Novel interfaces for modeling and painting

Sun, Qian

2014

Sun, Q. (2014). Novel interfaces for modeling and painting. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/59547>

<https://doi.org/10.32657/10356/59547>

Novel Interfaces for Modeling and Painting

Sun Qian

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2014

Abstract

The last decade witnessed the rapid development of interaction techniques. There has been a growing number of interactive devices, such as multi-touch displays, motion sensing input devices and voice input devices. Among them, touch based hardware are becoming more and more powerful in terms of screen size, resolution, accuracy, as well as the number of touch points, while the prices are dropping quickly. More and more electronic devices have emerged with touch screens, including smart phones, tablet PCs, GPS devices, ATM machines, various information and ticketing systems, *etc.*

Compared to traditional WIMP interfaces that require the user to input information indirectly with IO devices, such as mice, touch-based interfaces, including multi-touch and sketch-based interfaces, allow the user to interact with the content directly using fingers and pens; a technique which has proved to be extremely intuitive and easy to learn. Additionally, multi-touch based interfaces can be more efficient than mouse-based interfaces and have great potential to facilitate collaboration among multiple people. Also, sketch-based interfaces can be very intuitive due to the simple sketch-based input and physical direct manipulation with the input screen.

Because of the application-driven nature of computer graphics and the difficulty of visualizing and manipulating various aspects of computer graphics, including geometry creation, editing and so on, the interface and interaction method of applications play an important role in computer graphics applications. However, there are relatively few successful applications have been presented which have exploited novel interfaces in computer graphics applications due to the complexity of such applications. In this thesis, we demonstrate three graphics applications that use novel interfaces to improve the user interaction in terms of efficiency and intuitiveness. By conducting user studies for the specified applications and tasks, we also explore some touch interaction principles that

might be followed up by later work. Our applications cover a wide range, including touch-based 2D drawing, surfaces texturing and 3D modeling of tall buildings.

Painting with 2D Diffusion Curves. Diffusion curves are effective 2D vector-graphics primitives, for creating smoothly-shaded drawings with rich colors and unique styles. Conventional drawing systems for diffusion curves often require users to successively layout curve geometry and then specify colors, which is rather tedious for complex drawings. We provide a novel multi-touch sketching interface for efficient design of 2D vector graphics with diffusion curves. By developing a family of multi-touch gestures, our interface allows users to simultaneously sketch multiple diffusion curves and also to interactively edit and tune curve geometry and colors at the spot.

Surface painting with textures. Texturing is an important step in the graphics pipeline that adds various types of details to an existing 3D model. We inform Texture Brush, an interactive interface for 3D surfaces texturing. With Texture Brush, the user can easily specify a local parameterization with a single free-form stroke on the surface, and paste an image along the stroke. We also propose a set of powerful and intuitive operations including texture painting, texture cloning, texture animation, and texture editing. Compared to the existing surface texturing techniques, our method enables a smoother and more natural work flow so that the user can focus on the design task itself without switching back and forth among different tools or stages. The encouraging experimental results and positive evaluation by artists demonstrate the efficacy of our Texture Brush for interactive texture mapping.

Multi-touch based building modelling. Sketching and massing are two primary design techniques widely used in the early stages of building design. We develop a novel, multi-touch interface for supporting the early stages of building design, aiming to combine aspects of sketching and massing to aid the design process. We propose a family of multi-touch gestures, where one can start by sketching and editing the 2D contour of the base floor, and then extend and edit it to create a 3D building model with fairly complex geometric expression in a matter of minutes. The simple drag-and-drop texturing can further enrich the model’s appearance. In addition to 3D modeling, our interface computes and provides assorted architectural parameters for the building design.

Acknowledgments

I am grateful to all those who have helped and supported me throughout my Ph.D. life.

First and foremost I would like to thank my supervisor Dr. Ying He for continuous support and kindly concern of my Ph.D. study and research. This study would not have been possible without the guidance, patience and support of him. Besides the valuable research skills that I learnt from him, his personal fascination also affects me to be a kind, happy and self-motivated person.

I would grateful acknowledge Dr. Chi-Wing Fu, an expert in computer graphics and human computer interaction, for his precious advice and unlimited encouragement on various projects.

I also wish to thank all my friends at GameLab Dr. Jiazhi Xia, Dr. Xiang Ying, Dr. Dayong Wang, Ms. Xiaoqi Yan, Dr. Min Meng, Mr. William Lai, Ms. Xiang Xu, Mr. Minqi Zhang, Mr. Xiaoning Wang, Dr. Peng Song, Dr. Dao Thi Phuong Quynh, Dr. Juncong Lin, Dr. Long Zhang, Dr. Shi-Qing Xin, Dr. Xiaoming Chen, Mr. Shuchu Han, Mr. Fuwen Tan for their assistance and valuable advices on many research projects and all the precious memories we created together.

Last but not least, I would like to thank my parents for their constant supporting and encouragement. Although I was away from home for this study, I always feel their care and love.

Special thanks to my boyfriend, Mr. Wang Yao, for his trust, his encouragement and the happy memories we've shared.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vi
List of Tables	xi
1 Introduction	1
1.1 Background	1
1.1.1 Touch Systems	2
1.2 Objectives and Contributions	6
1.3 Thesis Organization	10
2 Related Work	11
2.1 Indirect and Direct-Touch Interaction Techniques.	11
2.2 Unimanual and Bimanual Interaction Techniques.	12
2.3 Multi-touch Interaction Techniques.	13
2.4 Touch-based Applications.	14
2.5 Painting and texturing.	18
3 Painting with 2D Diffusion Curves	22
3.1 Motivation	22
3.2 Multi-touch Sketching Interface	23
3.3 Implementation and Results	26
3.4 User study	33
3.5 Summary	37

4	Surface Painting with Texture Brush	39
4.1	Motivation	39
4.2	Extended Exponential Map & Local Parameterization	43
4.3	Texture Brush Interface	47
4.3.1	Texture Painting	48
4.3.2	Texture Cloning	49
4.3.3	Texture Animation	51
4.3.4	Texture Map Editing	52
4.4	Experimental Results	53
4.5	Expert Evaluation	55
4.6	Summary	56
5	Multitouch based Building Modelling	60
5.1	Motivation	60
5.2	Overview	63
5.3	Design Considerations	66
5.4	Multi-touch Gestures	68
5.4.1	Initial Building	68
5.4.2	Structure Editing	71
5.4.3	Appearance Enhancement	74
5.4.4	Camera View	75
5.4.5	Design Coherency	76
5.5	Implementation and Results	76
5.6	Expert review	77
5.7	User study	80
5.8	Summary	83
6	Conclusion and Future Work	85
6.1	Conclusion	85
6.2	Future Work	88
	References	89
	Publications	103

List of Figures

1.1	The development of multi-touch technology.	2
1.2	Buildings with progressively-varying and twisted structures: <i>Burj Al Arab</i> in Dubai, <i>Taipei 101</i> in Taipei, <i>Turning Torso</i> in Sweden, and <i>The Shard</i> in London (left to right).	9
3.1	Diffusion curves (see (a)) enable effective generation of smoothly-shaded 2D vector-graphics (see (b)). After sketching a spline (see (c)) and specifying colors on it (see (d)), we can apply a Poisson equation solver to generate the drawing on the canvas (see (e)).	24
3.2	Our multi-touch sketching interface for diffusion curves	24
3.3	Multi-touch to select colors (left) and adjust colors (right).	26
3.4	Different levels of the same image.	29
3.5	Different levels of the same image.	33
3.6	Paintings created by an artist with our interface: Disney cartoon, silk painting, and photo-realistic effect can be readily produced (from left to right). The artist used this system for the first time and was given 5 minutes in learning. The painting time varies from 15 to 40 minutes. . .	34
3.7	Tasks in Experiment 1 (left) and Experiment 2 (right).	36
3.8	User study results. C: conventional; M: multi-touch.	37
4.1	Texture brush painting on models. Left: the cup is textured with 15 painting operations using 8 texture brushes. Right: the fertility model is textured with 8 painting operations using 3 texture brushes. The colored lines indicate the painting strokes.	39

4.2	Extended exponential map. (a) The user sketches a curve on the surface. (b) Geodesic offsets of the curve. (c) The integral curves are the geodesics; large distortions can be viewed in the close-up view. (d) Local parameterization along the curve. (e) The fixed tracing lines by harmonic maps. (f) The fixed local parameterization.	44
4.3	(a) Measuring how the curved region bends. (b) Parameterization for highly curved strip region.	46
4.4	Our extended exponential map can achieve high quality parameterization with low distortion.	46
4.5	The user interface. (a) A snapshot of our interface; (b) We implement our texture brush on a multi-touch display so that all the operations can be done by using multi-touch gestures.	47
4.6	Texture painting. (a) The user sketches on the mesh surface; The red line illustrates the painting stroke; (b) Extended exponential map corresponding to the input curve; (c) The texture mapping result.	48
4.7	Texture cloning. (a) The user paints on the texture object; (b) The extracted texture in image space; (c) The matted texture object with alpha mask; (d) The user paints on the target surface with the texture brush; the texture is composited with the background texture.	49
4.8	Texture animation. (a) The yellow curve shows the route of animation, and the red curves are animated control sketches at different time stamps; (b) snapshots of the animation at different times.	49
4.9	Texture cloning from the source model (a) to the target model (d). . . .	51
4.10	Texture brush curve editing. The user drags the control points on the base curve to modify the shape of the texture mapping.	52
4.11	Texture deformation. (a) After texture painting on the surface, the user moves points on the texture object to deform the texture object; (b) The control points and the moving vector are mapped to image space; (c) Deformation is applied in image space; (d) Deformed texture object is mapped back to surface space.	53

4.12	The curve editing algorithm. (a) The user drags a point in the texture brush curve which is presented by the blue strip; the effective weights of the points are distributed as a gaussian distribution, where the red color refers to 1, the blue color refers to 0, and the gray color refers to inactive; (b) The editing result; the position of each point is moved by the dragging vector multiplied by the effective weight.	54
4.13	Comparison of parameterization in highly curved region. (a) Extended exponential map. (b) Exponential map.	54
4.14	Texture animation: the 2D flower textures are moving along the user-specified paths.	57
4.15	Bunny textured with I3D'13 logo.	58
4.16	More texturing results.	59
5.1	Buildings with progressively-varying and twisted structures: Burj Al Arab in Dubai, Taipei 101 in Taipei, Turning Torso in Sweden, and The Shard in London (left to right). Images are from www.flickr.com	61
5.2	An architect constructed prototypic models of famous buildings by using our multi-touch interface. The time taken to create these 3D building prototypes is shown below each model.	63
5.3	(a) Our multi-touch display system; (b) Our GUI interface consists of a main working window (left) that presents the building design in 3D and a floating tabbed panel (right) that offers a 2D floor plan editor, a building profile editor, and various resource libraries.	64

5.4	System overview. Here we show our design pipeline with the <i>Turning Torso</i> building. The user starts by creating a 2D floor plan, which can either be selected from the tabbed panel (see (a)) or sketched by the user. After that, the user can freely edit the 2D floor contour using the 2D floor plan editor by manipulating the control points on the 2D contour. Our system provides multi-touch gestures and a novel grid&symmetry snapping mechanism for users to edit the 2D floor contour (see (b)). The inset for the two-finger anchor gesture specifies the symmetry axis. Later, the user employs one finger to select the floor plan and create the first ground-level floor (see (c)). After that, the user can create multiple floors by duplicating the ground floor (see (d)). Then, the user can apply an image texture to the selected floors by a simple drag-and-drop (see (e)). Our interface also allows the user to easily copy and paste selected floors to extend the building levels (see (f)), and zoom out using a two-finger pinch (see (g)). Finally, the user can select the whole building and use a one-finger horizontal drag to twist the building structure (see (h)) to produce the final result (see (i)).	65
5.5	Multi-touch gestures for editing 2D floor plan contour. (a) Two fingers to specify the axis for symmetry snapping; while we hold the fingers on screen, we can still move them to adjust the symmetry axis (see the inset); (b) We can freely move a control point with a finger and it could snap to a mirror location of a point on the other side of the symmetry axis; see the inset for making a symmetric shape; (c) To add a control point, we can double-tap on a line/curve segment. The inset shows the result; (d) To remove a control point, we can drag it off the drawing area.	69
5.6	Multi-touch gestures for structure editing: (a) copying floors; (b) removing floors; (c) scaling floors (heights or contours); (d) editing the building profile for the entire building; (e) editing the building profile locally on selected floors; and (f) twisting.	71
5.7	Cropping the facade.	74
5.8	Gestures for camera view controls.	75

5.9	Side-by-side comparison of designs is desirable for users to clarify their design ideas and to arrive at a coherent building solution. To delete a design, one can simply drag-and-drop it (a); the view can be updated (b).	77
5.10	Some results from the architects who participated in the expert review. All models were made in a few minutes.	78
5.11	Questionnaire in the expert review.	79
5.12	User study. We designed two experiments for buildings with non-trivial geometry to compare the efficiency of our multi-touch interface against Multi-touch Google Sketchup in terms of fast building prototyping. . . .	81
5.13	User study results. The vertical axis shows the average time taken for the participants to complete each experiment. The black vertical line on each bar is the standard deviation. Time was measured in seconds.	82
5.14	Our interface can be used with other types of 3D design, where the 3D models have repeated geometric features, planar/rotational symmetry, or curved profiling.	84

List of Tables

3.1	Resultant t-value.	36
4.1	Comparison of extended exponential map and exponential map.	55

Chapter 1

Introduction

1.1 Background

We have seen in movies such as *Minority Report* and *Avatar* how we may one day use novel interaction methods such as using our fingers as input to manage digital information. In recent years, this idea has been emerged greatly to modern electronic products, from smart phones, tablet PCs, GPS devices, e-book readers, ATM machines, various information and ticketing systems to just about anything. Touch technology may be considered to have a key role in the success of modern electronic products [37].

Surprisingly, touch technology has actually existed for decades. The early development of touch devices can be traced back to the late 1960's [37] before personal computers had been invented. In 1972, IBM released the PLATO IV computer (Figure 1.1(a)), which enabled the user to touch on the screen with a single finger [37]. The resolution of the touching array was only 16×16 .

In 1982, Nimish Mehta et al [37]. from the Input Research Group of the University of Toronto, designed the the first multitouch system, which consisted of a frosted-glass panel, with a camera placed behind the glass. Touch points were detected by whether black dots on the glass (caused by the fingers) were caught by the camera. Interestingly, this system was pressure-sensitive since it could recognize how hard the user was pressing on the glass by measuring the size of the dot.

(a) IBM's Plato IV 1972 [37]

(b) Touch Screen.

Image is from www.flickr.com.

(c) Multi Touch.

Image is from www.apple.com.

Figure 1.1: The development of multi-touch technology.

The first multi-touch screen using capacitance was built in 1985 by the University of Toronto group. Capacitance-based touch screens are much more portable than traditionally camera-based optical sensing systems.

The year 2007 [37] was doubtlessly the biggest milestone in the development of touch technology, when Apple released the iPhone((Figure 1.1(c))) and Microsoft released its Surface Computer. Soon touch devices become commonplace in both personal and commercial usage, with a wide range of screen sizes and touch capabilities.

