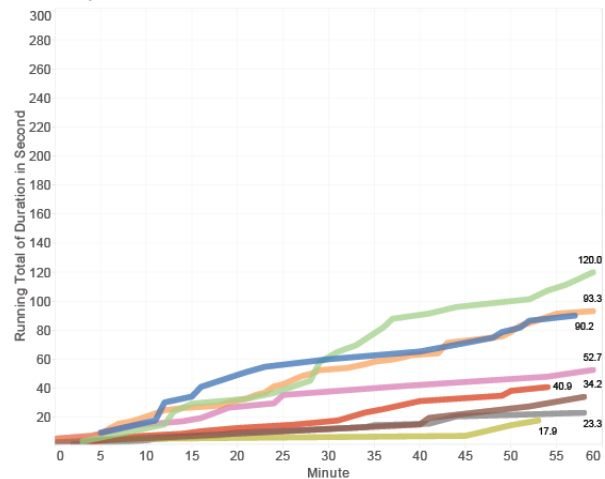


Figure 7.9: The running sum of the duration throughout the entire study. The gray dashed lines in Participant 1~3 indicate the time when they stopped using freehand sketch.

Similarly, as mentioned previously, Figure 7.9 shows that the usages of “CreateShape” (A.b-1) and “CreateWall” (A.d-2) are increased in Linda’s and John’s activities when they are reducing the

uses of “Freehand Sketch”. We found that the participants used these “Content Creation” tools instead of “Freehand Sketch” (both on paper and in SolidSketch) to study the spatial configurations once they became familiar with the tools according to the recorded video. As shown in Figure 7.10, even though we provided the undo and clean buttons for the participants to conveniently reverse their designs, they tended to use scratch gestures much more often (blue bars in Figure 7.10) than these two UI buttons (orange bars in Figure 7.10) when trying to reverse their designs. The participants may use the undo button to reverse the state and redraw it typically in the case of wrong recognition results from the system (~77% of the time⁶). If they were in the flow of designing and exploring the geometries, they typically just used scratch gestures.

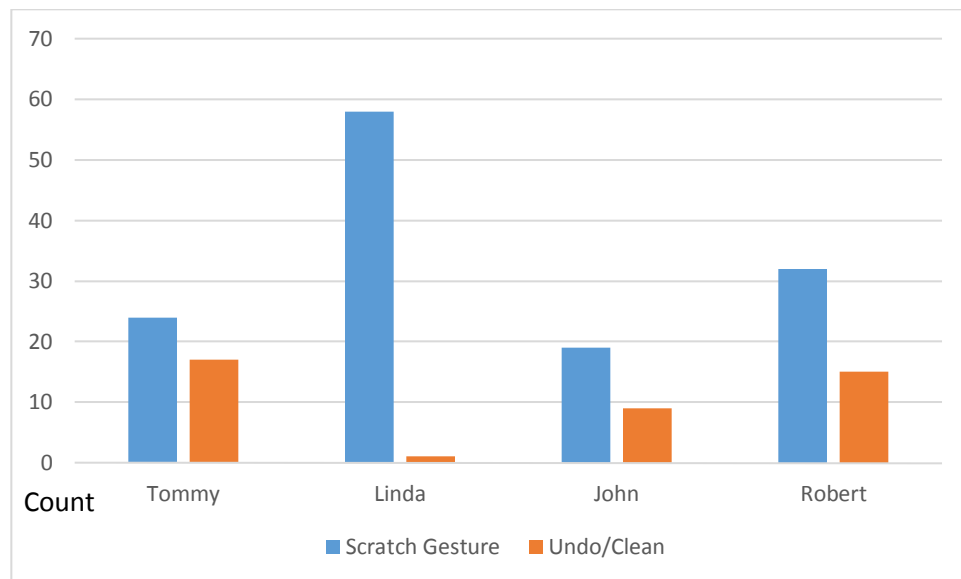


Figure 7.10: The count of using scratching gesture to delete the geometries and undo/redo to reverse the recent creation of geometries.

Participant 4, Robert, had a different behavior pattern in this case study. He was the youngest participant in the study and had good skills of using CAD according to the prequestionnaire survey. In

⁶ We considered the events when the participants had encountered wrong recognition commands within 5 seconds prior to hitting the undo button.

the 25 minutes of his training session, he ensured that he knew how to use every facet of the system's features through trial and error. He was the only participant who constantly used two hands to interact with the system. He was also the only participant who created holes as openings in the experiment session. He directly used "Content Creation" features provided by the system much more often than the other participants during his exploration of the designs without using "Freehand Sketch". During design exploration, he constantly orbited and zoomed the canvas in different viewpoints to create shapes and made sure that his designs fulfilled the needs of both spatial layouts and 3D forms. Thus, there was no significant time period when he used particular features of the system. As shown in Figure 7.9, the running total of the durations for each category are increased equally throughout the experiment session.

7.3.4 Creating Precise Designs

While the system was not built for supporting designs with high precision, we still found that all four subjects tried to use the tool to create accurate designs. These attempts caused some frustration from the participants. According to the modification bars in Figure 7.3, the participants spent some time modifying their designs (Tommy: 23.2%, Linda: 18.6%, John: 19.8%, and Robert: 22.6%). There are two explanations for these modification attempts:

- (1) The participants tried to explore different design possibilities through modifications.
- (2) The system was not able to fully support what they were trying to create.

Through video observations, we found that participants often used over-sketching to modify shapes when studying the spatial configurations and forms of the building. During over-sketching, participants sometimes felt frustrated when trying to convey a shape with precise dimensions to the system. Another source of frustration came from attempting to draw a perfectly straight line between two points as observed in the video, since they tried to start and end right at the grid plane. These actions usually happened at the beginning or at the end of using the system. When they

started to use SolidSketch, it is likely that participants applied their previous experiences of using conventional CAD systems to the study sessions. As shown in Figure 7.11, Tommy, in particular, began to draw the first few shapes with very slow movements to trace the gridlines on the canvas to produce an orthogonal polygon. When the recognition results did not match what he had expected, he modified the shapes with very slow movements once again. At the end of the experiment session, we found that all four participants tried to crystalize their ideas by creating more precise models. While creating a precise model, they sometimes used the scratch gesture to delete existing shapes and redraw them, or they might over-sketch to correct original shapes.

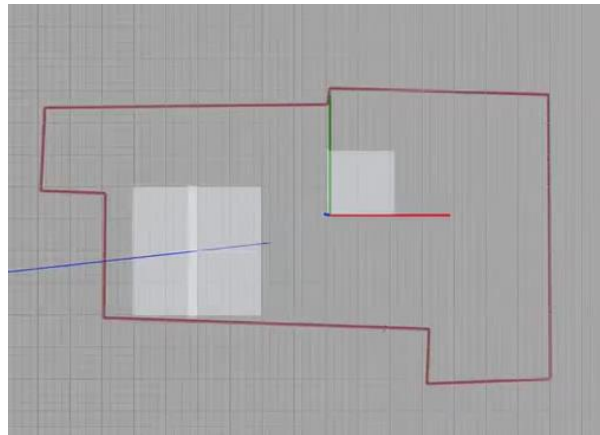


Figure 7.11: The orthogonal shape Tommy created at the beginning of the study session.

7.4 Semi-Structured Interviews

Here, we report the findings from the semi-structured interviews following every regular experiment session. In general, the participants intuitively grasped the technological bridge between physical sketching and digital 3D modeling in SolidSketch. Initially, the participants tended to use their prior experiences of using conventional CAD tools when using SolidSketch. P2 stated, *"I started adding stuff, and then I started realizing, I can be more creative, because earlier I just think of steps"*. As they used SolidSketch, the participants gradually became aware of the increased freedom and flexibility it offered and thus gradually reduced their extent of freehand sketching. Two participants (Tommy and Linda) explicitly noted the lack of flexibility in popular modern CAD programs, such as

SolidWorks and SketchUp, compared with SolidSketch. Tommy stated, *"I'm not a big fan of SolidWorks....it's a great tool, but I don't have much freedom...I feel exhausted after working in Solidworks both physically and mentally."* Linda noted that *"... even if I do SketchUp, that will harm me in the long run, double lines... But in this case [SolidSketch], if it's only just a sketch, it doesn't really matter."* When Tommy compared his experience with SolidSketch interactions with that of traditional CAD interactions, he noted that he found it easier to accomplish certain common exploratory actions with SolidSketch. In particular, Tommy reported that shape-editing tasks, such as joining and modifying shapes, were much more complex in a traditional CAD interface than in SolidSketch, saying *"Sometimes [shape editing] really requires a lot of tools and a lot of moving of vertices and points to make sure things are straight."* With SolidSketch, conversely, the participant *"wasn't stressing about making a straight line, or whether I'm going to be this way or that way, in SolidSketch you just select these two and you make relatively quickly 'right angle' and 'same sides' [gestures]."*

John noted that he felt more engaged because *"it [SolidSketch] respected your unique line style"* that he worked hard to craft throughout his years of training as a designer. He explained his view of personal sketching line style by saying *"It's kind of like a fingerprint...everyone has their line....that's what I find most interesting from the program [SolidSketch]...that's what keeps you engaged."* The *"organic feel"* of the lines offered by sketch-based shape editing was exciting to John and increased his engagement with the tool. Overall, designers were excited about the possibility of integrating their physical sketching practices into the digital realm by using SolidSketch and provided many interesting ideas for extending SolidSketch in the future.