1.1.1 Touch Systems

According to the implementation mechanism, existing touch systems can be classified into two categories: sensor-based systems and computer-vision-based systems.

Sensor-based systems. Lee et al. presented the Fast Multiple-Touch Sensitive Input Device (FMTSID) [66] in 1985. By measuring the changes of capacitance, their system could detect the positions and the pressure of touch points accurately. In 2001, Dietz et al. presented Diamond Touch [75], which could not only detect multiple "touch" movements of a single user, but also enabled multiple users to interact with the desktop. This system could also indicate the pressure at each touch point. However, since the detection of Diamond Touch was based on light and a camera, it would fail when objects were put on the touch screen [26]. Rekimoto et al. proposed Smart Skin [92] in 2002. Smart Skin

could identify the hand touch position, and the shape and distance between the hand and the contact surface. In 2007, Apple release the iPhone, the first mobile device that uses a multi-touch screen as its main interaction media. Based on the capacitance, the iPhone supports multiple touch points at the same time.

Computer-vision-based systems. Computer-vision-based systems employ computer vision techniques to identify human touch interactions. Instead of using sensors to detect the touch points, vision-based systems set up a camera system around the screen which captures the images when the user interacts with the screen. It then identifies the touch interaction by analyzing the images. Compared to sensor-based systems, vision-based systems can support quite large touch screen sizes. Vision-based systems can be further divided into purely vision-based systems and hybrid vision and optical-based systems. Purely vision-based multi-touch systems identify touch and it's positions solely by means of image processing techniques. Representative systems in this category include the Everywhere Display system presented by Pinhanez et al. [88], and Microsoft's PlayAnywhere [112]. Theoretically, such systems can use any flat surface as the input media, thus they have advantages in both low cost and high portability. However, a known issue of such systems is their low precision. Hybrid vision and optical-based systems take advantage of special optical phenomena to detect the touch points. Jefferson [41] presented a low-cost multi-touch system based on Frustrated Total Internal Reflection (FTIR). Microsoft Surface [103] is based on the diffuse illumination technology. Generally speaking, hybrid vision and optical based multi-touch systems are cheaper than sensor-based systems thus can support very large touch screen size. However, they need more space to place the special optical devices.

Advantages of Touch Interfaces:

Intuitive Compared to traditional WIMP(windows, icons, menus, pointer) interfaces in which the user interacts with the system by using indirect input devices such as a

mouse, a touch-based interface allows a user to directly interact with a system by touching the device screen. This interaction method is extremely intuitive, which makes touch devices suitable for any kind of user, even kids. In addition, touch-based devices are now widely used in various information and ticketing systems for the public, who can operate them without any special training.

Effective Though there are many advantages of mouse-based interfaces, such as being easy to learn, and precise, there are only two degree-of-freedom with mice, which leads to very limited interaction capabilities. In contrast, our fingers have a much higher degree of freedom [4]. Thus, tasks that are completed with a sequence of interaction steps using mouse-based input devices could be done in a single multi-touch based interaction by using touch devices. Furthermore, with the development of touch-based hardware, it is easy to implement various fancy gestures on touch devices, which greatly enhances the flexibility and efficiency of touch-based interfaces. As a result, touch-based interactions are more efficient than traditional mouse-based interactions in many situations.

Illustrative Touch interfaces are more effective than traditional mouse-based interfaces for the purpose of illustration. Mouse-based interfaces sometimes distract users from the content of an illustration due to the small size of the cursor and the low color contrast between the cursor and the background. Thus, the user may lose track of the mouse cursor, and it may take a while to find it again. In contrast, a touch interface is able to keep the user's focus on the illustration. When showcasing a live demo, the audience can focus on the presenter and the illustration, according to the action of the presenter. When the presenter is stationary and talking, the audience can focus on the illustration. Also the audience can focus on the presenter if he/she changed his/her postures and gestures. Both postures and gestures are

designed to be as easy to understand as possible. Hence, when the presenter uses a touch operation to trigger an event, such as selection and flip a page, the process is seamless because the touch interactions are meaningful and easy to trace.

Collaborative Multi-touch interfaces are capable of handling multiple users' input for the same touch device. In other words, different users can interact with the same content concurrently. This capability has much potential for collaborative work among multiple users.

Disadvantages of Touch Interfaces:

Touch-based interfaces have their own drawbacks compared with traditional mouse-based interfaces. First of all, due to the size of human fingers, touch-based interfaces generally have lower precision than mouse-based interfaces. To address this problem, commercial products such as the iPhone provide large icons and buttons that reduce the requirement for input precision. In the research community, various methods have been proposed to increase the precision of touch-based interactions, including slowing down the movement of the cursor around the initial touch point [98], locally zooming in/out the display [80, 10, 15], and providing certain screen widgets [3, 12] to assist in selecting small targets, etc.

Another problem of touch-based interfaces is the visual occlusion caused by the fingers and hands. Unlike mouse-based interactions, the fingers and the hands may occlude a large area of the screen, making some operations, such as selection, difficult to perform. Potter et al. [21] addressed this problem by proposing a so called Take-Off technique which offsets the cursor by a fixed number of pixels when the user is performing touch-based interactions. Wu and Balakrishnan [115] also proposed a smart solution to this problem.

In addition, as pointed out by Buxton [18], most touch systems can not distinguish the hover (tracking) and dragging interactions.

As a result, though a number of brilliant research works have been conducted on touch techniques, there are still many problems related to this area which need to be explored and solved.

1.2 Objectives and Contributions

Although the research work focused on how to build a touch system, there has also been a lot of work investigating how to take advantage of existing touch systems to develop new applications, or to improve the user interaction for traditional applications. The research work described in this thesis falls into this category.

We aim to improve users' interaction experience with graphics applications by using touch interaction techniques. Furthermore, we aim to explore touch interaction principles by conducting user studies for specified applications and tasks. Accordingly, we provide three touch-based interfaces for 2D drawing, 3D texturing and 3D modeling .

In 2D drawing project, we mainly consider the effective utility of touch interfaces for the design of 2D vector graphics with diffusion curves. With our interface, users can simultaneously sketch multiple diffusion curves and also interactively edit and tune curve geometry and colors at the spot.

In 3D texturing project, we mainly take the advantage of intuitiveness of sketch based interfaces for sketching on 3D surfaces. With Texture Brush, the user can easily specify such local parameterization with a single free-form stroke on the surface. In 3D building modeling project, we combine most advantages of multi-touch interfaces, such as intuitive, effective, and Illustrative. Using our interface allows one to quickly define and edit the geometry of a building model. Such rapid 3D building prototyping is not easily achievable with conventional multi-touch 3D modeling interfaces, existing 3D sketching tools, and general 3D modeling software.

Painting with 2D Diffusion Curves. Diffusion curves are effective 2D vector-graphics primitives, for creating smoothly-shaded drawings with rich colors and unique styles. Conventional drawing systems for diffusion curves often require users to successively layout curve geometry and then specify colors, which is rather tedious for complex drawings. We provide a novel multi-touch sketching interface for efficient design of 2D vector graphics with diffusion curves. In sharp contrast to previous interfaces, we develop a family of multi-touch gestures, allowing users to simultaneously sketch multiple diffusion curves and also to interactively edit and tune curve geometry and colors at the spot. Our experiments show that this not only brings novel painting experience to users but also provides a practical and effective tool for vector graphics design, useful for styles like silk painting, Disney cartoon, art poster, and photo-realistic effects. Using a working system we developed, we conducted a user study to demonstrate the practical value of this interface design with both professional artists and novice users. Results show that this novel design not only brings new painting experience, but also provides a practical and effective tool for vector graphics design, useful for styles like silk painting, Disney cartoon, art poster, and photo-realistic effects. Lastly, we conduct a user study to explore the interface’s intuitive and efficient drawing capability with both professional 2D artists and novice users.

Surface painting with textures. We inform Texture Brush, an interactive interface for texturing 3D surfaces. We extend the conventional exponential map to a more general setting, in which the generator can be an arbitrary curve. Based on our extended exponential map, we develop a local parameterization method which naturally supports anisotropic texture mapping. With Texture Brush, the user can easily specify such local parameterization with a single free-form stroke on the surface. Because the interaction metaphor is the same as conventional 3D painting, artists’ experience of 3D painting can be transferred to our texture mapping interface naturally. Artists can focus on the

high level semantics of texture mapping, such as the general shape, without paying attention to the parameters of the underlying parameterization. We also propose a set of powerful and intuitive operations which are mainly based on 3D painting metaphor, including texture painting, texture cloning, texture animation design, and texture editing. Compared to the existing surface texturing techniques, our method enables a smoother and more natural work flow so that the user can focus on the design task itself without switching back and forth among different tools or stages. The encouraging experimental results and positive evaluation by artists demonstrate the efficacy of our Texture Brush for interactive texture mapping.

Multitouch based building modelling. Building design is an event per excellence, involving complex considerations and requiring designers to explore a vast problem space. Sketching and massing are two primary design techniques widely used in the early stages of building design [100]: *sketching* is a common design method used for generating ideas and exploring options with pen and paper, while *massing* is a primary design process normally used for understanding the overall shape and size of a building in relation to its surrounding environment with approximate volumetric information. We develop a novel, multi-touch interface design for supporting the early stages of building design, aiming to combine aspects of sketching and massing to aid the design process. In particular, we focus on the design of tall buildings due to the scale and complexity involved in such design. Moreover, we can observe the trend in tall buildings to be more expressive in its silhouette due to the level of maturity we have reached using modern construction techniques (see Figure 1.2). We propose a family of multi-touch gestures, where one can start by sketching and editing the 2D contour of the base floor, and then extend and edit it to create a 3D building model with fairly complex geometric expression in a matter of minutes (see Figure 5.2). The simple drag-and-drop texturing can further enrich the model’s appearance. In addition to 3D modeling, our interface computes and

provides assorted architectural parameters for the building design, e.g., its dimensions, facade area, and volume, thereby providing useful hints for guiding the design process.

Compared to the conventional keyboard-mouse interface, our interface incorporates *hand-oriented actions* such as sketching, slicing, curving, and twisting, which allow one to quickly define the geometry and manipulate its volume (massing). Moreover, we also develop a number of novel interface tools for fast geometric modeling (such as symmetry snapping and easy texturing of image and facade geometry) on the building model, and various editing operations. Thus, our interface can contribute to the conceptual phase of contemporary building design, where building designers and architects can apply it to do quick mockups, try alternative ideas, and get fast feedback to support major design decisions. Moreover, our proposed interface can serve as a rapid 3D modeling tool for creating content in digital media and games, where we need buildings with good-looking shapes and appearance rather than a high level of detail. Such rapid 3D building prototyping is not easily achievable with conventional multi-touch 3D modeling interfaces, existing 3D sketching tools, and general 3D modeling software. To the best

Figure 1.2: Buildings with progressively-varying and twisted structures: *Burj Al Arab* in Dubai, *Taipei 101* in Taipei, *Turning Torso* in Sweden, and *The Shard* in London (left to right).

of our knowledge, this is the first work we aware of to explore and develop multi-touch interactions for rapid creation and editing of tall buildings with non-trivial geometry.

1.3 Thesis Organization

The remainder of this thesis is organized as follow: Chapter 2 reviews the related works of multitouch techniques and surface texturing techniques. Then we give details about three multitouch based applications: an interactive multi-touch sketching interface for diffusion curves in Chapter 3, Texture Brush: an interactive surface texturing interface in Chapter 4 and a multi-touch interface for fast architectural sketching and massing in Chapter 5. The last chapter discusses some future works.

Chapter 2

Related Work

Our work is related to several research topics, including touch techniques, painting, texturing, parameterization, modeling.

2.1 Indirect and Direct-Touch Interaction Techniques.

Touch-based interactions have been frequently compared with mouse-based interactions in user studies [54], such as a selection study. Though touch-based interactions demonstrate advantages in speed by using direct touch techniques [82], precise interactions on touch devices is a challenge due to the occlusion caused by the fingers. Potter et al. [90] developed a take-off strategy for touch screens, which enabled the cursor be slightly up of the user's touch finger when the user was interacting with the touch screen. Also, a click operation was made on release if the user lifted his/her touch finger off the touch screen. This strategy worked well when the target was larger than four pixels. However, if the target was small, the accuracy of a selection is reduced. Sears and Schneiderman [98] made an investigation on how the target size effect on selection accuracy and speed. Their investigation showed that if the width of a target is larger than 0.64 cm, the target could be selected faster with a touch-based interaction than a mouse-based interaction. Accordingly, they explored a method to support pixel-precise selection by slowing down the cursor move speed in the areas around the initial touch point location. However,

for a sketching task, their method would fail because the sketch speed was varying in a single stroke. In 2003, Albinsson and Zhai [3] provided several single-touch widgets for increasing the precision of small target selections. Since the interactions of their widgets were based on single touch point, the user had to complete a sequence of gestures to select a target. A gesture was considered as complete if the user lifted the touch finger off the screen.

2.2 Unimanual and Bimanual Interaction Techniques.

Bimanual interaction techniques were explored by many researchers before. In the late 19th century, Guiard's [39, 40] introduced the Kinematic Chain theory, which was a tentative theoretical framework for bimanual Interaction Technique. Later, Balakrishnan et al. [7] and Hinckley et al. [47] proved the robustness of the Kinematic Chain model by their experiments. In 1986, Buxton et al. [19] investigated bimanual input for multi-touch interactions. The results demonstrated that the degree of parallelism could effect users' speed in selection tasks. Later, Kabbash et al. [53] studied both unimanual and bimanual techniques. Also, the results showed that, if the design of the gesture is inappropriate, bimanual interactions could be worse than unimanual interactions.

Barnert [8] found that, interactions using two mice for the selection task outperformed interactions using one mouse. However, using two hands for the same task is outperformed using single hand. Barnert's task was designed for small target selection tasks; thus, the results of his work were hard to be extended to tasks for larger targets. In 2007, Forlines et al. [30] pointed out that, direct-touch-based interactions might not outperform mouse-based interactions in tabletop devices. They conducted two experiments, which showed that, mouse-based techniques were suitable for precise single-task selection, while direct-touch-based interactions could be used for symmetric bimanual tasks.

Based on these research works, many systems were designed using bimanual interaction techniques. Bier et al. [14] introduced Toolglass and Magic Lenses system, which could help users to reduce steps, cursor motion, and errors of reading interactions by using Toolglass and Magic Lenses tools. Later, Hinckley et al. provided a bimanual-interaction-based system for 3D visualisation. At the same year, Hinckley et al. an multi-touch based interface for map navigation. However, although there are many interfaces from research works were designed for bimanual interaction techniques, most tablets intended to using a dominant hand. Recently, Wagner et al. [109] proposed BiTouch and BiPad, which might change this situation.

2.3 Multi-touch Interaction Techniques.

There has been a lot of work which investigated the performance of multi-touch based interactions. Moscovich and Hughes [79] conducted an experiment to test the efficiency of the multi-touch interface in object alignments. They compared the efficiency of using two fingers on the same hand, and two fingers on separate hands. They found that when performing separable tasks, two handed interaction is more efficient, while for integral tasks unimanual interaction is better. Hancock et al. [43] experimented the performance of the multi-touch interface for a docking task with degrees of freedom up to 5 or 6. They found that for such a task with high degrees of freedom, three-finger interaction leads to the best overall performance, two fingers on the dominant hand and one finger on the non-dominant hand. Kin et al. [59] conducted a set of experiments to compare the efficiency of user interaction with single touch, bimanual and multifinger input. They explored how to effectively use multiple fingers to perform many independent tasks like multiple selection. A lot of studies were presented on two-finger and two-handed interactions for activating various tools, menus and widgets [17, 29, 92, 38, 46, 85, 24, 11].