As described in the previous sections, we found behavioral differences between the first three participants and the fourth participant. The first three participants thought in 2D when exploring the design possibilities instead of 3D, while Robert rarely used "Freehand Sketch" prior to the uses of 3D

geometry creation features. In the version we used for the study sessions, we did not include freehand sketch mode in the system. Thus, Tommy reported that *“I think it would be easier if you think in a 2D way, and then be able to make it 3D, otherwise I have to think about how will I make this, rather than ...oh I have a sketch, let me make it 3D”*. Linda also suggested that we could add a sketch mode without any recognition from SolidSketch. Based on these comments, we have added a freehand sketch mode in SolidSketch as shown in Figure 4.4 that enables users to quickly sketch out their ideas without any interpretation in the system. On the other hand, Robert sketched naturally to create geometries in 3D spaces. He said *“I thought it was very nice for sort of quick sketch that is something in 3D and just making a shape for the sake of that. That was really nice. I felt good response to it.”*

7.5 Results and Discussion

Figure 7.12 shows that the subjects were able to design a building within the 50-minute time frame. During the experimental task, the four subjects used the tools of SolidSketch differently. Some participants, especially Tommy and Linda, initially treated it as a conventional CAD program but later became aware of the flexibilities that the sketch interactions provided. For instance, they tried to maintain good practices to the extent possible in the model since they had previously created high-quality models using conventional CAD programs. In addition to the precision mentioned in the previous sections, Linda also described that she tried to avoid multiple lines stacking together. However, all of the subjects were eventually able to use this program as an exploration tool. Tommy, John, and Robert even tried to experiment with the layout of the floor plans directly on the canvas using the “Create Wall” tool (A.d-2). As shown in the drawing in the upper right corner of Figure 7.12, Linda draws 2D sketches lines first, and then uses the “Create Wall” tool to trace those lines and draw walls on top of it. The follow-up interview provided more details about how participants

designed the house with SolidSketch and corroborated the findings, e.g. interaction sequences, usages of freehand sketch, and embodied thinking, developed through the logged data and video coding. As shown in Figure 7.12, the participants were able to create meaningful 3D models within 50 minutes (Tommy: 48 minutes, Linda: 30 minutes, John: 29 minutes, and Robert: 50 minutes). While creating the models, they were not aware that they were embedding parametric or basic semantic information simultaneously.

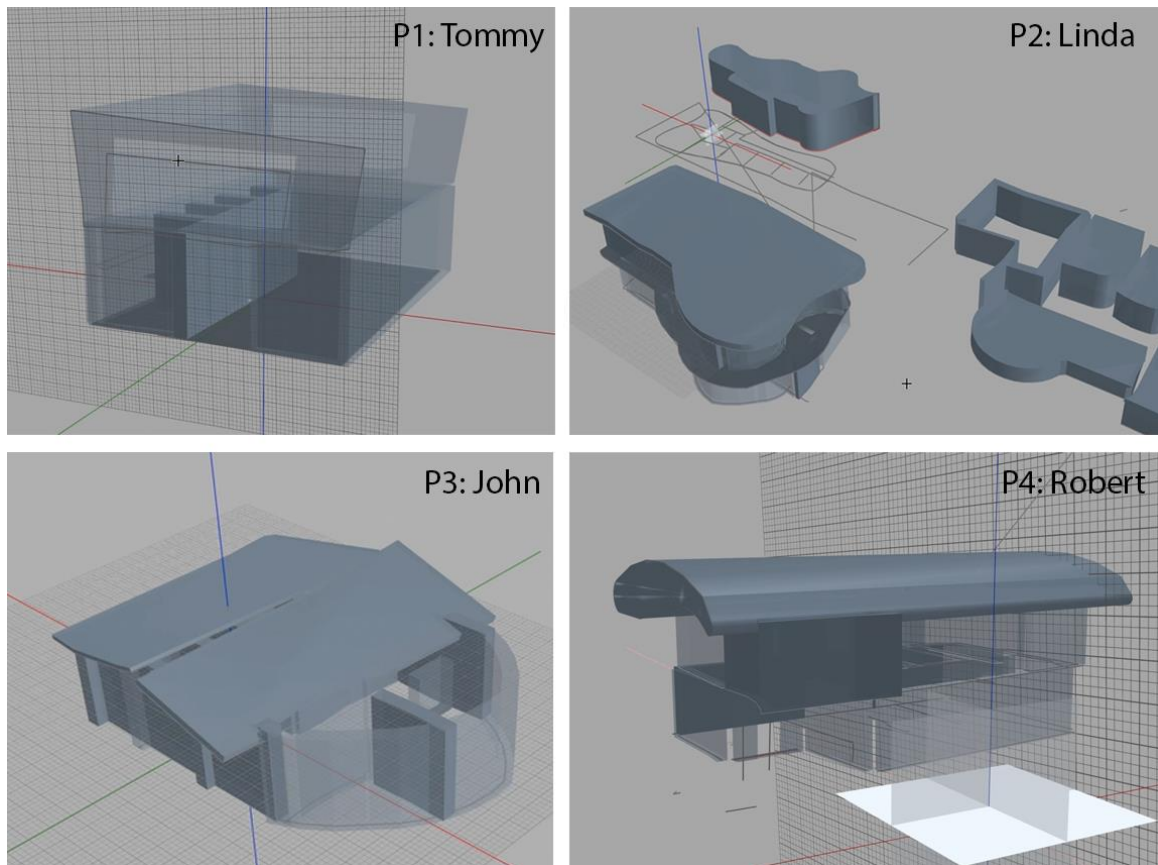


Figure 7.12: Designs of simple houses using SolidSketch from the four professional participants

7.6 Summary of the Dissertation

We started this dissertation by asking the following three questions:

(1) How can we combine the direct input and real-time feedback in sketch interaction with the computational power of BIM in one tool?

(2) How does this combination affect the early stage of design? Does it facilitate a rapid, flexible, and iterative design process?

(3) What techniques could be used to disambiguate sketch input in the 3D environment? How can we create parametric geometries and relationships between geometries through sketch/multi-touch interactions?

To answer the first question, we started our investigations by asking the following two questions:

(1) how can we create a design tool that can fit into the modern design workflow that adopts BIM tools in the later stage of design? And (2) what are the most important aspects in the traditional design tools, such as pen and paper? To answer the first question, we discussed the importance of interoperability in the first part of Chapter 2, and used Concept Design BIM 2010 (CDB10) as a starting point for our use case. To answer the second question, we first reviewed the modern cognitive theories, including distributed cognition, situated actions, and embodiment. These modern cognitive theories suggest that direct and continuous interactions are critical to the early stage of design when designers need to take the design constraints and emerging ideas into account for solving ill-defined problems. We then created sketch and multi-touch interactions that provide continuous feedback, including generating geometries from a “prototype template”, while the user is moving his stylus.

How does this combination affect the early stages of design? The results of the study and comparisons earlier in this chapter show evidence that the experiences of using SolidSketch were close to experiences with traditional design tools. In other words, SolidSketch offers rapid design creation, provides a flexible environment for manipulating geometries, and supports iterative design processes as shown in the first three rows of Table 7.6. Table 7.1 shows that SolidSketch requires fewer steps for modifying the 2D and 3D geometries than conventional CAD systems, and about the same steps comparing to other sketch prototypes. The case study further shows that the participants

can rapidly complete design tasks within an hour. Sections 7.3.2 and 7.3.3 reveal further details of how users are able to quickly change the drawings based on their emerging ideas, including the example shown in Figure 7.5. The interaction sequences discussed in Section 7.3.1 show that users can iteratively change their designs and geometries. The duration of using freehand sketches are generally reduced over time during the study sessions, while the durations of using features provided from SolidSketch in the category “Content Creation” increase. The participants used the system tools that continuously generate the geometries and change the viewpoints to validate layouts of the floor plans as well as studying 3D forms. Based on the suggestions from the participants, we have included the freehand sketch mode in the system to help designers sketch out their ideas freely in 3D spaces in a way similar to traditional sketching.

Table 7.6: SolidSketch in the criteria from Table 2.1

Criteria	SolidSketch
1. Low-Cost and Rapid Design Process	<ol style="list-style-type: none"> 1. Users draw lines without thinking what commands they need to execute 2. SolidSketch recognizes segments automatically 3. Users employ their fingers to directly control the viewpoints and manipulate the shapes 4. Freehand sketch mode for quick idea explorations
2. Flexible and Iterative Design Process	<ol style="list-style-type: none"> 1. SolidSketch enables over-sketch to modify both polylines and polygons 2. Users can stay in the flow while using sketch and multi-touch interactions; they do not need to use GUI buttons to achieve certain tasks 3. From video observations, users can switch between different modes, e.g. explore functions of design while studying the forms of design
3. Direct and Continuous Interactions	<ol style="list-style-type: none"> 1. Users employ continuous movement to create designs similar to traditional tools instead of specifying key points like CAD tools 2. Users are free to use their hand movements in the air to simulate their emerging ideas 3. The system tries to continuously generate geometries while users are moving their hands
4. Constraint Assignment	<ol style="list-style-type: none"> 1. Users can manually assign constraints by drawing gestures, e.g. right angle, ticks, or lasso. 2. The system automatically infer some constraints such as on surface and connect to.
5. Construction of Generative Rules	<ol style="list-style-type: none"> 1. Template composition mode that enables the users to create generative rules
6. Design Automation	<ol style="list-style-type: none"> 1. Drawing a stroke with a “prototype template” will lead to automatic generation of geometries
7. Interoperability	<ol style="list-style-type: none"> 1. The system saves the data as IFC format with MVD similar to CDB 2010.
8. 3D Geometry Creation	<ol style="list-style-type: none"> 1. Users can create extrusion and swept solid through sketch 2. Users can specify 3D drawing planes by manipulating axis widget 3. Users can sketch freely on any surface of the geometries

Chapters 5 and 6 introduced ways of inferring users' intents from the sketch and multi-touch interaction signals, and then translating them to 3D geometries and their relationships in machine-readable data structures. The parameters of the geometries and the relationships among them were parsed in the background while users design and create their models. These data can be easily mapped to a neutral format, IFC, for other BIM programs, e.g. Revit, to read. The system takes the sketch and multi-touch inputs and constructs the data by building up meanings from segmenting sketch strokes. Through segmenting a sketch stroke into line and curve segments, the system is able to construct precise parametric lines as building blocks of a building model. As shown in Table 7.6-4, the system employs the user's stylus gestures, such as right angle, ticks, and lasso, with automatic inferences for creating constraints and geometrical relationships.

As shown in Table 7.6-5, these constraints and relationships can be used in template composition mode for constructing generative rules with operations, which compose the rules similar to those used by a generative design system. When using a wall template with windows, such as the one shown in Figure 6.12, users can quickly draw a stroke and let the system automatically generate the rest of the geometries according to the operations, as pointed out in Table 7.6-6. These walls generated from this wall template with object relations, such as "a solid shape on top of another solid shape", can then be used for inferring basic semantic information, such as slabs and openings, when creating IFC files that conform to the MVD we simplified from CDB10, as mentioned in Table 7.6-7.

However, some issues remain that need to be addressed in the near future. For instance, the participants reported that they felt frustrated when they wanted to specify precise drawings. Selecting objects behind objects is sometimes frustrating if there are no preset rules such as those shown in Figure 5.11. In addition, we only implemented the recognitions of basic building elements in the current version of SolidSketch, e.g. slab, wall, roof, and building stories. The current prototype

also does not support simulations, such as energy analysis and building circulation analysis.

Furthermore, the current operations for prototype templates are limited to simple rules. In the next chapter, we will present the most important improvements that need to be addressed in the near future.

Chapter VIII. Future Work

In the previous chapters, we described the SolidSketch prototype and discussed its strengths and weaknesses. We provided evidence that SolidSketch can support design thinking and ensure smooth transitions between tools in the early stages of design. The data generated in SolidSketch are structured in a way that can be utilized in Building Information Modeling (BIM) software. However, the expert users who were selected as study participants have also suggested several directions that this research could take in the near future. Also, we can employ more automatic procedures in SolidSketch to generate comprehensive semantic data for a variety of purposes. These data structures also enable data exchange for further utilization of other software. In this chapter, we discuss suggested future work in two main parts: (1) Improving interactions; and (2) Incorporating more computational power.

8.1 Improving Interactions

Certain frustrations were observed in the case study sessions. By removing these frustrations, we can improve the users' experiences, and thus provide a better system for designers to use in early stages of design. For instance, one frustration comes from the wrong recognition of users' intents. While the system takes every facet of a user's interaction signals into account when making decisions, it sometimes also includes a user's unintended actions. One issue observed during the experiments is that the participants sometimes intended to navigate the view, but accidentally touch the existing geometries, and thus caused other actions, e.g. moving or copying those shapes. We think that we can add a vibration feedback for users to be aware that they are touching an existing object. There were additional interaction issues observed during the experiments and further possibilities of the system by engaging the interactions employed by the system. We will discuss them and provide potential solutions in the following paragraphs.

8.1.1 Improving Precision in Sketch When Necessary

One complaint from participants is that they were not able to create precise drawings with exact dimensions. For instance, participants sometimes tried to draw a straight line. However, the system sometimes recognized the raw stroke as a combination of two line segments. In addition, as mentioned in the previous chapter, there are two explanations for participants frequently modifying shapes using over-sketching: (1) to explore and iterate designs, and (2) to achieve more precise drawings. While participants felt comfortable using over-sketching for exploration, they often felt frustrated when trying to make precise drawings similar to what they would achieve in a CAD system.

As discussed in the previous chapter, we found that participants used SolidSketch to accomplish three tasks: (1) exploring spatial configurations, (2) studying 3D forms, and (3) creating precise designs. Even though it is not necessary to create precise drawings in the first two tasks, we found that participants still required precision for the third task. In the process of developing SolidSketch, we recognized this problem and tried to infer the sketch precision settings solely using the speed of the stroke. However, findings from an informal study suggest that users may not employ a particular speed for different degrees of precision.

Since these tasks are roughly in the chronological order during the 50 minutes of the testing period, the system may be able to predict what tasks they are working on at the moment while using SolidSketch to design. According to other contextual information, such as in what region they are sketching on the canvas, what other elements are near the current drawing area, and the speed of the stroke, the system may be able to find clues and adjust the parameters automatically through machine learning classifiers to decide whether users are creating rough and sketchy drawings, or precise drawings during recognition.

We may also be able to use a combination of multi-touch and sketch gestures to solve the problem. As proposed by Robert, *"It would be just as easy as laying fingers on something (like using*

combination of touch and stylus)". In fact, a similar idea has been implemented by the Pen + Touch prototype (Hinckley et al., 2010) that enables users to create straight lines. However, since the device and software environment we are using for building SolidSketch does not allow simultaneous pen and touch inputs, we are not able to create this type of interaction without hacking the device's drivers.

There were other suggestions from the participants regarding recognizing users' intentions while they were trying to draw precise drawings. For instance, Tommy would have liked to draw round corners on certain shapes. It is possible that users can sketch a small arc segment at the corner of the shape for the system to recognize it as a round corner. However, we have to carefully design this type of interaction and avoid adding too much burden on remembering gesture commands for more features. Some participants also requested precise dimensions. We think that we can also employ text recognition to achieve this, similar to what other sketch programs do, e.g. SIMI (Johnson et al. 2012).

8.1.2 3D Object Selection

We also found that the participants navigated the views very often during the interactions. We have explained the interaction sequences in the previous chapter, which usually start and end with touch-based navigations. While we found that navigation is important for exploring other opportunities and inspecting designs, we still recognize that users would like to stay in the same view while operating on objects situated in the background in the same task, i.e. behind and occluded by objects in the foreground. Even though the system enables intelligent filtering through the process of checking contexts when users perform a gesture, participants were still frustrated when they were trying to perform a simple action, such as selecting an object situated behind another object. Thus, users would need to orbit the view prior to choosing the target object. John suggested, *"maybe if there was a way to cycle...if I have a geometry here, here...to cycle to pick the geometry...because*

once it gets more complicated...I have a plane here, a plane here, so I'm not rotating so much to try to grab it." We believe that we should be able to resolve this issue by using multi-touch or sketch gestures, such as double-tapping or applying harder pressure on a stacking of objects for enabling the selection cycles that John mentioned.