There were also some work on how to realize a hardware system that enables multi-touch interactions. Matsushita et al. [74] introduced the *HoloWall* system, which allowed user interaction with a finger, a hand, as well as a full body gesture. Their system was based on the infra-red sensing. In 2001, Dietz et al. presented Diamond Touch [75], which could not only detect multiple "touch" movements of a single user, but also enabled multiple users to interact with the desktop. It could also indicate the pressure at each touch point. Rekimoto et al. proposed the Smart Skin [92] system in 2002. Smart Skin could identify the hand touch position, and the shape and distance between the hand and the contact surface. They also proposed some multi-touch techniques which enabled the users to manipulate objects projected on the surface. Malik et al. [72] presented a hand tracking system that allowed the user to interact with the system using multifinger or whole-hand gestures. Jefferson [41, 42] also presented a low-cost multi-touch system based on Frustrated Total Internal Reflection (FTIR).

2.4 Touch-based Applications.

Multitouch techniques [41, 73, 111] have been employed as an intuitive and effective interaction method in wide ranges of applications.

Igarashi et al. [5] developed multi-touch interactions to intuitively manipulate 2D shapes while Bae et al. [5] presented a 3D sketching system for professional designers to create conceptual models with pen and finger gestures. Recently, Lepinski et al. [68] designed multi-touch marking menus that significantly outperform traditional marking menus while Fu et al. [32] proposed multi-touch gestures for exploring large-scale virtual space.

The ability to collect user input simultaneously from multiple fingers can greatly enrich our interaction. Igarashi et al. [49] proposed scenarios for how multi-touch can be used to manipulate and deform complex 2D shapes. Moscovich and Hughes [78] devised

the multi-finger cursor technique, offering smoother interaction with graphical displays. Benko et al. [12] developed a new technique to use multi-touch to improve precision when selecting small targets on a touch screen. Bau et al. [9] enhanced touch interfaces by a new tactile method that used the electro vibration principle to control the electrostatic friction between the finger and surface.

More recently, Frisch et al. [31] developed a novel idea of using grids and guides to improve the precision when positioning and moving graphical objects on a multi-touch system. Bi et al. [13] investigated various theoretical and practical issues when we use a planar multi-touch surface in a desktop computing environment. Findlater et al. [28] analyzed unconstrained typing patterns on a flat glass with a number of expert typists, suggesting that we may personalize such interaction to support eyes-free touch-typing. Kirk et al. [62] evaluated how the multi-touch and physics enabled interactive tabletop can be used in everyday homes and with regular users. Kin et al. [60] investigated multi-stroke marking menus for multi-touch devices and show that using two hands can improve performance. Lü and Li [71] presented a tool that can automatically generate user-modifiable code that detects intended multi-touch gestures and provide callbacks for invoking application actions from a few sample gestures given by the developer. Moeller and Kerne [77] proposed the ZeroTouch, a hardware/software architecture for multi-touch sensing, enabling new forms of interaction. Sato et al. [95] presented the Swept Frequency Capacitive Sensing, allowing minimally-instrumented objects to capture the context of touch interaction.

Touch ways. More than contact points, some researchers suggested to enrich multi-touch interaction by considering shapes of human hands that rest and move on the display surface. Wu and Balakrishnan [115] developed *RoomPlanner*, where multiple users can use their fingers and hands as gestural input on a tabletop to arrange furniture in a floor plan. Wilson et al. [113] modeled human touches by their outlines on the display

to simulate real-time physics with a game engine. Wigdor et al. [110] proposed *Rock & Rails*, a family of simple-to-use shape-based gestures, to enrich direct touch interaction on a multi-touch surface, while Kazi et al [55] developed *SandCanvas*, a multi-touch whole-hand digital artistic medium for performing sand animations. Zhang et al. [123] presented a lightweight method that uses finger orientation for distinguishing touches from multiple users on digital tabletops. Vogel and Casiez [108] examined the shape of hand and forearm occlusion on a multi-touch table for different touch contact types and tasks, created occlusion awareness templates to guide interface layout decisions.

3D Manipulation. Recently, a number of research works have focused on studying the use of multi-touch to manipulate 3D contents. Grossman et al. [38] presented a family of multi-touch gestures to manipulate 3D wireframe objects over a spherical volumetric display. Later, Hancock et al. [43, 44] proposed a multi-touch concept called *Sticky Fingers* to grab and manipulate 3D objects on a tabletop. Reisman et al. [91] formulated a screen-space method, allowing users to directly rotate-scale-translate 3D objects with three or more contact points. Kim et al. [57] designed multi-touch interactions on an iPhone/iPod Touch to control walking in a 3D virtual environment. Chang et al. [20] presented an interesting origami modeling interface by proposing a family of specially-designed multi-touch gestures. Fu et al. [32] proposed multi-touch gestures for controlling navigation in a large-scale astrophysical simulation shown on a wall display. Cohé et al. [23] adopted the use of 3D transformation widgets to transform 3D objects with multi-touch. Kin et al. [61] proposed multi-touch gestures for positioning, arranging, and scaling 3D objects in virtual organic environments, while Chiu et al. [22] explored document folding actions and proposed practical multi-touch gestures for various manipulation scenarios. Compared to the above work, which mainly focuses on the manipulation of 3D contents/views or navigation in virtual space, we aim to create a fast architectural sketching and massing tool for designing tall buildings. Our interface offers a rich set

of novel multi-touch gestures dedicated to efficiently designing a non-trivial geometry (sketching) and manipulating its volume (massing) in a matter of minutes, which has great potential for the conceptual phase of contemporary building design.

3D Modeling.

3D Modeling is a basic part of the scene creation process. There are many approaches of 3D modelling. Basically, 3D modelling can be classified in three categories: polygonal modeling, curve modeling and digital sculpting.

MozArt [99] is a multi-modal interface that supports speech and gesture input to Google SketchUp. Their interface contains only two gestures, i.e., two finger pinch to zoom in/out, and single touch drag to specify the content of modify and operate. All the other modeling and viewing operations are specified via voice. Their interface is very easy to learn due to its simplicity. However, their interface supports only simple geometric shapes such as cylinders and boxes which are extruded from the pre-defined 2D primitives. The user can change only the height of the models. TreeSketch [70] is a system for modeling complex, natural-looking trees using Palubicki et al.'s algorithm [83]. Taking advantages of a multi-touch tablet, the user can directly grow a tree structure by using a few sketches to intuitively specify the direction of the trunk and branches. All the other controls, such as changing parameters as the tree grows, interleaving controlled and autonomous growth, and editing generated forms, are still based on the conventional interface with menus, dialogs, buttons, and scrollbar. In contrast, users of our system can make a complete 3D building by multi-touch gestures alone. Bae et al. [5] presented ILoveSketch, a pen-gesture-based 3D curve sketching system, which enables professional designers to efficiently create conceptual 3D curve models. The system integrates quite a few interaction techniques to facilitate sketch-based modeling, including an automatic view rotation scheme to improve sketching, an axis widget for sketch surface selection, and a set of sketch-based gestures. Later, they improved the sketching system by incorporating many new interactions [6]. Sun et al. [100] proposed an interface which does not

rely on voice input, and supports a much wider range of modeling functions, allowing the users to make non-trivial shapes. Unlike pen-based gesture design concept, their system offers a rich set of direct finger gestures, and allows rapid design of non-trivial modern buildings. In addition, the output of their system can be directly imported into general modeling software such as Autodesk and 3Ds Max.

2.5 Painting and texturing.

Diffusion curves. In 1979, Koenderink et al. [63] informed that most color variations in an image can be assumed to be caused by edges. Inspired from this work, Orzan et al. [81] first introduced diffusion curves as 2D vector-graphics primitives for modeling images with only colors defined along region/curve boundaries; a multi-grid solver was employed to efficiently solve the Poisson equation. Later, Jeschke et al. [51] presented an alternative Laplacian solver; it can run as fast as the multi-grid solver, but is simpler to implement and feasible for arbitrary image resolutions. Moreover, they [52] also extended diffusion curves to 3D surfaces, enabling vector textures on 3D objects. Takayama et al. [105] generalized 2D diffusion curves to 3D diffusion surfaces for modeling anisotropic solid textures. Several alternative methods approximating solutions of diffusion curve images rendering were also explored, such as a ray tracing based approach provided by Bowers et al. [16] and a method based on interpolants by using mean value coordinates provided by Pang et al. [84]. Very recently, Sun et al. [102] introduced diffusion curve textures as a vector representation for accurate rendering diffusion curve images on 3D surfaces in real time.

Pattern Brush. Other than painting with solid colors, the pattern brushes support painting with patterns and texture objects. In image editing software such as Adobe Illustrator, Adobe Photoshop and Vignette system [56], pattern brush interfaces have been proved to be successful for resource re-use in image editing. Such a tool was not

available in 3D till recently, because a texture mapping along the painting sketch is nontrivial. Furthermore, deformation is not supported by these tools. In 2013, Sun et al. [101] introduced Texture Brush, enabling free-form texturing on 3D surfaces. At the same year, Schmidt [96] present a different algorithm for the same purpose.

3D Painting. Interactive 3D painting [45] has been the industry standard of surface texture design for years. The artists interact with the 3D surface directly, without understanding the underlying parameterization. Debry et al. [25] utilized octree-based texture mapping to support painting on the surface without parameterization. Agrawala et al. [1] proposed a system that enables users to paint on a physical object by utilizing a space tracker. Fu et al. [33] introduced a multi-layer approach that enables painting on occluded regions intuitively. Theoretically, 3D painting can be used to authorize arbitrary surface textures. However, it could be tedious work for artists to paint a full model. Also there is no way of reusing existing textures in current 3D painting systems. Our sketch-based texture mapping interface can be used to efficiently create a rich effect of surface texturing based on a small set of texture images. Also our approach can be integrated into existing 3D painting systems seamlessly.

Decal Systems. Decaling is a popular surface texturing method involving pasting small texture images, i.e., decals, on specified regions of the 3D surface. Pedersen [86] proposed an interactive texture design tool with which the copy region can be interactively dragged across the surface. However, the region of the decal can only be constrained by four points and the geodesics connecting them. The decal specifying interface is tedious and different from the 3D painting metaphor. Lefebvre et al. [67] introduced an interactive decaling system by using hardware-accelerated octree textures. Planar projection is employed for local parameterization. However, their method does not guarantee a one-to-one mapping, resulting in various artifacts, such as self-intersection, large distortions, etc. Schmidt et al. [97] introduced the discrete exponential map into interactive decal

compositing. However, users can only control the central point and the orientation of the decal. Geobrush [104] applied the discrete exponential map to the geometry detail cloning, presenting a copy-and-paste interface. The copy-and-paste interface metaphor is also adopted by many existing digital sculpting tools, e.g., Autodesk Mudbox, Pixologic ZBrush. Therefore, these tools share the above limitation. In this paper, we introduce a novel curve-based parameterization method that enables users to control the shape and deformation of the decal by sketching on the 3D surface.

Constrained Parameterization. Constrained parameterization [69, 64] gave a solution for resuing existing texture images. Users are required to specify a set of corresponding feature points as mapping constraints between the source texture image and the target surface. Global optimization is performed to minimize the distortion while fulfilling the constraints. Zwicker et al. [125] extended constrained mapping on point clouds. Zhou et al. [124] proposed TextureMontage to seamlessly map a patchwork of texture images onto a 3D model by generating a texture atlas from the user-input constraints. Tzur [107] introduced a photogrammetric texture mapping method by formulating the mapping estimation as a Moving-Least-Squares problem. Although the interface to specify constraints [36] is intuitive, it is still tedious work to specify a large set of constraints.

Discrete Geodesics and Exponential Map. Computing geodesics on polygonal meshes plays an important role in geometric modeling and computer graphics, such as remeshing [87], non-rigid registration [48], surface parametrization [65] [116] [35] shape editing [122] and shape segmentation [94] and many others. Many elegant and efficient algorithms, such as MMP [76], CH [21], ICH [118], the fast marching method [58], etc, have been proposed to compute single-source-all-destination geodesics. Other types of geodesics, such as geodesic loops [117] [114], geodesic offsets [119], and all-pairs geodesics [120], have also been studied intensively.

Exponential map naturally defines the geodesic polar coordinate system on curved surfaces. Schmidt et al. [97] proposed an $O(n \log n)$ algorithm to approximate the exponential map, which requires only a single additional step in Dijkstra’s shortest path algorithm. Based on the ICH algorithm, Ying et al. [121] computed the exact exponential map on triangular meshes, and applied it to Poisson disk sampling. Our work extends the exponential map to a more general setting in which the generator can be arbitrary curve. Such extended exponential map favors anisotropic texture mapping.

Chapter 3

Painting with 2D Diffusion Curves

3.1 Motivation

Diffusion curves [81] are 2D vector-graphic primitives capable of effectively generating smoothly-shaded images. Basically, each diffusion curve has two pieces of information: 1) curve geometry, often defined by 2D splines; and 2) two sequences of colors, each along one side of the curve. After sketching diffusion curves on a 2D drawing canvas, we can apply the Poisson equation [89] with curve colors as boundary conditions to solve for color at any location on the 2D canvas. Thus, we can spread colors like heat diffusion, e.g., see Figure 3.1 (b&e). Lastly, since splines are used as the curve geometry, resultant drawings can have infinite resolution.

To draw with diffusion curves, users often start with a freehand-style sketch using 2D input devices like mouses and tablets. In the user interface, each sketched curve is first converted into a 2D spline geometry using Bezier splines or B-splines, see Figure 3.1(c). Then, users can specify the colors along each side of the curve, see Figure 3.1(d). Note that such a procedure could be rather tedious if the drawing is complicated and requires spatially-varying colors, meaning that a large number of curves with different colors are needed. In addition, users may also need to further adjust the curve geometry by manipulating the splines' control points one by one. Finally, a Poisson equation solver is invoked in the system to diffuse the colors over the canvas, see Figure 3.1(e).

Recent prosperity of multi-touch technology in the market of consumer computer systems enables general users to directly use their fingers to interact with the graphical contents. Such interactions have been widely manifested in many applications on smart phones and laptops, and demonstrated to be far more intuitive than that with traditional devices.

We develop the first multi-touch based diffusion curve generator. We propose *a novel multi-touch sketching interface* enabling interactive and practical design of 2D vector graphics with diffusion curves. We take advantages of the intuitiveness and efficiency of multi-touch to support simultaneous sketching of multiple diffusion curves and at-the-spot tuning of diffusion colors. In detail, a family of multi-touch gestures is proposed for sketching and editing diffusion curves, as well as for specifying and tuning the curve colors. For instance, users can simultaneously sketch multiple diffusion curves with different colors in our interface, see the rainbow in Figure 3.2 (bottom). Finally, the proposed interface is developed into a working system and experimented with a number of users, including both professional artists and novice users, demonstrating its values as a practical drawing tool. Our experiments show that this not only brings novel painting experience to users but also provides a practical and effective tool for vector graphics design, useful for styles like silk painting, Disney cartoon, art poster, and photo-realistic effects.

3.2 Multi-touch Sketching Interface

Our proposed multi-touch sketching interface supports the following four major operations:

1) Select Colors with Multi-touch. Each diffusion curve has at least four colors, two on each side of it, to produce spatially-varying colors (by linear interpolation) along the curve geometry, see Figure 3.1 (a&d). In our proposed interface, users can *multi-touch the color palette* with four fingers (see, Figure 3.3 (left)) to specify these boundary

Figure 3.1: Diffusion curves (see (a)) enable effective generation of smoothly-shaded 2D vector-graphics (see (b)). After sketching a spline (see (c)) and specifying colors on it (see (d)), we can apply a Poisson equation solver to generate the drawing on the canvas (see (e)).

Figure 3.2: Our multi-touch sketching interface for diffusion curves: (a) multi-touch to select colors; (b) one-finger sketch to form a diffusion curve; (c) the second finger can help to adjust the curve geometry during the sketch; (d) complete the single curve; (e,f,g) sketch two additional curves to form a rainbow with multiple-fingers movement simultaneously; note that each curve (finger) can carry different colors during such an action; (h) our system setup: a 19" multi-touch screen and a Dell workstation; the lower-right corner shows the final image and the diffusion curve geometry.

colors. And to make it more practical, users can also flexibly select or adjust one, two, three, or four colors by simultaneously touching the palette with respective number of fingers, and our system determines the color-touch match by the order of finger-screen contacts; moreover, users can further apply multi-touch to exchange colors of curves over the drawing canvas.

2) Sketch Multiple Curves with Multi-touch. In addition, users can also use multi-touch to sketch multiple diffusion curves simultaneously. Our design is motivated by the following observations on finger and hand ergonomics:

- Fingers of the same hand can naturally move together (in parallel) to sketch *curves with similar shape* at the same time, e.g., rainbow and water wave;
- When using both hands, the non-dominant hand can easily follow the dominant hand so that associated fingers from both hands can easily draw *mirror-reflected shapes*.

To sketch multiple diffusion curves in practice, user can first subsequently select colors for each curve he/she planned. Associated colors for each curve are immediately shown next to the curves' endpoints as visual feedback to users. After that, users can exercise the above ideas to *simultaneously sketch multiple diffusion curves* with one or two hands on our interface, e.g., we found that rainbow can be easily drawn with multiple fingers of one hand while the heart shape can be easily drawn with fingers from two hands.

3) Adjust Colors with Multi-touch. Thirdly, users can adjust curve colors while sketching a diffusion curve. In detail, *hue, lightness and saturation* can be interactively adjusted by first multi-touching (with another finger) an empty location near the circular color tag followed by a finger drag. As shown in Figure 3.3 (right), dragging perpendicular and parallel to the curve's local tangent adjust the hue and lightness, respectively, at the related color tag based on the finger displacement. Performing angular change

Figure 3.3: Multi-touch to select colors (left) and adjust colors (right).

towards the opposite direction of lightness adjusts the saturation. Compared to conventional painting interfaces which require switching to the palette panel window for color tuning, users of our interface can directly adjust colors at the spot. This strategy helps *minimize contextual switch*, thereby allowing users to concentrate on their designs in the drawing canvas.

4) Additional Features. Lastly, users can also directly edit the control points and curves in a way similar to other works, e.g., see [49]. Moreover, since diffusion curves are vector graphics primitives with infinite resolution, our interface has a zoom mode for users to view and edit the same drawing at different scales. Supported with a real-time GPU-based multi-grid solver running on a high-performance graphics board, we can diffuse colors interactively while users change the drawing scale. Users can thus sketch fine details with higher precision on important parts in their drawings.

3.3 Implementation and Results

Hardware. Our multi-touch sketching interface was implemented and experimented on a Dell workstation T3500 with a 2.67GHz Xeon CPU, 12GB memory, a GeForce GTX295 (1792MB) graphics card, and a 3M 19" multi-touch display. See Figure 3.2 (lower right) for the hardware setup.

In our implementation, the basic pipeline is 1) user input, 2) diffusion by multi-grid solver.

User Input in Software.

We use 3M MT7Trace API for user input. There are two objects of the API, trace object, and event object. The MT7Trace API uses a callback system to inform the application that an event occurred within the touch system. In our system, there are two reasons why the API invokes a callback:

- New trace creation.
- New event on an existing trace.

A trace represents a series of events produced on a sensor by a single touch. Each trace is composed of events, including touchdown event, drag event and liftoff event, as shown in Figure 3.4. All traces begin with a single touchdown event. It occurs when a finger first makes contact with the sensor. If the finger moves across the sensor, there will be drag events that are added into the trace. Finally, when the user removes the finger, the trace receives a liftoff event.

An event (Figure 3.4) corresponds to some activity on the sensor. There are three types of events.

- A touchdown event starts a trace. It represents where a finger first makes contact with the sensor.
- There may be any number of drag events in a trace. Each drag event represents a change in the location of a finger on the sensor while the finger is still in contact with the sensor. It is possible for a trace have no drag event if the finger does not move before lifting off.

- A liftoff event completes a trace. It represents the location on the touch sensor where the finer of stylus broke contact. An application should release a trace handle when it sees a liftoff event.

When an application first retrieves a trace handle, it has at least one event in it, the touchdown event. In addition, a unique trace ID will be set to this trace, which allows us to follow events of the trace. It may have other events in it, even a terminating liftoff event if the API receives the liftoff event before the application can retrieve the trace's events. If a trace handle does not have the liftoff event, then the application must obtain updates to the trace.

In practice, we use a “map” structure to connect a trace and events that belong to it. When a new trace starts, we construct a new map, of which first integer number represents trace ID and the second integer number represents number of events. Once a new event is detected, the second number of the map will increase one. Then, we add the event to the corresponding trace, including the position on the screen and the event ID.

In our implementation, the curve data is updated with user's fingers. Thus, user can have a preview of final image when the user is painting. By using “map” structure, we implement the multiple stroke function. User can use more than one finger to paint on the display simultaneously. Note that, the process of multi-touch is multi-thread process, if we do not use “map” structure to connect a trace and events, the results will be unpredictable.

Multi-grid solver for diffusion curves.

Diffusion curves [81] combine curve primitives with a diffusion method that smoothly spreads color from the curves across the image. The fact that the representation relies on a small number of simple entities makes it particularly well suited for artists. Diffusion is the key component that enables a rich, yet simple definition of resolution-independent

Figure 3.4: Different levels of the same image.

illustrations. Diffusion processes often underlain the method for solving the Poisson equation, as its solution minimizes the deviation from a given gradient field. The Poisson equation also is the basis of diffusion curves, where colors are sparsely defined along curves in the image and interpolated everywhere else.

Solving Poisson equation. Diffusion operation is based on Poisson-equation framework, which has been applied to many applications [81]. Consider an image I , colors of it are addressed individually as a grid simply as $I_{x,y}$, where x, y are the coordinates of a pixel in the image. The goal is to solve the Poisson equation for image I with sets of pixel colors $\{C_{i,j} \mid i, j \in \mathcal{I}\}$ as constraints, where $\mathcal{I}$ contains a set of pixel positions represented by coordinates of constrain pixels, and having a gradient close (L_2 -norm) to a given vector field $w = \{w_{i,j} \mid i, j \in \mathcal{I}\}$. The vector field values w can be represented by $w_{i,j} := (w_{i,j}^x, w_{i,j}^y)$, where $w_{i,j}^x$ and $w_{i,j}^y$ are the differential values along x and y axis respectively.

In the case of diffusion curves, colors are specified along each side of the curve as hard constraints. In addition, the vector field w is zero everywhere except across constraint curves. In other words, the solution will show a continuous, smooth change of colors except across hard constraints.

The image I is defined implicitly using the Poisson equation:

$$\begin{cases} \Delta I = \text{div } w \\ \text{and } I_c = C_{i,j} \forall (i,j) \in \mathcal{I} \end{cases} \quad (\text{Eq. 3.1})$$

where Δ represent the Laplace operator, and div is the divergence operator. Since a RGB image usually has three channels, red, green and blue, we need to solve the equation for each color channel. There are several options to solve the equation efficiently, such as using Gauss-Seidel solver, conjugate gradient and multi-grid solver [89]. Since multi-grid solver is efficient enough to achieve real-time interaction, and can be implemented easily, we use Jacobin method optimized by a GPU based multi-grid solver to solve the equation.

The Jacobi method. Jacobi method can be used to solve the discrete Poisson equation by applying Jacobi iterations. The end goal of applying discrete Poisson equation to an image is to minimize the Laplacian $\nabla^2 G$ of the image. Adding fixed pixels, such as diffusion curves, imposes color constraints $C_{x,y}$ on the system:

$$\begin{cases} G(x, y) = C(x, y) & \text{if } (x, y) \text{ is on a diffusion curve, and} \\ \nabla^2 G(x, y) = 0 & \text{otherwise} \end{cases} \quad (\text{Eq. 3.2})$$

where (x, y) is the position of an image pixel, and C demotes sets of constraint pixels.

The Laplacian operator is discretized as

$$\nabla^2 G(x, y) = \sum_{i=1}^4 G(n_i) - 4G(x, y), \quad (\text{Eq. 3.3})$$

where n_i is a pixel with the 4 surrounding connected neighborhood of (x, y) . A standard Jacobi solver iterates toward the solution by setting each pixel to the average of its direct neighbors:

$$G(x, y) = \frac{1}{4} \sum_{i=1}^4 G(n_i). \quad (\text{Eq. 3.4})$$

Diffusion curve constraints are maintained by fixing pixel color on the curve. However, the basic Jacobi iteration approach converges quite slowly; our interface cannot achieve real-time interaction by applying it directly to our system.

A pseudo-code fragment is shown here to describe the algorithm of one iteration of Jacobin method, where variable “Image” represents the image which user can paint on, and variable “Constraints” represents an image with constraint curves. To determine whether a pixel is the constraint, we can check the value of w , which represent the transparency of a pixel. If the pixel is the constraint, the value of w is equal to one, or else the value of w is equal to zero. Function `texture2DRect(Img, Coord)` returns the RGB colors of a pixel with the coordinate “Coord” in the image “Img”.

Jacobin(Image, Constraints)

```
1      coord ← FragmentCoord
2      constraint ← texture2DRect(Constraints, coord)
3      if constraint.w > 0.01
4          then FragmentColor = constraint
5      else
6          a = texture2DRect(Image, coord + (-1, 0))
7          b = texture2DRect(Image, coord + (1, 0))
8          c = texture2DRect(Image, coord + (0, -1))
9          d = texture2DRect(Image, coord + (0, 1))
10         FragmentColor = 0.25*(a + b + c + d)
11     end if
```

The Multi-grid Solver. In order to solve the discrete Poisson equation at an interactive speed, we implemented a multi-grid solver [89] based on the GPU (graphics processing unit). The rough idea is to solve low-frequency components with a coarse domain and then refine high-frequency components iteratively with a finer domain.

As described in previous paragraph, discrete Poisson equation ($\Delta I = \text{div}(G)$) can be

solved by using Jacobi iterations. ΔI is expressed as

$$4I(x, y) - I(x - 1, y) - I(x + 1, y) - I(x, y - 1) - I(x, y + 1),$$

which gives:

$$4I(x, y) - I(x - 1, y) - I(x + 1, y) - I(x, y - 1) - I(x, y + 1) = \text{div}(G),$$

which leads to:

$$I(x, y) = [\text{div}(G) + I(x - 1, y) + I(x + 1, y) + I(x, y - 1) + I(x, y + 1)]/4,$$

$\nabla^2 G(x, y)$ is computed in a preprocess as:

$$\nabla^2 G(x, y) = 0.5 \times (G_x(x - 1, y) - G_x(x + 1, y)) + 0.5 \times (G_y(x, y - 1) - G_y(x, y + 1)),$$

with G_x and G_y the x and y derivatives of the image.

The discrete Jacobi method is implemented as an iteration process. The level zero image I is initialized with the grey value (150, 150, 150). Each Jacobi iteration consists in computing $I(x, y)$ with the values of I from the previous iteration.

A local constraint of value v_i and position (x_i, y_i) is imposed after each iteration by setting

$$I(x_i, y_i) = v_i$$

.

Since Jacobi iterations are usually very slow to converge, we use a multi-scale approach: the multi-grid solver.

We down sample the image I several times until we reach a very coarse solution, as shown in Figure 3.5. Then compute the gradient field G at this coarse solution by applying limited number of Jacobi iterations. Then we up sample the coarse solution and perform a few more iterations. This process is repeated until we reach the original resolution.

In our implementation, we apply a fixed number of Jacobi iterations for each level image. The local constraints are down sampled with an average filter: a pixel at the coarse scale receives the average of the four corresponding pixels in the finer scale.

Figure 3.5: Different levels of the same image.

Results. Figure 3.6 shows three example vector graphics drawings by an artist using our interface: Disney cartoon, silk painting and photo-realistic effect. As demonstrated in the accompanied video, our system can support interactive painting on a canvas of resolution 780×780 ; with the GPU support, it can update the visuals with multi-touch interaction at 20Hz, e.g., with the examples shown in Figure 3.6.

3.4 User study

We conducted a user study to explore the benefits of our proposed multi-touch interface for general users. Our interface is compared against the conventional diffusion curves sketching interface provided by Orzan et al. [81]. As geometry and color are the two key elements in diffusion curves painting, our goal is to measure how the interfaces help users to specify these two elements. Note that experienced artists and novice users may have significantly different painting styles that could lead to inaccurate measurements in the study. To counteract such discrepancy, strokes employed in our experiments are simple and distinctive. And after consulting with several artists and pilot-testing with

Figure 3.6: Paintings created by an artist with our interface: Disney cartoon, silk painting, and photo-realistic effect can be readily produced (from left to right). The artist used this system for the first time and was given 5 minutes in learning. The painting time varies from 15 to 40 minutes.

some users with different levels of painting skill, two experiments with varying difficulty are designed.

Participants. Twenty-one participants were recruited in our user study, and they were divided into two groups of roughly equal sizes. The first group (G1) had 5 males and 5 females with an average age of 27.1, whereas the second group (G2) had 7 males and 4 females with an average age of 25.8. Among them, six participants are professional 2D and 3D artists, whereas others are novice users. These professional artists were split randomly and equally into the two groups.

Overall Procedure. Before going to tasks in the two experiments, we first introduced the key concepts of diffusion curves to the participants, and gave them 5 minutes to get familiar with the sketch-based input device. There are two stages in the user study. In the first stage (for the first group), we first showed our proposed interface to the participants, and gave them five minutes to try the operations in the interface. Then, they were asked to complete two painting tasks (two experiments) with it. After that, we moved on to the second stage and introduced the conventional interface to them. Again, we gave them another five minutes to try the interface. After that, they were asked to do the (same) two experiments but with the conventional interface this time. To eliminate potential discrepancy about familiarity with the interfaces, the second group did exactly

the same tasks but in a reverse order when using the two interfaces. During the course of the user study, the time taken by each participant to complete the experiment task was measured. In addition, to help the participants (especially the novice users) complete the tasks, we showed them the expected diffusion results before starting the experiment tasks.