8.1.3 Embedding More Semantics

Since SolidSketch knows the topology of the geometries on the canvas, it should be able to infer and embed a basic level of semantics on the geometries. At the moment, SolidSketch can infer simple building elements from the topologies, such as walls, slabs, and openings. However, more semantic recognition should be possible by further leveraging context with topologies. For instance, if users define a final product of the geometry as a chair, the system will be able to automatically assign a chair body, a backrest, a seat rest, and legs through rule-based algorithms. More complicated algorithms could even identify complicated building elements in a model highly accurately (Cantzler, 2003). The sketched lines from the freehand sketch mode can be used to cross-check uncertain semantics of the geometry or add further details to the recognized building elements. For instance, if the system has high confidence in recognizing brick-like patterns on a geometry with uncertain semantics, the system can set this geometry as a wall. With the size and the shape of the pattern, the system could even associate various types of brick wall information with it. In other words, freehand sketches and 3D geometries with topological information can provide evidence that help the system to recognize the semantics of the model accurately. Once the model contains this type of semantic information, we should be able to increase the effectiveness with which the computational powers of CAD can be fully integrated into naturalistic sketching interactions.

8.1.4 Additional Ways to Build Template Modules

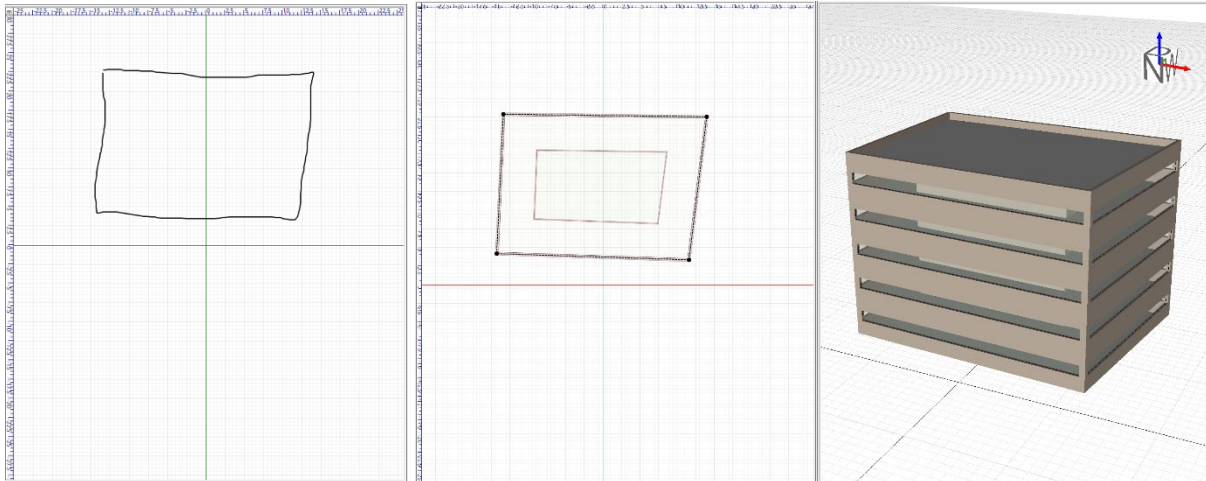


Figure 8.1: After selecting a two-dimensional template, the user can quickly sketch a closed polygon (left) to generate the template floor plan and building (right).

Currently, we enable a one-dimensional template-building procedure. Once users enter the building template mode, they employ multi-touch and sketch actions to interact with the pre-existing line on the canvas to create the template with rules and constraints as described in Chapter 4.2. The system records these interactions and gradually builds a template that designers can subsequently reuse. However, the same design rationale and interaction methods could be applied to two-dimensional template-building procedures that would enable designers to build their templates to create a myriad of floor plans. For instance, as shown in Figure 8.1, a user could create a floor plan with a perimeter and core zones. Designers often use this type of building plan to evaluate energy consumption at specific locations. Users could quickly create and modify shapes and feed the information into a simulation engine to collect data about energy consumption and to study design alternatives that address any issues.

8.2 Incorporating More Computational Powers

8.2.1 Interoperability

SolidSketch produces data that include basic building elements such as walls, slabs, and openings. It also includes abstract entities such as sites, buildings, and building stories. The semantic properties

of these objects are also beneficial for data exchange. As mentioned in Chapter 2, without sufficient information in a proper data structure, a CAD or BIM program would not be able to further augment an architect's intellect. For instance, to avoid the sun reflection from a façade, a program must have the material information of the façade. To compute the interior physical environment, a program must know the materials of the windows and the properties of the walls. This semantic information must be rigorously exported according to corresponding MVDs, e.g. CDB10.

In addition, there are two types of data that are not supported by IFC. The first one is comprehensive design constraints that describe the relationships among geometric objects. The current IFC schema supports the constraints for later stages of design, such as rebar and reinforced concrete to facilitate the communications between different engineering software. Unfortunately, the latest version of IFC, IFC4, does not contain a definition of parametric constraints for 2D and 3D geometries for exchanging generative rules.

The second type of data not supported by IFC is the data related to users' freehand sketches. It is possible that we could implement a data exchange for these sketch drawings by attaching bitmap files. However, bitmaps only capture a snapshot of the current design drawings. In this way, it is not possible to recover the sketch data when users want to exchange the design back and forth between SolidSketch and other programs. Since increasingly many designers start their rough sketches on their tablets, it is necessary to add new classes of entities in IFC to include these types of data throughout the design workflow. To solve these issues, we will need to define our property sets at the moment. However, a better approach is to solicit agreements from software vendors and developers so that this information can be exchanged properly and seamlessly.

8.2.2 Simulations

Accessing the results of simple simulations quickly is critical in the early stages of the design process. A designer would be able to modify a design in the early stages before incurring the high

costs of design errors discovered in the later stages of the design process. Typically, the results of simulation remain unknown until comprehensive geometries and information are constructed in the later stages of the design process. This dissertation has presented a promising system that could provide such computational power in the early stages of design. SolidSketch provides designers with a possibility of seamlessly evaluating their designs with simulations. For example, since the system understands relationships between geometries, it could compute emergency plans for a building in the near future. If a designer wishes to add rooms or modify floor plans, the system could continuously report whether the newly added sketched lines will create problems or violate regulations. In addition, the system could incorporate information from the designer's initial building program, such as a business process model or spreadsheets listing building specifications. The system could heuristically display contextual hints about whether the designer is following the pre-specified information.

8.3 Conclusion

The early stages of design dramatically influence the end products of a building. Design errors during the early stages of design often incur large costs if they are discovered in the later stages of design. If designers can identify a problem as early as possible, they will be able to avoid many different issues and lower the costs of these design errors. During the early stages of design, a line from the designer's pen would result in a building object, such as a wall, a slab, or a space, or a design intent, such as circulations or views. In this dissertation, we started from the goal of making the workflows smoother in the early stages of design and reducing the needs of switching between different tools by recognizing the designer's intents and generating geometries continuously with their finger movements to enable a rapid, iterative, and embodied design environment. The design data generated in SolidSketch can then be used in other BIM programs, such as Revit. Thus,

SolidSketch creates opportunities for designers to trace how the building was designed during the early stages. In other words, when designers used SolidSketch for their design explorations, they had much more access to how this line would affect the quality of space when it is built. While many researchers have developed tools and standards to extend the building information models in the later stages of design to improve the efficiency of building construction, this dissertation addresses a gap in the early stages of the design process and enables the possibility of extending building information models for design explorations.

APPENDIX A Case Study Materials

1. Training and Tutorial Session

In this session, the experimenter will conduct a series of tutorials that cover the basic interactions of SolidSketch. These tutorials include the following:

- 1. Navigation**
Use conventional multi-touch gestures to pan, zoom, and orbit the view
- 2. Sketching a 2D shape on a 2D plane**
Sketch a line and a closed polygon
- 3. Editing the existing 2D shapes**
Over-sketch on 2D shapes to edit the shape
- 4. Defining 2D constraints**
 - a. Create the same lengths between two line segments
 - b. Make right angle constraints on a corner
 - c. Trace the existing lines
- 5. Extending 1D line segment to 2D shapes**
 - a. Select short line segments
 - b. Hold the button on the stylus while drawing a stroke
- 6. Extruding the 2D shape to 3D shapes**
 - a. Select a closed 2D polygon
 - b. Hold the button on the stylus while drawing a stroke
- 7. Erasing an existing shape**
Perform a scratch gesture on a shape
- 8. Making a hole on a 3D shape**
 - a. Draw a closed polygon on the surface of a 3D shape
 - b. Select the 3D shape
 - c. Use scratching gesture to create a hole
- 9. Moving and rotating the shape**
 - a. Drag the shape to move
 - b. Hold the axis on one finger and drag on the other to rotate
- 10. Creating Booleans between two 3D shapes**
 - a. Move and rotate a 3D shape to intersect with another 3D shape
 - b. Select a 3D shape
 - c. Perform scratch gesture on another shape
- 11. Defining a new drawing plane through lasso gesture**
 - a. Perform a lasso gesture on a shape
 - b. Tap on the widget
- 12. Create an array of shapes**
 - a. Tap on the array button to change the mode
 - b. Select a shape
 - c. Hold the stylus button while drawing a stroke

2. Post-Participation Questionnaire

1. How mentally demanding was the task?

1	2	3	4	5	6	7
Very Low			Neutral			Very High

2. How physically demanding was the task?

1	2	3	4	5	6	7
Very Low			Neutral			Very High

3. How hurried or rushed was the pace of the task?

1	2	3	4	5	6	7
Very Low			Neutral			Very High

4. How successful were you in accomplishing what you were asked to do?

1	2	3	4	5	6	7
Perfect						Failure

5. How hard did you have to work to accomplish your level of performance?

1	2	3	4	5	6	7
Very Low			Neutral			Very High

6. How insecure, discouraged, irritated, stressed, and annoyed were you?

1	2	3	4	5	6	7
Very Low			Neutral			Very High

7. I was satisfied with what I got out of the system or tool

1	2	3	4	5	6	7
Highly Disagree			Neutral			Highly Agree

8. It was easy for me to explore many different ideas, options, designs or outcomes, using this system or tool.

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral				Highly Agree

9. I would be happy to use this system or tool on a regular basis

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral				Highly Agree

10. I was able to be very creative while doing the activity inside this system

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral			Highly Agree	

11. My attention was fully tuned to the activity, and I forgot about the system or tool that I was using.

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral			Highly Agree	

12. I enjoyed using this system or tool.

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral			Highly Agree	

13. The system or tool was helpful in allowing me to track different ideas, outcomes, or possibilities.

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral			Highly Agree	

14. What I was able to produce was worth the effort I had to exert to produce it.

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral			Highly Agree	

15. The system or tool allowed me to be very expressive.

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral			Highly Agree	

16. I became so absorbed in the activity that I forgot about the system or tool that I was using

1	2	3	4	5	6	7	8	9
Highly Disagree				Neutral			Highly Agree	

3. Semi-Structured Interview Questions

1. What would you change about the sketch-based interface to make your experience better?
2. What task was most difficult and confusing for you to accomplish?
3. What task was most enjoyable?

4. Did you have any problems remembering how to execute commands in either of the interfaces?
5. If you have any other comments, please leave them here. Thanks for participating in this experiment.

APPENDIX B Case Study Results

1. Trending Models for Selected Data from Video Coding

1.1 Trending Model for "Sketch" Codes from Participant 1

P-value: 0.017732

Equation: Running Sum of Duration = $42.2097 \cdot \log(\text{Minute}) + -24.1265$

Coefficients

<u>Term</u>	<u>Value</u>	<u>StdErr</u>	<u>t-value</u>	<u>p-value</u>
log(Minute)	42.2097	5.69675	7.40943	0.017732
intercept	-24.1265	15.3189	-1.57495	0.255944

1.2 Trending Model for "Sketch" Codes from Participant 2

P-value: 0.027999

Equation: Running Sum of Duration = $47.4971 \cdot \log(\text{Minute of End}) + -52.4006$

Coefficients

<u>Term</u>	<u>Value</u>	<u>StdErr</u>	<u>t-value</u>	<u>p-value</u>
log(Minute)	47.4971	11.8727	4.00052	0.027999
intercept	-52.4006	26.8224	-1.95361	0.145759

1.3 Trending Model for "Sketch" Codes from Participant 3

P-value: 0.100352

Equation: Running Sum of Duration = $51.1686 \cdot \log(\text{Minute of End}) + 145.173$

Coefficients

<u>Term</u>	<u>Value</u>	<u>StdErr</u>	<u>t-value</u>	<u>p-value</u>
log(Minute)	51.1686	17.5597	2.91398	0.100352
intercept	145.173	22.4104	6.47793	0.0230109

1.4 Trending Model for "Sketch" Codes from Participant 4

P-value: < 0.0001

Equation: Running Sum of Duration = $0.764772 \cdot \text{Second} - 1.76568$

Coefficients

<u>Term</u>	<u>Value</u>	<u>StdErr</u>	<u>t-value</u>	<u>p-value</u>
Second	0.764772	0.0370222	20.6571	< 0.0001
intercept	-1.76568	1.2338	-1.43109	0.180194

2. Semi-Structured Interview Transcripts

In this section, we report the transcription of the semi-structured interviews. R represents the investigator, and P indicates the participants.

2.1 Session 1

R: Could you describe what you would change about the sketch-based interactions to make the experience better?

P: I think it would be easier if you think in a 2D ways, and then be able to make it 3D; otherwise I have to think about how will I make this, rather than.. oh I have a sketch, let me make it 3D.

R: So for SolidSketch tool, do you think it's more 3D-oriented?

P: I think it's more 3D.

R: There are lots of other sketch to 3D programs problem, I would say that our human brain has some 2D elements.

P: Yeah, I think...Let me draw....

R: Ok, so currently SolidSketch assumes those things are holes, but sometimes you don't want them to be holes?

P: Right. Maybe that would be good.

R: Maybe the same interaction? Maybe like you select the shape and scratch that section, like you do on 2D.

P: Yeah, that would thinking in a 2D, because this way I think, I have to make 3D first, then I have to extrude it, then here I have to draw another one too.... You have to kind of pre-plan in 3D, versus just I have a floor plan like this, and I want to add here, instead of maybe building walls, I want to be able to draw this here, and then say, this here is a hole, then I scratch that, then I extrude it.

P: Maybe you are able to do it right now...but I had a hard time with it because it was maybe, I wasn't precise enough or something like that.

P: Maybe I'm preplanning too much in 3D.

R: Well...we are trained as architects, we will always preplan something...but we want to know what the tool helps to offload that process...What were you saying?

P: I was trying to think more about it.....when I train my students to use 3D max, I train them in a way where they have to preplan their actions when they are thinking in 3D, I'm

thinking of sketching ideas and all this, I have a preplan, so maybe I approach with that here, or...I'm just trying to filter out what was my thought process....

P: I think I had in mind sketch up...the way that sketch up works....something like this...but I thought it was more of a 3D sketching, than from there....

P: Now SolidWorks....I'm not a big fan of SolidWorks....It's a great tool, but I don't have much freedom, I feel exhausted after working in SolidWorks both physically and mentally.

P: I think it (SolidSketch) is a great tool...maybe add a couple interactions like you can choose where you can choose 'inner' it would be great. I think that is exactly what I had in mind.

R: What task was the most difficult?

P: I think it was drawing inner shapes before extrusion...

R: What else?

P: I didn't think it was hard to use. I think one that is a little harder, I think I would replace this [corner], with that [arc].

R: Ok, yeah, I was originally planning that for another gesture.

P: Ok, I thought that was a little bit tricky, I had to do that several times. I think others were pretty easily.

P: The concept looked very fluent and very easy, of course, after tutorial, it doesn't follow exactly.

R: What task was the most enjoyable?

P: I think the one where you have uneven shape, then you are able to join those and it gives you....modifier, because sometimes that really requires a lot of tools and a lot of moving of vertices and points to make sure things are straight.

R: For the traditional tool?

P: In different tools, in order to make them straight.

R: Like different curvature?

P: To me, it was mostly straightening these lines....I wasn't stressing about making a straight line, or whether I'm going to be this way or that way, but in sketch you just select these two and you make relatively quickly 'right angle' and 'same sides' (relationship commands).

2.2 Session 2

R: What to change?

P: Change? I think you showed me 2 or 3 different ways to copy or delete, one or the other, and then I was like...oh why is it? Is it copy? That would be kind of confusing...

R: Why?

P: Yeah...because one is lines, and one is actually using two....

R: Which one is better, in your mind?