Experiment #1: The first experiment is to evaluate the performance of specifying and adjusting colors on a simple diffusion curve. Participants were asked to first sketch an arbitrary stroke with four specific colors, and then increase the brightness of one color and decrease the others, see Figure 3.7 (left). Since a number of successive interface clicks are required to specify colors with the conventional interface, adjusting curve colors on an existing diffusion curve using the conventional interface is much more tedious.

In sharp contrast, participants of our multi-touch interface can directly use their (four) fingers to select colors simultaneously. To adjust colors, participants can simply touch one endpoint with a finger and then move another finger to a certain direction to control hue, saturation and lightness. With the multi-touch interface, participants can directly specify and adjust colors at the spot without switching frequently between the canvas and color palette. Furthermore, since the resultant drawing can be updated in real-time with the multi-touch interaction, immediate visual feedbacks can be provided to further improve users' performance. As shown in Figure 3.8, the average time to complete this experiment with multi-touch interface is 2.5 times shorter than that with the conventional interface.

Experiment #2: The second experiment compares the multiple-curve sketching function (in our multi-touch interface) with the single-stroke sketching function (in conventional interface). Sketching multiple diffusion curves with similar shape is a very common task in many practical scenarios, e.g., water waves, hairs, furs, and rainbows. In experiment 2, participants were asked to sketch a rainbow with three strokes (near

Figure 3.7: Tasks in Experiment 1 (left) and Experiment 2 (right).

parallel) together with six different boundary colors, see Figure 3.7 (right). With conventional interface, participants have to sketch three curves one after the other, thus the curves may not result naturally with similar shapes. However, the multi-touch interface allows the participants to sketch the three curves in the rainbow simultaneously with three fingers. This can save up to roughly 70% of drawing time as compared to the conventional interface, see Figure 3.8.

Paired t-test. To analyze the user study data, we conducted the paired t-test with the null hypothesis H_0 : the mean values of the drawing time using two interfaces are equal. We used a significance level of 0.01, given H_0 is true. As shown in Table 3.4, all resultant t values are larger than the critical value 2.72 with degree of freedom $DOF=11$ from the t-test table; hence we can reject H_0 and show that our interface does perform better and help.

DOF=11	Experiment #1		Experiment #2	
Group	G1	G2	G1	G2
t value	8.9	11.03	7.12	6.74

Table 3.1: Resultant t-value.

Rating. After the two tasks, the participants were asked to rate the two interfaces on

Figure 3.8: User study results. C: conventional; M: multi-touch.

their “effectiveness” and “interesting to use” on a scale of 1 to 5: 1 means “very hard to use, tedious, not satisfied at all” and 5 means “easy to use, highly effective, completely satisfied.” As a result, the multi-touch interface receives an average score of 4.5 and 4.7 for “effectiveness” and “interesting to use,” respectively, as compared to 3.2 and 2.9 for the conventional interface.

The participants also commented that the multi-touch interface is highly intuitive for setting and tuning multiple colors, and the painting with multiple strokes in Experiment #2 is very novel; even the professional painters had never experienced this with existing 2D/3D painting systems before.

3.5 Summary

This paper presents a novel multi-touch sketching interface for efficient design of 2D vector graphics drawings with diffusion curves. In sharp contrast to conventional interfaces that require tedious working steps for constructing diffusion curves, our interface

supports a family of new multi-touch gestures, enabling users to simultaneously sketch multiple curves, to edit curve geometry, as well as to interactively edit and tune the curve colors at the spot. Using a working system we developed, we conducted a user study to demonstrate the practical value of this interface design with both professional artists and novice users. Results show that this novel design not only brings new painting experience, but also provides a practical and effective tool for vector graphics design, typically useful for styles like silk painting, cartoon, and photo-realistic effects.

Chapter 4

Surface Painting with Texture Brush

Figure 4.1: Texture brush painting on models. Left: the cup is textured with 15 painting operations using 8 texture brushes. Right: the fertility model is textured with 8 painting operations using 3 texture brushes. The colored lines indicate the painting strokes.

4.1 Motivation

Texture mapping is an effective tool to generate realistic and visually rich images by adding surface material attributes. It is one of the important stages in the computer graphics pipeline. However, texture design is generally a manual work and requires a significant amount of effort. The conventional surface texturing pipeline consists of three steps: assigning texture coordinates to the mesh vertices (parameterization), texture editing according to the mapped polygons, and rendering the textured model. Because the three stages are often performed iteratively and views of these stages are generally separate, artists must switch back and forth among these stages [36]. Both the efforts for

the texturing process and skills required by the complicated interface prevent inexperienced users from carrying out surface texturing. Therefore, it is desirable to develop an integrated texturing interface that naturally combines texture mapping, texture editing, and 3D rendering.

In recent years, efforts have been made to build intuitive interfaces, eliminating the separation among these stages. The 3D WYSIWYG painting system [45] allows the users to paint directly on the surface and create textures by sketching on surfaces. This intuitive interface has been the real workhorse of interactive texture design for decades. However, the parameterization stage is still separated from the interface. Thus, texture mapping operations such as copy-and-paste, are unavailable and it is not known how to re-use existing textures. The decaling system [86, 97] provides a practical solution for texture mapping and painting. It allows the user to interactively specify the location, orientation and scale of the textures, which can be mapped onto the 3D surface by using exponential map. The decaling interface works pretty well for isotropic and small textures, however, it may fail for anisotropic textures, due to the large distortion. Furthermore, the existing systems do not allow the user to edit the mapped textures on the 3D models. As a result, the users have to modify the texture image in the separate 2D view. Besides the technical challenges, the cognition gap between the pasting metaphor and the 3D painting metaphor prevents the current texture mapping interface and the 3D painting interface from being merged with each other naturally.

We introduce the painting metaphor into interactive texture mapping and propose a painting-based texture mapping interface. As demonstrated in Figure 4.1, the user can specify the texture mapping by a single stroke. Because the interaction metaphor is the same as conventional 3D painting, artists' experience of 3D painting can be transferred to our texture mapping interface naturally. Artists can focus on the high level semantics of texture mapping, such as the general shape, without paying attention to the parameters of the underlying parameterization. Our specific contributions include:

- We propose an extended exponential map algorithm enabling local parameterization in a strip along a curve; anisotropic texture mapping is supported;
- We introduce the 3D painting metaphor into interactive texture mapping; the texture mapping interface and texture painting/editing interface are integrated naturally; the proposed system provides a more natural user experience and smoother work flow;
- We propose a set of powerful texturing interactive tools, including texture painting, texture cloning, texture animation design and texture map editing.

Our work is related to several research topics, including 3D painting, constrained parameterization, decaling interfaces, image pattern brush, discrete geodesics and exponential map.

3D Painting: Interactive 3D painting [45] has been the industry standard of surface texture design for years. The artists interact with the 3D surface directly, without understanding the underlying parameterization. Debry et al. [25] utilized octree-based texture mapping to support painting on the surface without parameterization. Agrawala et al. [1] proposed a system that enables users to paint on a physical object by utilizing a space tracker. Fu et al. [33] introduced a multi-layer approach that enables painting on occluded regions intuitively. Theoretically, 3D painting can be used to authorize arbitrary surface textures. However, it could be tedious work for artists to paint a full model. Also there is no way of reusing existing textures in current 3D painting systems. Our sketch-based texture mapping interface can be used to efficiently create a rich effect of surface texturing based on a small set of texture images. Also our approach can be integrated into existing 3D painting systems seamlessly.

Constrained Parameterization: Constrained parameterization [69, 64] gave a solution for reusing existing texture images. Users are required to specify a set of corresponding feature points as mapping constraints between the source texture image and the

target surface. Global optimization is performed to minimize the distortion while fulfilling the constraints. Zwicker et al. [125] extended constrained mapping on point clouds. Zhou et al. [124] proposed TextureMontage to seamlessly map a patchwork of texture images onto a 3D model by generating a texture atlas from the user-input constraints. Tzur [107] introduced a photogrammetric texture mapping method by formulating the mapping estimation as a Moving-Least-Squares problem. Although the interface to specify constraints [36] is intuitive, it is still tedious work to specify a large set of constraints.

Pattern Brush: Our interface is also inspired by the pattern brush in image editing software such as Adobe Illustrator, Adobe Photoshop and Vignette system [56]. Other than painting with solid colors, the pattern brushes support painting with patterns and texture objects. This interface has been proved to be successful for resource re-use in image editing. However, such a tool is not available in 3D because a texture mapping along the painting sketch is nontrivial. Furthermore, deformation is not supported by these tools.

Decal Systems: Decaling is a popular surface texturing method involving pasting small texture images, i.e., decals, on specified regions of the 3D surface. Pedersen [86] proposed an interactive texture design tool with which the copy region can be interactively dragged across the surface. However, the region of the decal can only be constrained by four points and the geodesics connecting them. The decal specifying interface is tedious and different from the 3D painting metaphor. Lefebvre et al. [67] introduced an interactive decaling system by using hardware-accelerated octree textures. Planar projection is employed for local parameterization. However, their method does not guarantee a one-to-one mapping, resulting in various artifacts, such as self-intersection, large distortions, etc. Schmidt et al. [97] introduced the discrete exponential map into interactive decal compositing. However, users can only control the central point and the orientation of the decal. Geobrush [104] applied the discrete exponential map to the geometry detail

cloning, presenting a copy-and-paste interface. The copy-and-paste interface metaphor is also adopted by many existing digital sculpting tools, e.g., Autodesk Mudbox, Pixologic ZBrush. Therefore, these tools share the above limitation. In this work, we introduce a novel curve-based parameterization method that enables users to control the shape and deformation of the decal by sketching on the 3D surface.

Discrete Geodesics and Exponential Map: Computing geodesics on polygonal meshes plays an important role in geometric modeling and computer graphics, such as remeshing [87], non-rigid registration [48], surface parametrization [65] [116] [35] shape editing [122] and shape segmentation [94] and many others. Many elegant and efficient algorithms, such as MMP [76], CH [21], ICH [118], the fast marching method [58], etc, have been proposed to compute single-source-all-destination geodesics. Other types of geodesics, such as geodesic loops [117] [114], geodesic offsets [119], and all-pairs geodesics [120], have also been studied intensively.

Exponential map naturally defines the geodesic polar coordinate system on curved surfaces. Schmidt et al. [97] proposed an $O(n \log n)$ algorithm to approximate the exponential map, which requires only a single additional step in Dijkstra’s shortest path algorithm. Based on the ICH algorithm, Ying et al. [121] computed the exact exponential map on triangular meshes, and applied it to Poisson disk sampling. Our work extends the exponential map to a more general setting in which the generator can be arbitrary curve. Such extended exponential map favors anisotropic texture mapping.

4.2 Extended Exponential Map & Local Parameterization

The core of our texture brush interface is the underlying local parameterization which is aligned to a specified curve. The main challenge of our painting-based interface is to depict the shape of the texture object with only one sketch. Although the sketch

Figure 4.2: Extended exponential map. (a) The user sketches a curve on the surface. (b) Geodesic offsets of the curve. (c) The integral curves are the geodesics; large distortions can be viewed in the close-up view. (d) Local parameterization along the curve. (e) The fixed tracing lines by harmonic maps. (f) The fixed local parameterization.

drawing is considered to be consistent with the user’s perception of shape, a local parameterization along the sketch expressing the shape faithfully is not straightforward. Existing parameterization methods either introduce large distortions or do not support the mapping constraints.

This section presents an extended exponential map algorithm. The parameterization is defined by a geodesic coordinate system based on the user-sketched curve. Thus, the promising properties of the exponential map such as the balance between conformal and area-preserving parameterizations are inherited. Furthermore, the shape of the source region can be semantically depicted by the specified sketch as the major skeleton and preserved in the parameterization space.

Our local parameterization algorithm is as follows:

Step 1. Given a user input curve γ on the 3D surface (see Figure 4.2a), we compute

the geodesic offsets of the curve γ (see Figure 4.2b) using the method in [119]. Each geodesic offset is given a 1D coordinate as v according to the value of geodesic field.

Step 2. Curve γ is parameterized according to arc length. Also we uniformly sample curve γ , each sampling point p having a unique 1D coordinate u_p in u direction.

Step 3. For each sampling point p in γ , we trace the integral curve α of the gradient vector field to the outer most geodesic offset. The curve α_p starting from point p is assigned a 1D coordinate u_p in the u direction. As a result, each point in the geodesic field is given two coordinates (u, v) as the initial parameterization (see Figure 4.2cd).

Step 4. The initial parameterization may not be bijective in the highly curved region. For each point p (see Figure 4.3(a)) in the curve γ , we define $r = \frac{|u_s - u_t|}{l(o_s, o_t)}$ to measure how the curved region bends at p , where s and t are two points around p in γ , $d = u_s - u_p = u_t - u_p$ is a short distance, o is the outer most geodesic offset, points o_s and o_t correspond to s and t respectively, and $l(o_s, o_t)$ refers to the distance from o_s to o_t along the geodesic offset. The basic idea is that if the curved region around p is highly curved, the nearby curve segment is obviously longer or shorter than the corresponding segment in the outmost geodesic offset. In the straight line case, $r = 1$ for each point. We define a point p as curved point if $r_p > 1 + \lambda$ or $r_p < 1 - \lambda$, where λ is 0.2 in our implementation. A highly curved region is defined by a sequence of continuous curved points, from the first to the last curved point along the curve.

Step 5. After detecting the highly curved strip region M , we fix its parameterization by mapping it onto a rectangle parameterization space P using harmonic map [27]. As shown in Figure 4.3(b), rather than computing the map directly, we first parameterize M to the unit disc, say $f : M \rightarrow \mathbb{D}$ such that $\triangle f = 0$. The boundary condition is given by arc length parameterization. In detail, the four corners of M are mapped to four points in the boundary of $\mathbb{D}$ and the segments between them are parameterized by arc length. Similarly we also map P to the unit disc using harmonic map $g : P_i \rightarrow \mathbb{D}$. It

Figure 4.3: (a) Measuring how the curved region bends. (b) Parameterization for highly curved strip region.

worth noting that the four corners of M and P are mapped to the same points in the unit disc correspondingly. Then an identity map i is given between the two unit disks. Then the map between M and P is given by $\phi = f \circ i \circ g^{-1}$. Finally, the refined local parameterization with less distortion is achieved.

Figure 4.4: Our extended exponential map can achieve high quality parameterization with low distortion.

Figure 4.4 illustrates the local parameterization results according to various painting sketches on different surfaces. The results show that our algorithm is robust on highly curved surfaces, on which other methods would fail.

Remark: Our method can guarantee that the extended exponential map is bijective. This is because the non-bijective region is fixed by a harmonic map which maps the

curved quadrilateral to a rectangle. According to Redo's theorem, the target domain (the rectangle) is convex, and the boundary map is homeomorphic, thus the harmonic map is a diffeomorphism. As we use the arc length parameterization to specify the boundary condition, the harmonic map can be glued with the rest of the parametric domain in a seamless manner. Thus, the whole map is bijective.