P: The line is not really intuitive, but with the two fingers... I'm not sure...some of it is trying to get used to it...Like I know the pen and the rotating and the pens, and zooming, all those three, it's probably going to be used a lot during sketching, but then trying to get used to: Is it two hand? Is it pen? Getting my pen out....which you never know until you start using it...what takes me longer is what makes me comfortable...I don't know yet...I think it's what's the purpose of the sketching, because before you gave me this program, I was thinking of this....[pointing to sketching on paper]....Then you gave me the program, then I had to switch to thinking about this....

R: Yeah, because the design is about the form as well as the function....

P: Yeah, I think if I'm more....because you showed me with the walls...and I could see that...I sketched it first then retraced it, that's the nice thing in my head, because in autocad, you can't retrace....because you have the lines and you retrace it with the walls, and we don't really care...

R: Why don't we care?

P: In my head, if I create this line, then I need to use that line for the wall. Like it's an AutoCAD thing, you don't want a double line, like in AutoCAD. I have to be all crash...

R: That is why you didn't realize this tool could do that?

P: Yeah, because even if I do sketch up, that will harm me in the long run, double lines... But in this case, if it's only just a sketch, it doesn't really matter...

R: Here is the proposal: if you think there are two models:

Current program

Sketch on paper

Would that be more helpful? Or would you still want to sketch on paper then sketch in the program?

One combines paper sketching with non-paper, and then there is a switch in the tool. Later you could switch to do tracing and model building, is that helpful?

P: Yeah. If it could do this over there. I could see that works. Yeah. Because my problem sketching by hands is trying to figure out the autographic things, I keep on going back to what actually it would look like if I actually Boolean this with this, but again that is a different type of design. I like to work with basic shapes, and have that basic shape Boolean and that create new shapes. Other people just create a flow, like a Frank

Gehry type of shape, which is also good, because it is going to always intersect with other shapes [ambiguity]. If you don't need to think about that intersection there, then that's good.

R: What is the most difficult task?

P: I mean...if I draw this there, because I was trying to show you, because I am having a problem with the tweak....I start with the square, I have a free form, and then oh I am going to earlier that house, let's see if my house is actually on the....I'm going to put columns like this, so it's actually kind of raise, then I start thinking oh...how am I going to go down? The z plane? I'm not used to the x/y plane.

....That's probably it....

[She is used to sketching in perspective]

...with sketch up....there is an x/y plane....

P: Oh...and then probably the problem... if it has to be closed or not closed, like those lines and those lines, and how, what's the different, I suppose, if it's only lines, or if it's already extruded....I mean that's just I need to learn more because you showed me...Oh ok if you already have it extruded, you can still manipulated, but I think you can manipulate as long as it's a closed like....

Because I think you can add...or...you could change the shapes. You could do that if it's closed.

R: What is the most enjoyable tasks?

P: I think when I did the last one... I started adding stuff, and then I started realizing, I can be more creative, because earlier I just think of steps...like how I am going to make this tool work...then when we just started doing it, I just added stuff.

2.3 Session 3

P: I was thinking...if seeing how XML assigned names to geometry...I wish there was a setting with just the 3 ones, if it would project on the side, that numerical number it gave the geometry.

P: If I draw a wall here and another wall here, then it would project 29 here on the face, and tell you the name the XML file. The object name.

R: The object name?

P: Yeah! I find it interesting.

R: Why?

P: I'm attracted to unfinished building. To a construction building 70 percent done. I think as a purist. It kind of reminds me of how some starts are named.

R: You have an interest in knowing those names?

P: I find it interesting, it's kind of like a shipping container.

R: Ok...like a barcode?

P: Yeah. A barcode, like a systematic, it's like AI, like it's naming it for you.

R: Currently the name for each object looks like this...

P: If there was a name to abbreviate that...

R: Like Wall 1, Wall 2...

P: Yeah...if there was a way to write that...

R: What would you change when you design the sketch interaction?

P: Maybe if there was a way to cycle...if I have a geometry here, here...to cycle to pick the geometry...because once it gets more complicated...I have a plane here, a plane here, so I'm not rotating so much to try to grab it..

R: Oh you mean because currently you have to do rotation, but you want a button to go around?

P: No just a cycle, like here, here, here, a list to selected items which one you want...it would go through all the possible options....sometimes I could see the back of the object without rotating...

P: Kind of like in Catea, you just go to the tree...that type of mentality.

P: What else would I change? I should have been taking notes as I was modeling, as ideas were coming...

R: As you were doing these models what kind of ideas?

P: Learning how organic my line is.... You draw a different line it might look like that...it is kind of an amplification of my line style....

R: For example, you are trying to create a wall; using that prototype wall creation tool, would it retain your style of line?

P: I think I would have to make the wall in the first place... What I like about the program is that it is a very organic feel because everything is based on your handwriting...it respected your unique line styles. I guess... the only person who I think who say no to that is someone who doesn't like their handwriting...your line is going to define how you draw....so you can practice your lines....

R: I remember drawing lines as an architect, trying to draw them beautifully...

P: Like an old professor, that has arthritis....that's his line! It's kind of like a fingerprint...everyone has their line....that's what I find most interesting from the program...that's what keeps you engaged...

R: What part disengaged you?

P: Being unable to select the geometry quickly...having to rotate...sometimes I have a view I like, but it gets the wrong thing...it's....yeah....and....I think...was there any command driven by double tap?

R: I could use those commands...for what?

P: Like, let's say, once you find your geometry, I have that geometry there, you double tap, and it kind of locks in, and it gives emphasis. It isolates.... it will isolate that block. It will gray out the rest, but it will...give modeling on...your detailing this block now....double tap to get out....back into the main model....and now actually, I want to detail this line. So it kind of focuses on the axis and the line, and I guess it's more of a visualization thing....

2.4 Session 4

R: Okay, what would you change for that sketch interaction and maybe touch interactions?

P: Hmm... I'll...

R: If it were you... You know, how would you design it?

P: Ugh... I think if I were doing more fluid sketches I definitely want an opportunity to be able to do something with just a line instead of a closed shape so you can manipulate different things with it. And probably for something that's more free like this I would want to see a loft and sweep function...

P: ...instead of just extrude. Because if you are moving fluidly, I think it would be a little more conducive to that.

R: Like that sweep when you are playing generate some varied shapes.

P: Instead of just one lines, you can do each sides.

R: So that's one problem with the geometry generation libraries I'm using; so what if there's library right now I'm using so that you are able to control the sweep with two lines. You would like to see that happens, right?

P: That would be nice. And maybe some easier control when you draw fabulously versus when you get some jaggedness.

R: Sometimes you want... like on and off, basically. I want to draw just a straight line right now and I don't want jaggy line.

P: It would be just as easy as laying fingers on something (like using combination of touch and stylus)... that would be fine.

P: I think it's pretty easy to change your work plane, but it would be nice to ... sort of quicker function. Okay so you want to work back and forwards between two.

R: Oh!! Like switch between.

P: ...because that's one of the benefits when you sketch. Because you can draw on your plane quickly and move to elevation and go back to plane and maybe some sort of overlay so you don't have to change something to see where it is coming from.

R: Okay, what do you mean by overlay?

P: Oh like using trace paper. So you can actually have...

R: So what do you think the most difficult when you are trying to do it?

R: Let's see.... Probably when use sort of editing the existing geometries when you have to... say when you are closing and there is a section left over and you have to try to delete that.

R: That's the recognition problem actually so that should be solved by a better algorithm.

R: What else... in terms of interaction?

P: Defining a plane is a little bit more difficult. It wasn't accessibly sealed but enough sort of messed with the fluidity of sketching. And ugh... it's pretty much it.

R: What was the most enjoyable and most engaging tasks?

P: I really like its and maybe this would be without as much defined tasks for somebody's who is doing more formal studies I thought it was very nice for sort of quick sketch that is something in 3D and just making a shape for the sake of that. That was really nice. I felt good and good response to it.

R: Did you have any problems remembering the commands? Different commands during interactions?

P: Ugh... I think specifically remembering... maybe not remembering to select but trying to keep track of what was selected and if something I didn't, de-select an object that was hard to remember.

R: So do you think if I improve the way visualizing the selection item, it will improve this issue?

P: Yeah, I think so.

R: Alright, do you have any more comments?

P: Ugh... I think of that was pretty thorough. As far as what I think about the experience.

P: The hand controls as far as moving around it. It was really good.

R: So it was like a part of which is the most enjoyable.

P: Yeah!!

R: Alright!!