4.3 Texture Brush Interface

Figure 4.5: The user interface. (a) A snapshot of our interface; (b) We implement our texture brush on a multi-touch display so that all the operations can be done by using multi-touch gestures.

We present an intuitive anisotropic texture mapping interface which can be integrated into existing 3D painting systems seamlessly. Our interface supports a set of interactive operations, including painting-based texture mapping, painting-based texture cloning, texture animation design and texture map editing. Figure 4.5 shows the interface of our system.

Since sketch is the underlying operation in texturing brush system, we design a sketch-based interface for our system to make sketch interaction more intuitive and easy to control. Users can sketch on the screen directly by using fingers. Also, during the process of observing and texturing the model, user can benefit from classic multi-touch

Figure 4.6: Texture painting. (a) The user sketches on the mesh surface; The red line illustrates the painting stroke; (b) Extended exponential map corresponding to the input curve; (c) The texture mapping result.

operations, such as zooming in by using two fingers. This means users don't need to switch between the zoom in mode and the texture mode by tapping a button far away from the texturing area. This can make the texturing process more immersive.

4.3.1 Texture Painting

The basic function of our system is sketch-based anisotropic texture mapping (See Figure 4.6). Given a 2D texture image as input, the user sketches an arbitrary curve on the surface, and the texture is mapped on the surface along the curve. The width of the texture can be automatically determined by the length of the curve and the aspect ratio of the texture image. The user is allowed to adjust the texture width interactively. The interaction in texture painting is similar to that in 3D painting systems. Thus the artists' experience in 3D painting can be transferred smoothly to our interface. Compared to decaling systems [97], our interface is more flexible. Based on the same texture image, one can easily obtain different texture mapping results by drawing different curves on the surface.

Figure 4.7: Texture cloning. (a) The user paints on the texture object; (b) The extracted texture in image space; (c) The matted texture object with alpha mask; (d) The user paints on the target surface with the texture brush; the texture is composited with the background texture.

4.3.2 Texture Cloning

Our system provides an interface to clone a texture patch from the source model and map it onto another target model. As demonstrated in Figure 4.7, the texture cloning contains two steps. The first step is to sketch on the source model with a “cloning” brush. The stroke generally indicates the region information of the texture object, such as location, shape, and also the width. After the sketch operation, the foreground texture object is segmented with alpha matting and mapped to the image space automatically. The second step is to apply the texture painting as introduced in the previous section, using the extracted texture image. With alpha blending, the mapped texture can be composed with the background color of the target model seamlessly.

Figure 4.8: Texture animation. (a) The yellow curve shows the route of animation, and the red curves are animated control sketches at different time stamps; (b) snapshots of the animation at different times.

The first stage of texture clone is to extract the texture object from existing textured surfaces or images using only one sketch. Given a user specified sketch, we can map the texture object from the source region to the image space using our curve-based local parameterization. The extracted texture object in the image space is bound to a new brush as the texture image. In the second stage, users can paint with the created brush on the target region. The texture object is smoothly cloned from the source region to the target region by only two painting operations. Figure 4.7 demonstrated the whole process of texture clone.

Besides the underlying local parameterization, another challenge is the texture object extraction. The texture objects are always composed with the background in the source region. However, the background of the target region is always different from the source. Artifacts would occur if we cloned the whole brush region from the source to the target. We present a fully automatic texture object extraction algorithm to address this problem. Given a texture image containing a set of foreground objects, our method involves the following 4 steps:

Step 1. We use the GrabCut [93] algorithm to generate a hard segmentation of the texture image. The GrabCut algorithm needs the user to specify a bounding rectangle of the foreground object. We automatically set one by shrinking the image frame by a few pixels.

Step 2. The hard segmentation generated by GrabCut may contain small holes and jagged artifacts on the segmentation boundary. We use image morphology operations, erosion and dilation, to divide the image pixels into three sets, i.e., foreground $\mathcal{F}$, background $\mathcal{B}$ and unknown $\mathcal{U}$.

Step 3. For each pixel $P_i \in \mathcal{U}$, we estimate an initial alpha value $\hat{\alpha}_i = p_f / (p_f + p_b)$, where p_f and p_b are the probability of the pixel belonging to the foreground and the background, respectively. The probabilities are computed with the Gaussian Mixture Model (GMM) obtained in the GrabCut algorithm.

Step 4. Finally, we optimize the alpha values of the pixels in the unknown region by solving the following energy function:

$$\begin{aligned} & \underset{\alpha_i}{\operatorname{argmin}} \sum_i (\alpha_i - \hat{\alpha}_i)^2 + \lambda \sum_{j \in N_i} (\alpha_i - \alpha_j)^2 \omega_{ij}, \\ & \text{s.t.} \begin{cases} \alpha_i = 1, & p_i \in \mathcal{F} \\ \alpha_i = 0, & p_i \in \mathcal{B} \end{cases} \end{aligned}$$

where $\omega_{ij} = e^{-\frac{\|C_i - C_j\|^2}{2\sigma^2}}$. The optimization smoothes the alpha values estimated above and makes the neighboring pixels with similar colors have similar alpha values. The most time-consuming step of our approach is step 1. For a common image with the resolution of 512×512 , the whole process takes about 0.4 seconds in our test platform. Figure 4.9 shows the whole process and the result of texture cloning.

Figure 4.9: Texture cloning from the source model (a) to the target model (d).

4.3.3 Texture Animation

Besides the static texture mapping, we propose a simple interface which allows the user to create a vivid texture animation on the surface. After setting the width and length of the mapping texture, the user sketches a long stroke on the surface, indicating the path of the texture animation. The path can be an open curve or a closed loop. Then

the texture image is mapped to a segment of the stroke and animated along the stroke. Figure 4.8 shows a texture animation created by our interface.

Users can easily design texture animation along a route on the surface using our interface. Given a user-specified route, the texture object is mapped according to a section of the route. Then the target section moves forward along the route and the texture object is animated. Because the underlying local parameterization is mainly based on geodesic distance which is intrinsic, the painted texture object animates across the surface without large distortion.

Figure 4.10: Texture brush curve editing. The user drags the control points on the base curve to modify the shape of the texture mapping.

4.3.4 Texture Map Editing

To enhance our texture mapping interface, we present two tools, curve editing and texture deformation, which enable the user to modify the mapped texture on the surface directly. Curve editing allows the user to modify the overall shape of the mapped texture by adjusting the input curve. Figure 4.10 shows the curve editing interface. The texture deformation tool allows the user to specify a set of control points on the mapped texture on the surface, and deform the mapped texture by dragging any of the control points. As

Figure 4.11: Texture deformation. (a) After texture painting on the surface, the user moves points on the texture object to deform the texture object; (b) The control points and the moving vector are mapped to image space; (c) Deformation is applied in image space; (d) Deformed texture object is mapped back to surface space.

showed in Figure 4.11(a), user set control points on the texture object in the surface space and then move those control points to control the deformation. The underlying process is illustrated in Figure 4.11(b)(c). Based on the local mapping between the 3D surface space and the texture space, the control points and control operations are mapped to the dual image space. Then real-time deformation [50] is performed in the dual image space. Finally, the deformed texture object is mapped back to the surface(see Figure 4.11). The other editing tool allows the user to modify the mapping curve interactively. By dragging a point on the user-input curve, the shape of the curve could be edited. The algorithm is illustrated in Figure 4.12.

4.4 Experimental Results

We implemented our system in C++ and tested it on a workstation with a 2.67GHz Xeon CPU, 12 GB of RAM and an NVIDIA GeForce GTX295 GPU. We used a multi-

Figure 4.12: The curve editing algorithm. (a) The user drags a point in the texture brush curve which is presented by the blue strip; the effective weights of the points are distributed as a gaussian distribution, where the red color refers to 1, the blue color refers to 0, and the gray color refers to inactive; (b) The editing result; the position of each point is moved by the dragging vector multiplied by the effective weight.

touch display as the input device. In our test, the major operations, including texture painting, texture cloning, texture map editing and texture animation design typically took less than 0.5 seconds.

Figure 4.13: Comparison of parameterization in highly curved region. (a) Extended exponential map. (b) Exponential map.

The extended exponential map led to lower distortion in highly curved regions than the conventional exponential map [97]. As demonstrated in Figure 4.13, we specified the extended exponential map with one stroke while the exponential map was specified by the center point, radius and the direction. Considering that our interface requires the

	Exponential map	Extended exponential map
Local parameterization	Yes	Yes
Local parameterization along the sketch	Not straightforward to implement	Straightforward to implement
Distortion in high curvature area	High (Acceptable)	Low (Acceptable)
Distortion on sharp geometry	High (Unacceptable)	Low (Acceptable)

Table 4.1: Comparison of extended exponential map and exponential map.

same amount of user interaction as a decal system, if not less, our interface allows more freedom to control the mapping by sketching the central axis of the texture and leads to lower distortion. From the interaction point of view, the decal interface [97] using the exponential map, supports only the control of the center point and radius of the decal. Thus, only the copy-and-paste operation is allowed. More details on the comparison of extended exponential map and exponential map are demonstrated in Table 4.4.

4.5 Expert Evaluation

To understand how artists feel about texture brush and how it compares with the conventional texturing experience with existing digital tools, we invited two artists to use texture brush and comment on it.

To avoid any biasing effect by us, we only provided our system and a set of base models. A simple function explanation, which took around 10 minutes, was also provided. Then, the artists were asked to use our tool for free design. More than 10 artworks were created by them. It took 5 – 60 minutes to finish a textured model. Figure 4.16 show some results of the artists. Among them, the Fertility model is fully textured. The artist used both color painting tools and texture painting tools to achieve the result. Figure 4.14 shows the animation snapshots which was designed by our texture animation design tool.

The comments from the artists show that our system has an intuitive texturing interface and a smooth work flow. “I like the idea of texture brush. I can focus on the general shape of the texture object, without considering the pixel-level details.” “I even forgot the object is a texture image. It is a semantic object to me.” “I am glad that I can control the shape of the texture object by sketching.” Artists also comment on the work flow. “I paint on the model with colors first. Then I use texture brush to add interesting details. If I find the texture mapping result is not as desired, I use the texture map editing tool to modify it. However, it is not a fixed process; I also use the color painting tool to refine the result. The more I become familiar with texture brush, the more I paint in arbitrary order.” “I like the experience of using this tool. I can complete all the operations, such as painting, texture mapping and map editing, in a single view.”

The artists were asked in what situation they would use texture brush and what its limitations were. “I would like to have it in painting tasks. I have a large number of texture resources. With texture brush, they could be vividly used in new paintings in the form of various shapes. ” “ The provided interactive tools are powerful in 3D painting. It would be interesting to paint an object first, such as an osier, then you could clone the osier and have a group of osiers of different shapes.” “I would like to call it a painting tool rather than a texture mapping tool. The advantage is that all the operations are with painting style. The interface is intuitive. I can get familiar with it in a few minutes. One disadvantage is that it might be difficult to set up a covering texture mapping, where the texture covers a large region in the model. In this situation, I would like to use existing tools, such as Zbrush.”

4.6 Summary

We have presented texture brush, an interactive tool for surface texturing. We proposed an extended exponential map to construct high quality local parameterization. Compared

Figure 4.14: Texture animation: the 2D flower textures are moving along the user-specified paths.

to the conventional exponential map, our method has low distortion and naturally supports the anisotropic texture mapping. We proposed a set of intuitive texturing tools, such as texture painting, texture cloning, texture animation and texture map editing. Our interface integrates texture mapping and editing in a seamless manner and enables a smoother and more natural work flow in which the user can focus on the design task itself without switching back and forth among different tools or stages. The encouraging experimental results and positive evaluation by artists demonstrate the efficacy of our Texture Brush for interactive texture mapping.

Limitation. First, each user-specified stroke has to be simple, i.e., without self-intersection. Thus, our parameterization method does not allow the self-intersected texture, although multiple textures can have overlap. It could be alleviated by cutting the stroke into non-self-intersecting smaller segments. However, we should process carefully about the u v values for the triangles along the cut boundary belonging to two pieces. Second, the proposed texture brush is based on the anisotropic local parameterization, which allows the user to map the texture *locally*. It would be difficult to map a texture to a large region or even the entire model.

Figure 4.15: Bunny textured with I3D'13 logo.

Figure 4.16: More texturing results.

Chapter 5

Multitouch based Building Modelling

5.1 Motivation

Building design is an event per excellence, involving complex considerations and requiring designers to explore a vast problem space. Such complexity tends to be beyond our perceptual capabilities, demanding multiple trial and error, and the decomposition of the problems through formal derivations. This is a rather time-consuming process and the exploration is usually limited by time and budget.

Sketching and massing are two primary design techniques widely used in the early stages of building design [34, 2]: *sketching* is a common design method used for generating ideas and exploring options with pen and paper, while *massing* is a primary design process normally used for understanding the overall shape and size of a building in relation to its surrounding environment with approximate volumetric information. During the early stages of design, architects explore multiple scenarios to clarify their design ideas and to arrive at a coherent building solution. Sketching and massing are not merely a presentation media, but are also effective approaches that aid designer's engagement with the spatial cognition process [106]. Hence, the functionality and the quality of design media can greatly affect the development of the design.

In the early design stages, a fast feedback loop between design actions and visualization helps architects to develop their design, and facilitate communication among design

Figure 5.1: Buildings with progressively-varying and twisted structures: Burj Al Arab in Dubai, Taipei 101 in Taipei, Turning Torso in Sweden, and The Shard in London (left to right). Images are from www.flickr.com.

teams and clients. Although there are many off-the-shelf commercial software for 3D modeling, e.g., Maya, 3Ds Max, AutoCAD, etc., most of them are exceedingly powerful and comprehensive, and are targeted mainly at general 3D modeling and highly-experienced/well-trained users, e.g., artists, engineers, designers, etc. Conversely, fast prototyping tools such as Google SketchUp are easy-to-learn and intuitive-to-use for both experts and novice users, allowing a quick mock-up of 3D prototypic structures. However, existing prototyping tools still focus mostly on general 3D shapes.

This work is a result of a collaborative work with a domain expert who is an architect. We develop a novel, multi-touch interface design for supporting the early stages of building design, aiming to combine aspects of sketching and massing to aid the design process. In particular, we focus on the design of tall buildings due to the scale and complexity involved in such design. Moreover, we can observe the trend in tall buildings to be more expressive in its silhouette due to the level of maturity we have reached using modern construction techniques (see Figure 5.1). We propose a family of multi-touch gestures, where one can start by sketching and editing the 2D contour of the base floor, and then

extend and edit it to create a 3D building model with fairly complex geometric expression in a matter of minutes (see Figure 5.2). The simple drag-and-drop texturing can further enrich the model’s appearance. In addition to 3D modeling, our interface computes and provides assorted architectural parameters for the building design, e.g., its dimensions, facade area, and volume, thereby providing useful hints for guiding the design process.

Compared to the conventional keyboard-mouse interface, our interface incorporates hand-oriented actions such as sketching, slicing, curving, and twisting, which allow one to quickly define the geometry and manipulate its volume (massing). Moreover, we also develop a number of novel interface tools for fast geometric modeling (such as symmetry snapping and easy texturing of image and facade geometry) on the building model, and various editing operations. Thus, our interface can contribute to the conceptual phase of contemporary building design, where building designers and architects can apply it to do quick mockups, try alternative ideas, and get fast feedback to support major design decisions. Moreover, our proposed interface can serve as a rapid 3D modeling tool for creating content in digital media and games, where we need buildings with good-looking shapes and appearance rather than a high level of detail. Such rapid 3D building prototyping is not easily achievable with conventional multi-touch 3D modeling interfaces, existing 3D sketching tools, and general 3D modeling software.