References

- Adams, A., Lunt, P., & Cairns, P. (2008). A Qualitative Approach to HCI Research. In P. Cairns & A. L. Cox (Eds.), *Research Methods for Human-Computer Interaction* (pp. 138–157). New York, New York, USA: Cambridge University Press.
- Anderl, R., & Mendgen, R. (1995). Modelling with constraints: Theoretical foundation and application. *Computer-Aided Design*, 28(3), 155–168.
- Andreas, E. (2010). Directive Minds: How Dynamics Shapes Cognition. In *Enaction: Toward a New Paradigm for Cognitive Science*. (pp. 219 – 244)
- Bae, S.-H., Balakrishnan, R., & Singh, K. (2008). ILoveSketch: As-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (pp. 151–160). New York, NY, USA: ACM. doi:10.1145/1449715.1449740
- Bae, S.-H., Balakrishnan, R., & Singh, K. (2009). EverybodyLovesSketch: 3D sketching for a broader audience. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology - UIST '09* (pp. 59 – 68). New York, New York, USA: ACM Press. doi:10.1145/1622176.1622189
- Baran, I., Lehtinen, J., & Popović, J. (2010). Sketching clothoid splines using shortest paths. In *Computer Graphics Forum* (Vol. 29, pp. 655–664).
- Bilda, Z., Gero, J. S., & Purcell, T. (2006). To sketch or not to sketch? That is the question. *Design Studies*, 27(5), 587–613. doi:10.1016/j.destud.2006.02.002
- Bokeloh, M., Wand, M., Seidel, H.-P., & Koltun, V. (2012). An Algebraic Model for Parameterized Shape Editing. *ACM Transactions on Graphics*, 31(4), 78:1–78:10. doi:10.1145/2185520.2185574
- Cantzler, H. (2003). *Improving Architectural 3D Reconstruction by Constrained Modelling*. PhD Dissertation. School of Informatics. University of Edinburgh.
- Card, S. K., Moran, T. P., & Newell, A. (1986). *The Psychology of Human-Computer Interaction*. London: Lawrence Erlbaum Associates.

- Cherry, E., & Latulipe, C. (2014). Quantifying the Creativity Support of Digital Tools through the Creativity Support Index. *ACM Transaction on Computer-Human Interaction*, 21(4), 21:1–21:25. doi:10.1145/2617588
- Csikszentmihalyi, M. (1991). *Flow: The Psychology of Optimal Experience* (Vol. 41). HarperPerennial New York.
- Do, E. Y. L. (2001). VR sketchpad. In *Proceedings of the CAAD Futures Conference* (pp. 161–172).
- Do, E. Y.-L. (1998). *The Right Tool at The Right Time*. PhD Dissertation. College of Architecture. Georgia Institute of Technology.
- Do, E. Y.-L. Gross, M. D. (1997). Inferring Design Intentions from Sketches. In *Proceedings of CAADRIA 97*, (pp. 217-227). Taipei, Taiwan.
- Dourish, P. (2004). *Where the Action Is: The Foundations of Embodied Interaction*. MIT Press.
- Dorsey, J., Xu, S., Smedresman, G., Rushmeier, H., & McMillan, L. (2007). The Mental Canvas: A Tool for Conceptual Architectural Design and Analysis. In *Proceedings of 15th Pacific Conference on Computer Graphics and Applications*, 2007. PG '07. (pp. 201–210). doi:10.1109/PG.2007.64
- Dorta, T. (2007). Implementing and assessing the hybrid ideation space: a cognitive artefact for conceptual design. *International Journal of Design Sciences and Technology*, 14(2), 119–133.
- Eastman, C. M. (1969). Cognitive processes and ill-defined problems: A case study from design. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence* (pp. 669–690). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved from <http://dl.acm.org/citation.cfm?id=1624562.1624622>
- Eastman, C. M., Jeong, Y.-S., Sacks, R., & Kaner, I. (2009). Exchange model and exchange object concepts for implementation of national BIM standards. *Journal of Computing in Civil Engineering*, 24(1), 25–34.
- Eastman, C. M., Teicholz, P., Sacks, R., & Liston, K. (2008). 2.3 Beyond Parametric Shapes. In *BIM Handbook*. New York City, USA: Wiley.

- Eastman, C. M., Teicholz, P., Sacks, R., & Liston, K. (2008). 3.3.6 Implications of IFC Interoperability. In *BIM Handbook*. New York City, USA: Wiley.
- Eggli, L., Hsu, C., Brüderlin, B. D., & Elber, G. (1997). Inferring 3D models from freehand sketches and constraints. *Computer-Aided Design*, 29(2), 101–112. doi:10.1016/S0010-4485(96)00039-5
- Fish, N., Averkiou, M., van Kaick, O., Sorkine-Hornung, O., Cohen-Or, D., & Mitra, N. J. (2014). Meta-representation of Shape Families. *ACM Transactions on Graphics*, 33(4), 34:1–34:11. doi:10.1145/2601097.2601185
- Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. New Jersey, USA: Lawrence Erlbaum Associates.
- Gingold, Y., Igarashi, T., & Zorin, D. (2009). Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics*, 28(5), 148:1–148:9. doi:10.1145/1618452.1618494
- Goel, V. (1995). *Sketches of Thought*. MIT Press.
- Gross, M. D. (1990). Relational Modeling: A Basis for Computer-Assisted Design. In W. Mitchell J. (Ed.), *The Electronic Design Studio* (pp. 123–130). Cambridge, Massachusetts, USA: MIT Press.
- Gross, M. D. (1996). The electronic cocktail napkin—a computational environment for working with design diagrams. *Design Studies*, 17(1), 53–69.
- Gross, M. D., & Do, E. Y.-L. (1996). Ambiguous Intentions: A paper-like Interface for Creative Design. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*. Seattle, Washington, United States: ACM. doi:10.1145/237091.237119
- Gross, M. D. (1986). *Design as Exploring Constraints*. PhD Dissertation. Massachusetts Institute of Technology.
- Gül, L. F. (2009). Studying the Impact of Immersion on Design Cognition. In *Proceedings of eCAADe*, (pp. 615–621). Istanbul. CuminCAD.

- Hart, S. G., & Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Advances in Psychology*, 52, 139–183.
- Hanne De Jaegher. (2010). *Enaction: Toward a New Paradigm for Cognitive Science*. MIT Press.
- Hinckley, K., Yatani, K., Pahud, M., Coddington, N., Rodenhouse, J., Wilson, A., Buxton, B. (2010). Pen + touch = new tools. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology* (pp. 27–36). New York, NY, USA: ACM. doi:10.1145/1866029.1866036
- Hollan, J., Hutchins, E., & Kirsh, D. (2000). Distributed cognition: Toward a new foundation for human-computer interaction research. *ACM Transactions on Computer-Human Interaction*, 7(2), 174–196. doi:10.1145/353485.353487
- Hornecker, E. (2005). A design theme for tangible interaction: Embodied facilitation. In *Proceedings of ECSCW 2005* (pp. 23–43).
- Hsiao, C.-P., Davis, N., & Do, E. Y.-L. (2012). Dancing on the Desktop - Gesture Modeling System to Augment Design Cognition. In *Proceedings of ACADIA 12* (pp. 419–428). San Francisco, CA, USA.
- Hutchins, E. (1995). *Cognition in the Wild*. The MIT Press.
- Igarashi, T., & Hughes, J. F. (2001). A Suggestive Interface for 3D Drawing. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology* (pp. 173–181). New York, NY, USA: ACM. doi:10.1145/502348.502379
- Igarashi, T., Matsuoka, S., & Tanaka, H. (1999). Teddy: A sketching interface for 3D freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 409–416). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. doi:10.1145/311535.311602
- Ihde, D. (1979). Heidegger's philosophy of technology. In *Technics and Praxis* (pp. 103–129). Springer.
- Jonson, B. (2005). Design ideation: The conceptual sketch in the digital age. *Design Studies*, 26(6), 613–624. doi:http://dx.doi.org/10.1016/j.destud.2005.03.001