To the best of our knowledge, this is the first work we aware of to explore and develop multi-touch interactions for rapid creation and editing of tall buildings with non-trivial geometry. Here we summarize its technical contributions:

- We propose a novel multi-touch design interface, facilitating rapid creation of non-trivial contemporary buildings through stage-by-stage editing of floor, structure, and appearance. Our interface is not limited to modeling 3D buildings; it can be adopted to other types of 3D models with repeated geometric features, rotational/planar symmetry, or curved profiling.

(a) Taipei 101 (183s) (b) Burj Al Arab (67s) (c) Turning torso (86s) (d) Burj Khalifa (273s)

Figure 5.2: An architect constructed prototypic models of famous buildings by using our multi-touch interface. The time taken to create these 3D building prototypes is shown below each model.

- We propose an effective symmetry snapping method, which allows us to keep the local geometric symmetry when editing the 2D floor layout with multi-touch. It is especially useful with imprecise finger interaction, and can be applied for editing 2D patterns and 3D models with local/global symmetry.
- As suggested by experienced architects, we propose a set of novel multi-touch gestures; some are particularly useful for massing, e.g., multi-touch twist for progressively-varying floor structures, crop for quick editing and testing of building designs (to meet the sunlight regulations prohibiting casting shadows over neighboring buildings), etc.

5.2 Overview

This section introduces our multi-touch user interface, and then describes the usage scenario when applying the interface to design 3D buildings.

User Interface. Our multi-touch interface consists of a main working window and a floating-tabbed panel. These are shown on the left and right sides of Figure 5.3(b), respectively. The main window has three buttons on the left, allowing the user to undo/redo a sequence of operations, enable/disable control point snapping when editing a 2D floor

plan, and bring out different designs for comparison. Note that the current GUI layout is mainly for right-handed users, but we may re-arrange the floating tabbed panel to the left and adapt the interface for left-handed users.

The main working window shows a perspective view of a 3D building model that is being designed and edited. See again Figure 5.3(b). In this window, the user can perform basic multi-touch manipulation to control the camera view as well as apply various multi-touch gestures to edit and decorate the design. The floating-tabbed panel consists of a number of windows: 1) a floor plan editor (shown in Figure 5.3(b)) for one to view and edit the 2D contour of the currently selected floor, 2) a building profile editor for editing the profile curve, and 3) resource libraries for one to retrieve and apply pre-defined floor plans, walls, window images, and geometric facades by simple drag-and-drop.

Usage Scenario. Basically, there are three major stages to quickly mock up a 3D building design, as illustrated in Figure 5.4(a)-(i):

- (i) **Initial Design.** First, we begin by constructing the 2D contour of the ground level floor. The user can either pick an existing floor plan from the related resource

Figure 5.3: (a) Our multi-touch display system; (b) Our GUI interface consists of a main working window (left) that presents the building design in 3D and a floating tabbed panel (right) that offers a 2D floor plan editor, a building profile editor, and various resource libraries.

Figure 5.4: System overview. Here we show our design pipeline with the *Turning Torso* building. The user starts by creating a 2D floor plan, which can either be selected from the tabbed panel (see (a)) or sketched by the user. After that, the user can freely edit the 2D floor contour using the 2D floor plan editor by manipulating the control points on the 2D contour. Our system provides multi-touch gestures and a novel grid&symmetry snapping mechanism for users to edit the 2D floor contour (see (b)). The inset for the two-finger anchor gesture specifies the symmetry axis. Later, the user employs one finger to select the floor plan and create the first ground-level floor (see (c)). After that, the user can create multiple floors by duplicating the ground floor (see (d)). Then, the user can apply an image texture to the selected floors by a simple drag-and-drop (see (e)). Our interface also allows the user to easily copy and paste selected floors to extend the building levels (see (f)), and zoom out using a two-finger pinch (see (g)). Finally, the user can select the whole building and use a one-finger horizontal drag to twist the building structure (see (h)) to produce the final result (see (i)).

library (see (a)) and/or sketch&edit his/her own contour using the 2D floor plan editor (see (b)). Then, the user drops the floor plan into the main working window, creates a single floor (see (c)), adjusts its height, and then duplicates it to create many other floors and form an initial design (see (d)).

- (ii) **Structure Editing.** After that, we enter the structure editing stage, where the user edits the building structure by various multi-touch gestures. Here, the user applies various multi-touch gestures (e.g., copying (see (f)), scaling (see (g)), twisting (see (h&i)), and slicing, etc) to edit the entire building or a range of

selected floors. Various building parameters including the facade area and volume are recomputed and presented interactively to guide the design.

- (iii) **Appearance Enhancement.** Lastly, the user decorates a building design by dragging and dropping image textures or geometric facades (see (e)) from the library resources.

Note that we group the interactive operations into these three stages to facilitate our understanding of design strategy, but in practice, one can perform them in any order as long as the operation being performed is appropriate to the current condition of the building design. For example, we can drag-and-drop image texture (see Figure 5.4(e)) onto the initial ground floor before the structure editing stage. As shown in Figure 5.4, a user can rapidly create a 3D building design with non-trivial geometry in a matter of minutes. However, such a task could be very tedious and time consuming, even when performed by experienced users with general 3D modeling software.

Besides the above major stages, the user can also record multiple building designs and compare them side by side visually in the main working window with their key architectural parameters. Moreover, the user can load a surrounding environment, put the current building design into it, and examine its relation to the surroundings. These features, as suggested also by an architect, allow one to clarify the design and to arrive at a more coherent building solution.

5.3 Design Considerations

Our system is implemented on a 3M M2256PW (22") multi-touch display system (see Figure 5.3(a)). We tilted the display to a near horizontal orientation, which allowed the users' elbows to rest on the display border during the interaction. In the followings,

we present the interface design considerations that guide and direct our user interaction design:

1) Sufficient modeling capability. The interface should provide sufficient modeling capability for one to design contemporary buildings, so that we can create different forms of structures and shapes that can be found in various existing buildings, see Figure 5.1, e.g.,

- Curved profile, see *Burj Al Arab*, Dubai;
- Disrupted floor plans, see *Taipei 101*, Taiwan;
- Twisted structure, see *Turning Torso*, Sweden;
- Symmetric floor plans, see *Burj Al Arab*, Dubai;
- Slanted side, see *The Shard*, London.

Regarding this, we surveyed a wide range of contemporary buildings and examined their shapes and structures before formulating the interaction design.

2) Fast architectural sketching and massing. The interface should support the rapid design of tall buildings. Regarding this, we identified three major stages needed in the overall building prototyping process: *initial building*, *structure editing*, and *appearance enhancement*, as well as assorted modeling and editing operations. The interface also allows the user to compare all the designs side by side and put a design into the target surrounding environment, so that the user can easily identify both the merits and defects, and make further improvements if necessary. Besides the editing capability, the interface also shows the key building parameters, such as dimensions, facade area, and volume, to help guide the design.

3) Easy-to-learn gestures. As a fast prototyping interface, the gestures should also be easy to learn, so that users can quickly understand the interface and perform

the necessary operations to rapidly develop and evolve their designs. Regarding this, we organize relevant interactions by their types and operational stages to facilitate a consistent sequence of interaction. As users usually prefer a small number of gestures, it is highly desirable to re-use the gestures (especially the standard ones) in different scenarios without ambiguity. Furthermore, the most frequently used gestures should be designed to be simple to perform.

4) Taking advantage of direct manipulation. Structure editing is usually the most critical and challenging part of the whole design process. To enable the users to focus on this task, we provide them with a direct-manipulation interface, where all structure editing operations, e.g., viewpoint navigation, floor selecting, and facade editing, can be done with multi-touch gestures. By this, the users can fully focus their attention on the main working window.

5.4 Multi-touch Gestures

This section presents our interaction design and the related implementation. Our design includes four sets of multi-touch gestures organized as follows:

5.4.1 Initial Building

In the process of prototyping a building, our approach is to first create an initial 3D building design and then further refine it by editing its 3D structure (next stage: *structure editing*) and enhancing its appearance (another stage: *appearance enhancement*). To sketch an initial design, we basically need to create a floor plan and then extrude it to form an initial building. In summary, we propose the following operations relevant to this stage:

i) Sketching and editing a floor plan. In our system, a floor plan is represented by a 2D floor plan contour, or simply a floor contour, which is a closed curve made up

Figure 5.5: Multi-touch gestures for editing 2D floor plan contour. (a) Two fingers to specify the axis for symmetry snapping; while we hold the fingers on screen, we can still move them to adjust the symmetry axis (see the inset); (b) We can freely move a control point with a finger and it could snap to a mirror location of a point on the other side of the symmetry axis; see the inset for making a symmetric shape; (c) To add a control point, we can double-tap on a line/curve segment. The inset shows the result; (d) To remove a control point, we can drag it off the drawing area.

of polylines and spline curve segments (we use interpolatory splines). Hence, we can create non-trivial shapes for floor plans, see Figure 5.4. In detail, the user can pick a built-in floor plan from the library resource, or sketch a closed curve as a floor contour by free-style sketching using the floor plan editor. With the editor, the user can directly use fingers to manipulate the control points on the floor contour to adjust the contour shape, see Figure 5.4(b). Furthermore, double-tapping on a line segment can create a new control point for refining the contour shape, see Figure 5.5(c); double-tapping on an existing control point can toggle its status between straight and curved; while dragging a control point off the drawing area can remove it, see Figure 5.5(d). Lastly, the user can also store an edited 2D floor plan in the library resource as a future asset.

ii) Grid snapping. To improve precision in editing the floor contours, our interface provides a grid snapping function that allows the user to precisely snap and put a control point on grid points. This function is enabled by default but can be toggled by the snapping button on the left hand side of the interface, see Figure 5.3(b).

iii) Symmetry snapping. Like most everyday objects, contemporary buildings usually contain strong symmetric structures. To improve the editing efficiency, we allow users to

preserve the symmetry (local or non-local) of floor plans during the multi-touch interaction by our novel multi-touch technique: *symmetry snapping*. The user can start with a *two-finger anchor gesture* on the display - holding two fingers on the floor plan editor for two seconds to create a straight line, see the inset in Figure 5.5(a). Once a symmetrical line is formed, the user can fine-tune its position (possibly snapping its control points to the grid also) by moving the two fingers on the screen. After that, symmetry snapping is enabled with the constructed line serving as the symmetry axis:

- When the user manipulates a control point, he/she can move it close to the mirror image position of points on the other side of the axis. After releasing his/her finger, our interface snaps the control point to a mirror image position.
- Furthermore, once two points are snapped together, the two points are said to be binded, and can be moved together to maintain the local symmetry, see Figure 5.5(b). If one wants to break this symmetry, one just needs to use another finger to hold one of the two points and move the other. This is a novel and efficient technique for creating and editing symmetric patterns in multi-touch manipulation - a process that usually involves imprecise finger movements.

Lastly, to ensure consistency in successive symmetry snapping, our interface remembers all symmetry axes created so far per floor plan. Thus, if a new axis is found to be close to a previous axis, our interface will snap it to the previous axis. Finally, dragging the symmetry axis out of the main working window can take it out and disable symmetry snapping.

iv) Initial building from a floor plan. Once an edited floor plan is ready in the floor plan editor or we find a desired floor plan in the library resource, we can drag and drop the floor plan to the main working window with a single finger. After that, we can use one finger to touch and select a floor plan, and then use another finger to drag up

to control the height of this initial ground floor. The floor height will be used for every single floor in the entire building design unless the user rescales some floor heights later. After we have created the first floor, we can touch the resulting floor to select it and then apply the copy gesture (to be described later) to duplicate this floor to form all other floors above it, see detail in the next subsection. Finally, an initial building can be readily modified by the structure editing operations in the next stage.

5.4.2 Structure Editing

Our interface provides a family of multi-touch gestures, allowing one to quickly edit and mock up a design that mimics a contemporary building structure, see previous section. In short, one can first select a range of floors (or a particular floor) to be edited, and then perform the relevant gestures to modify the structure/geometry/appearance of the selected range of floors. Typically, the user of this approach will first employ the fingers

Figure 5.6: Multi-touch gestures for structure editing: (a) copying floors; (b) removing floors; (c) scaling floors (heights or contours); (d) editing the building profile for the entire building; (e) editing the building profile locally on selected floors; and (f) twisting.

of one hand to make the selection, and will then use the fingers of the other hand to do the editing. If no floors are selected, the editing applies to the entire building.

i) Floor selector. Whenever the user puts a finger on the building, a floor selector is created for the associated floor being touched, and is presented as a semi-transparent yellow thin-plate tagged with a floor number, see Figure 5.4 (b,c&d). Now, the user can use his/her finger to drag the selector up/down to choose a particular floor. To select a range of floors, he/she can use two fingers to touch two different floor levels. In this way, a pair of floor selectors can be created simultaneously with all floors in-between (inclusively) selected, see Figure 5.4(e). Once a floor selector is created and shown, we can still touch it and drag it to a different floor level on the building. To de-select the floor(s), the user can simply tap on a screen region outside the building. If there are no selected floors, the entire building is considered to be selected and the user edits will be applied to the entire building. Using this selection mechanism, the user can hold down the floor selection, allowing us to disambiguate the reuse of drag and pinch for object manipulation in the same window.

ii) Copy floors. To facilitate the modeling of tall buildings, we offer a simple gesture to copy blocks of floors. After selecting a single floor(or a range of floors) and holding down the selection, one can apply a one-finger vertical drag up to copy the selected floor(s) above the selected floors. The number of copies is proportional to the vertical distance between the initial and end touch points. Since the 3D building design can change interactively with the finger drag, the user can observe the visual changes in the prototyping design and adjust the number of copies interactively from the visual feedback, see Figure 5.4 (b&c) for copying a single floor and Figure 5.4(f) and Figure 5.6 for copying a range of floors.

iii) Removing floors. As well as creating more floors, one can also remove the selected floors by a one-finger vertical drag-down gesture, which is a reversed version of copying floors, see Figure 5.6(b).

iv) Rescaling floor height. Since rescaling floor height is a rather infrequent action, we assign a more complicated gesture for it: a two-finger vertical pinch open gesture. By repeating this two-finger pinch open gesture, one can continue to increase the height of the selected floor(s); reversing it (after a slight pinch open) with a pinch close can decrease the floor height.

v) Scaling floor plan. In addition, one can uniformly scale the contours of a range of floors simultaneously to quickly try out different shapes on the building design. This can be done by a simple two-finger horizontal pinch on the screen after selecting one or more floor(s), see Figure 5.6(c). Note that to allow a more comfortable execution of this gesture and prevent possible strain to the wrist, we do not require the two-finger gesture to be performed horizontally to the display. As long as the pinch angle is 45° degrees or below on the display, we regard it as a horizontal pinch.

vi) Twisting. Modern structural engineering techniques allow us to have more complicated and creative building profiles that can actually be constructed. One of these is the twisted structure, see again the Turning-Torso building in Figure 1.2. To create a design that includes twisting, one can first select a range of floors (at least two floors), and then use a one-finger horizontal drag to control the amount of twisting, see Figure 5.6(f).