- Johnson, G., Gross, M., Do, E. Y.-L., & Hong, J. (2012). Sketch it, make it: Sketching precise drawings for laser cutting. In *Proceedings of the 2012 ACM Annual Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 1079–1082). New York, NY, USA: ACM. doi:10.1145/2212360.2212390
- Kang, Y., Kim, H., Suzuki, H., & Han, S. (2015). Editing 3D models on smart devices. *Computer-Aided Design*, 59(0), 229–238.
doi:http://dx.doi.org/10.1016/j.cad.2013.08.001
- Kazi, R. H., Chevalier, F., Grossman, T., & Fitzmaurice, G. (2014). Kitty: Sketching Dynamic and Interactive Illustrations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (pp. 395–405). New York, NY, USA: ACM. doi:10.1145/2642918.2647375
- Kazi, R. H., Igarashi, T., Zhao, S., & Davis, R. (2012). Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proceedings of the 2012 ACM Annual Conference on Human Factors in Computing Systems* (pp. 1727–1736). New York, NY, USA: ACM. doi:10.1145/2208276.2208302
- Kin, K., Miller, T., Bollensdorff, B., DeRose, T., Hartmann, B., & Agrawala, M. (2011). Eden: A professional multitouch tool for constructing virtual organic environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1343–1352). New York, NY, USA: ACM. doi:10.1145/1978942.1979141
- Lakoff, G., & Johnson, M. (2003). *Metaphors We Live By*. University Of Chicago Press.
- Landay, J. A., & Myers, B. A. (1995). Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Denver, Colorado, United States: ACM Press/Addison-Wesley Publishing Co. doi:10.1145/223904.223910
- Lawson, B. (1999). “Fake” and “Real” Creativity Using Computer Aided Design: Some lessons from Herman Hertzberger. In *Proceedings of the 3rd Conference on Creativity & Cognition* (pp. 174–179). New York, NY, USA: ACM. doi:10.1145/317561.317591
- Le Dantec, C. A. (2009). Situated design: Toward an understanding of design through social creation and cultural cognition. In *Proceedings of the Seventh ACM Conference on Creativity and Cognition* (pp. 69–78). New York, NY, USA: ACM. doi:10.1145/1640233.1640247

- Li, W., Agrawala, M., Curless, B., & Salesin, D. (2008). Automated Generation of Interactive 3D Exploded View Diagrams. In *ACM SIGGRAPH 2008 Papers* (pp. 101:1–101:7). New York, NY, USA: ACM. doi:10.1145/1399504.1360700
- Lin, J., Cohen-Or, D., Zhang, H., Liang, C., Sharf, A., Deussen, O., & Chen, B. (2011). Structure-preserving Retargeting of Irregular 3D Architecture. *ACM Transactions on Graphics*, 30(6), 183:1–183:10. doi:10.1145/2070781.2024217
- Liu, Z., Nersessian, N., & Stasko, J. (2008). Distributed Cognition as a Theoretical Framework for Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6), 1173–1180. doi:10.1109/TVCG.2008.121
- Longay, S., Runions, A., Boudon, F., & Prusinkiewicz, P. (2012). TreeSketch: Interactive procedural modeling of trees on a tablet. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling* (pp. 107–120). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.
- Masry, M., & Lipson, H. (2007). A sketch-based interface for iterative design and analysis of 3D objects. In *ACM SIGGRAPH 2007 courses*. New York, NY, USA: ACM. doi:10.1145/1281500.1281542
- McGrenere, J., & Ho, W. (2000). Affordances: Clarifying and Evolving a Concept. In *Graphics Interface 2000* (pp. 179–186). Montreal, Canada.
- Medyckyj-Scott, D., & Hearnshaw, H. M. (1993). *Human Factors in Geographical Information Systems*. Halsted Press.
- Miller, N. (2009). Parametric Strategies in Civic Architecture Design. In *ACADIA 09: reForm() - Building a Better Tomorrow* (pp. 144–152).
- Mitra, N. J., Wand, M., Zhang, H., Cohen-Or, D., Kim, V., & Huang, Q.-X. (2014). Structure-aware Shape Processing. In *ACM SIGGRAPH 2014 Courses* (pp. 13:1–13:21). New York, NY, USA: ACM. doi:10.1145/2614028.2615401
- Nersessian, N. J. (2008). *Creating Scientific Concepts*. Cambridge, MA: The MIT Press.
- Norman, D. (1990). *The Design of Everyday Things*. Cambridge, MA: The MIT Press.

- Norman, D. A. (2008). The Way I See It: Signifiers, Not Affordances. *Interactions*, 15(6), 18–19. doi:10.1145/1409040.1409044
- Piccolotto, M. A. (1998). *Sketchpad+ Architectural Modeling through Perspective Sketching on a Pen-based Display*. Master Thesis. Cornell University.
- Pu, J., & Gur, D. (2009). Automated freehand sketch segmentation using radial basis functions. *Computer-Aided Design*, 41(12), 857–864. doi:10.1016/j.cad.2009.05.005
- Rafiei, D., & Mendelzon, A. (1997). Similarity-based queries for time series data. *SIGMOD Rec.*, 26(2), 13–25. doi:10.1145/253262.253264
- Rittel, H. W. J., & Webber, M. M. (1973). Dilemmas in a General Theory of Planning. *Policy Sciences*, 4, 155–169.
- Sawyer, R. K. (2000). Improvisational Cultures: Collaborative Emergence and Creativity in Improvisation. *Mind, Culture, and Activity*, 7(3), 180–185. doi:10.1207/S15327884MCA0703_05
- Sawyer, R. K. (2011). *Explaining creativity: The Science of Human Innovation*. Oxford University Press.
- Schön, D. A. (1992). Designing as reflective conversation with the materials of a design situation. *Knowledge-Based Systems*, 5(1), 3–14.
- Schmidt, R., Khan, A., Singh, K., & Kurtenbach, G. (2009). Analytic drawing of 3D scaffolds. In *ACM SIGGRAPH Asia 2009* (pp. 149:1–149:10). New York, NY, USA: ACM. doi:10.1145/1661412.1618495
- Schweikardt, E., & Gross, M. D. (2000). Digital clay: Deriving digital models from freehand sketches. *Automation in Construction*, 9(1), 107–115. doi:10.1016/S0926-5805(99)00052-7
- See, R. (2009). Concept Design BIM 2010. Retrieved from <http://www.blis-project.org/IAI-MVD/MVDs/GSA-005/Overview.pdf>
- Shneiderman, B. (2007). Creativity Support Tools: Accelerating Discovery and Innovation. *Communications of the ACM*, 50(12), 20–32. doi:10.1145/1323688.1323689

- Suchman, L. (1986). *Plans and Situated Actions*. New York, Cambridge University.
- Sutherland, I. (1963). SketchPad: A Man-Machine Graphical Communication System. *Spring Joint Computer Conference*, 329–345.
- Suwa, M., & Tversky, B. (1997). What do architects and students perceive in their design sketches? A protocol analysis. *Design Studies*, 18(4), 385–403.
doi:http://dx.doi.org/10.1016/S0142-694X(97)00008-2
- Tolba, O., Dorsey, J., & McMillan, L. (1999). Sketching with projective 2D strokes. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology* (pp. 149–157). New York, NY, USA: ACM. doi:10.1145/320719.322596
- Tversky, B. (2002). What do Sketches say about Thinking? *Proceedings of AAAI Spring Symposium on Sketch Understanding* (pp. 148-151) Stanford University, AAAI Technical Report SS-02-08. 2002.
- Ullmer, B., & Ishii, H. (2000). Emerging frameworks for tangible user interfaces. *IBM Systems Journal*, 39(3.4), 915–931. doi:10.1147/sj.393.0915
- Varela, F. J., Rosch, E., & Thompson, E. (1992). *The Embodied Mind: Cognitive Science and Human Experience*. MIT press.
- William J, C. (n.d.). Situated action: A neuropsychological interpretation response to Vera and Simon. *Cognitive Science*, 17(1), 87–116.
- Wobbrock, J. O., Wilson, A. D., & Li, Y. (2007). Gestures without libraries, toolkits or training: A \$1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology* (pp. 159–168). New York, NY, USA: ACM. doi:10.1145/1294211.1294238
- Xu, B., Chang, W., Sheffer, A., Bousseau, A., McCrae, J., & Singh, K. (2014). True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization. *ACM Transactions on Graphics*, 33(4), 131:1–131:13. doi:10.1145/2601097.2601128
- Yang, C., Sharon, D., & van de Panne, M. (2005). Sketch-based Modeling of Parameterized Objects. In *ACM SIGGRAPH 2005 Sketches*. New York, NY, USA: ACM. doi:10.1145/1187112.1187219

- Zelevnik, P. C. (1996). SKETCH: An Interface for Sketching 3D Scenes. In *Proceedings of SIGGRAPH 96, Computer Graphics* (pp. 163-170). New York, NY, USA: ACM.
- Zheng, Y., Fu, H., Cohen-Or, D., Au, O. K.-C., & Tai, C.-L. (2011). Component-wise Controllers for Structure-Preserving Shape Manipulation. In *Computer Graphics Forum* (Vol. 30, pp. 563–572).