In our implementation, we first compute the total twisting angle relative to the finger drag displacement. Then, we divide it by $n - 1$ to compute the twisting angle between each pair of successive floors, where n is the total number of selected floors. Like the copy gesture, one can again interactively perform this gesture to observe the results while moving one's finger(s). Lastly, since buildings normally cannot be twisted to a large angle due to structural constraints, we employ six degrees as the upper limit of the twisting angle between successive floors.

vii) Editing profile curve. Another interesting feature of contemporary buildings is the curved profile shape, such as the Burj Al Arab Hotel (see again Figure 1.2). We

can use the building profile editor in the tabbed panel to edit a profile curve to control the amount of scaling on the selected range of floors, see Figure 5.6(d)-(e). As shown in the building profile editor view, such a profile curve is constrained, and must start from the top and end at the bottom, corresponding to the top most and bottom most floors currently selected. The user can interactively manipulate the control points in ways similar to those used in the floor plan editor, and can observe the visual changes on the building design as his/her fingers move.

viii) Cropping the facade. Lastly, our interface also offers a handy editing tool for producing a slanted side, as in the modern building *The Shard*. As illustrated in Figure 5.7, the user can tap four fingers on any region outside the region in the main window to enable cropping. Then he/she can sketch a freeform curve on the building silhouette and cut out a portion of the building. Note that apart from artistic concerns, a number of real building designs have to be modified by such a cropping to allow sufficient sunlight to reach surrounding buildings in accordance with some building construction regulations. This interface feature allows us to quickly modify the building and observe its relation to the surroundings.

5.4.3 Appearance Enhancement

With the aforementioned gestures, the user can quickly make a building design with non-trivial geometry. However, the design lacks surface details, such as its facade. Our

Figure 5.7: Cropping the facade.

interface provides libraries of image textures and geometric facades for the user to quickly drag and drop to apply to the building.

The more complicated operation here is in fact the underlying implementation behind the user interaction. To map an image texture onto the building, our interface first computes the u and v texture coordinates for each vertex, where the v coordinate is simply interpolated along the height of a floor, whereas the u parameter can be found by an arc-length parameterization of each floor contour. Regarding a geometric facade, which is in fact a 3D geometric texture (e.g., windows), our interface can compute a volumetric space with the w texture coordinate in addition to u and v , where w is defined along the local normal direction on the building surface. In this way, we can appropriately map and attach a 3D geometry texture by deforming each geometric facade and fit them one by one onto the building surface like a 3D image.

5.4.4 Camera View

Camera view controls offer basic but essential operations for users to see their designs from different views. However, instead of allowing unrestricted control as in general 3D modeling software, we constrain the camera view to center only on locations along the building's central vertical axis from the ground to its tip. As a sketching and massing interface, this control is sufficient and brings greater efficiency since our view always focuses on the design without taking off.

Figure 5.8: Gestures for camera view controls.

In summary, our interface has the following camera-view gestures, see also Figure 5.8:

- First, we can rotate using a one-finger drag (left): the horizontal component of the drag rotates the building about its central vertical axis while the vertical component of the drag updates the camera's elevation angle, whose range is restricted from 0 (i.e., the camera cannot go below the ground) to 90 degrees (i.e., the camera cannot tilt beyond the building top);
- Second, a two-finger drag (middle) can move the view up and down by translating the view center (look-at) along the central vertical axis of the building;
- Last, users can also zoom in/out using a standard two-finger pinch (right), or double-tap to quickly zoom out (reset) to see the entire building design.

5.4.5 Design Coherency

In the building design process, one may have come up with various designs with (slightly) different variations. With our interface, the user can touch any screen region outside the building with three fingers. Then all existing designs can be displayed for comparison. Similar to the camera view control for a single building, the user can explore *all* buildings simultaneously. To delete a model, the user can simply drag it out. Furthermore, the user can touch the Sort button to re-order all the buildings according to a selected architectural parameter such as the facade area. After identifying the model that requires further editing, the user can double-tap it and put it into the normal editing mode. This comparison feature is desirable to allow the users to clarify their design ideas and to arrive at a coherent building solution, see Figure 5.9.

5.5 Implementation and Results

Our multi-touch building prototyping interface was built on a Dell workstation T3500 with a 2.67GHz Xeon CPU, 12GB memory, and a GeForce GTX295 (1792MB) graphics

Figure 5.9: Side-by-side comparison of designs is desirable for users to clarify their design ideas and to arrive at a coherent building solution. To delete a design, one can simply drag-and-drop it (a); the view can be updated (b).

card, as well as a 3M M2256PW (22") multi-touch display system, see Figure 5.3(a). The software is written in C++ and Qt 4.7.

With our multi-touch interface, users can quickly design 2D floor plans with non-trivial shapes, as well as 3D buildings with non-trivial appearance by twisting, profiling, cropping, and local non-uniform scaling. Figure 5.2 shows a collection of building models that mimic famous contemporary buildings around the world. All these models can be constructed in a matter of minutes by a 3D modeling artist, who only needs some pictures of each building, see Figure 5.2.

5.6 Expert review

We invited three architects to evaluate our multi-touch interface: two with 10+ years experience and one with three years experience. All were very familiar with multi-touch

Figure 5.10: Some results from the architects who participated in the expert review. All models were made in a few minutes.

devices, such as the iPad, iPhone, etc. The purpose of this expert review was to evaluate our interface from the architects' perspective, e.g., could our system meet their requirements? Was our interface intuitive and easy-to-use? Did the architects enjoy using our interface and did they want to use it in their case studies?

The architects first watched a 5-minute video tutorial that demonstrated the interaction gestures for 3D navigation, initial modeling, structure editing, appearance enhancement, and design comparison. The tutorial also contained step-by-step instructions for constructing the Turning Torso building. Next, they were provided with a gesture table, and were given 10 minutes to practice all the gestures. Then, each of them had 30 minutes to freely make building designs with our tool. All the participants were impressed by our multi-touch interface, and made various non-trivial designs, some of which are shown in Figure 5.10. Lastly, each participant was asked to complete a questionnaire and rate our interface on a scale from 1 to 5, see results in Figure 5.11.

Some gestures in our interface are also common in other multi-touch applications, e.g., the two-finger pinch for camera zoom in/out, and the one finger drag to change view (see Figure 5.8(a) and (c)). To reduce the user's learning curve, we also extend the usage of these gestures to other scenarios. For example, during the structure editing, when a range of floors is selected, a two-finger pinch vertically (resp. horizontally) scales the floor plan (resp. height), see Figure 5.6(c)).

Figure 5.11: Questionnaire in the expert review.

The architects also commented that our gesture set was easy to use. We observed that after the 15-minute training period (watching the video tutorial and practicing for 10 minutes), all the architects had been able to learn the gestures well. Furthermore, the gestures were also easy to recall, as evidenced by the fact that the architects seldom needed to look at the gesture table during the building design stage. Our system also recorded the frequency of each gesture when the architects performed them during the design stage. Statistics show that the most frequently used gestures were camera rotation (26%), floor selection (25%), editing 2D floor plan (17%), and applying the texture (12%). The least frequently used gestures were floor duplication (1%), floor scaling (3%), and initial building from the ground floor plan (3%).

All the participants agreed that our interface was easy to use, and was efficient for designing buildings. They were also satisfied with the results produced by our interface. In addition, they were very pleased with the capability of our interface to allow the entire building to be made just with the fingers, without any other form of input. They also appreciated that all the frequently used editing and navigation operations could be done completely by gestures, and that there were very few buttons for undoing, sorting and snapping. They could use drag-and-drop texturing to flexibly apply texture to a range of floors or to the entire building, which was very effective for improving the appearance. Such a texturing procedure is often tedious with conventional 3D modeling

tools. Moreover, our multi-hand gestures, such as twisting, scaling, and duplication, were very powerful for 3D editing, and they had not found their equal in any other multi-touch applications. However, they also pointed out several limitations of our current prototype system. The interface was limited to designing a *single* tall building, thus, it could not be used for the Petronas Twin Towers in Kuala Lumpur or a group of buildings. Moreover, the interface could not support the design of *freeform* structures, such as the Milan Trade Fair Complex.

5.7 User study

To quantitatively evaluate the performance of our interface, we compared our interface with Google Sketchup, a popular general-purpose 3D prototyping tool. Furthermore, we installed a multi-touch plug-in¹ on Google Sketchup, so that users could perform simple multi-touch gestures, such as pinching. We recruited twenty participants, 10 males and 10 females aged from 18 to 26. All were undergraduate or postgraduate students in a university architecture department.

We designed two experiments to compare our interface with multi-touch Google Sketchup. The first experiment was to create a 10-storey building with a concave floor contour and curved/twisted profile (see Figure 5.12(a)). Each participant had to first perform the initial modeling gestures to create a 2D floor contour and then extrude the floor plan to make a 3D building. Next, the participant had to non-uniformly scale only the building profile for the top levels, edit the related profile curve, and then twist it. Finally, the participant had to attach image textures to improve the appearance. The second experiment was to further improve the appearance of a building (produced from the first experiment) by applying image textures to it (see Figure 5.12(b)). There were

¹Sketchup MultiTouch project by Sharma, Solyga, and Bintahir: <http://code.google.com/p/sketchupmultitouch/>

Figure 5.12: User study. We designed two experiments for buildings with non-trivial geometry to compare the efficiency of our multi-touch interface against Multi-touch Google Sketchup in terms of fast building prototyping.

three facade textures available to the participants, and we also provided a table that specified the texture to be applied to each floor.

At the beginning of the user study, we introduced our *multi-touch* tool and tutored each participant in the same way as we did in the expert review process. After being given the figures that show the buildings in experiments #1 and #2, each participant was asked to complete the buildings, and our interface recorded the time taken by each of them. After that, we introduced multi-touch Google Sketchup to the participants. As a general purpose 3D modeling tool, Google Sketchup has many buttons, thus posing extra difficulty for beginner users. Though the plug-in enables participants to use Google Sketchup with simple multi-touch gestures, modeling operations such as extruding a face can only be done by first tapping a button. To reduce the learning curve for the participants and thus allow a fairer comparison, we discussed which buttons were necessary for the participants in order to complete the experiments. To assist the participants to tap these buttons, we reset the screen resolution to enlarge the buttons. The participants were given 15 minutes to familiarize themselves with these functionalities. Finally, they were asked to repeat the experiments with multi-touch Google Sketchup.

Figure 5.13: User study results. The vertical axis shows the average time taken for the participants to complete each experiment. The black vertical line on each bar is the standard deviation. Time was measured in seconds.

As shown in Figure 5.13, the average times taken to complete the two experiments with our multi-touch interface were 33 and 30 seconds, respectively, whereas the times taken with multi-touch Google Sketchup were 163 and 146 seconds, respectively. To analyze the user study data, we conducted a paired t -test with the null hypothesis H_0 : the mean values of the task completion time using the two interfaces are equal. We used a significance level of 0.005, given H_0 is true. The resultant t values for experiments one and two were 27.6 and 26.2, respectively. Both values were larger than the critical value, i.e., 2.86, (with a degree of freedom $DOF=19$) obtained from the t -test table; hence, we could reject H_0 , and showed that our interface can help users to complete the tasks faster.

Multi-touch Google Sketchup is slower than our interface for the following reasons. First, although users can use basic multi-touch gestures in Google Sketchup, they have to frequently tap various buttons to perform the different modeling operations needed to complete the tasks; this is not only tedious but also distracting to the users. Second, our drag-and-drop texturing function allows the users to quickly apply textures to a selected range of floors (or the entire building). Although multi-touch Google Sketchup also allows

users to apply textures to selected faces (i.e., windows in a floor), such selection is done on polygon levels (rather than semantic levels on the buildings); an approach, which is typically tedious and prone to error.

5.8 Summary

This work proposes a novel multi-touch interface comprising a rich set of direct finger gestures to facilitate rapid prototyping of modern building structures. Using our interface, users can quickly mock up, in a matter of minutes, reasonably complex building structures that can be found in many renowned modern buildings around the world.

In our approach, we divide the prototyping process into three stages, and devise effective multi-touch gestures dedicated to relevant modeling and editing needs in different working stages. Among them, our work contributes particularly to a number of novel modeling and editing techniques, including symmetry snapping for constrained multi-touch 2D editing, interface design for efficient multi-touch modeling of buildings, as well as a collection of effective gestures for quick refinement of building designs, e.g., copy and collapse, non-uniform deformation, profile curve editing, and twisting. These interaction gestures are not only efficient and relatively easy to learn and use, but also have great potential for supporting more general 2D/3D modeling and editing in other application scenarios. Finally, we developed a working system, and also evaluated its effectiveness with a number of users. The expert review justified that our interface has the potential for use in the conceptual phase of contemporary building design.

Generalization. Existing multi-touch techniques to interact with 3D contents focus mainly on 3D navigation, 3D object manipulation, physical simulation, and virtual scene design. However, our work advances the multi-touch interaction technology with a comprehensive and novel set of 3D modeling and editing gestures. With this, we can enhance the capability of multi-touch for designing 2D shapes as well as 3D models with

Figure 5.14: Our interface can be used with other types of 3D design, where the 3D models have repeated geometric features, planar/rotational symmetry, or curved profiling.

non-trivial structures involving twisting, profiling, and local non-uniform scaling. It is worth noting that the proposed multi-touch interface is not limited to designing 3D buildings. In addition, it can be used for other types of geometric designs: 1) for 3D models with repeated geometric features, e.g., bookshelves, cabinets, etc., see Figure 5.14(a)(b), which shows a bookshelf created with our interface; we first sketched the cross-section of the bookshelf using the 2D floor plan editor, and then adjusted the number of levels by the floor select/extruding feature in our tool; and 2) for 3D models with progressively varying and symmetric cross-sections, e.g., vases, bottles, bowls, etc. See the star fruit 3D model shown in Figure 5.14(c)(d). It can be quickly made with our interface in just two minutes.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

With the booming development of novel input technologies, various touch-based electronic devices have reached mature stages of development in recent years. Many research works aim to bring new and immersive interaction experience to users by exploring the capabilities of novel interfaces for computer graphics applications such as 2D/3D painting and 3D modeling. In this thesis, we described three graphics applications that used multi-touch and sketch based interfaces to improve the user interaction in terms of efficiency and intuitiveness. We also explored some interaction principles by conducting user studies on the specified applications and tasks. The general design insights of how to design a multi-touch interface for computer graphics applications can be listed as follows:

- Intuitive. Gestures should be designed as natural as possible. Thus, users can understand the gestures better and learn the gestures faster.
- Easy-to-use. We should keep the gestures as simple as possible.
- Well organized. Since computer graphics applications such as modelling require a lot of operations, we should divide multiple gestures into well organized groups. This can decrease the study curve of the users.

Designing multi-touch based interface by following these design insights and principles can improve the interactions between users and computer graphics applications. The proposed applications included multi-touch-based interfaces for diffusion curves, 3D texturing and sketching and massing of tall buildings.

Painting with 2D Diffusion Curves. We presented a novel multi-touch based interface for efficient design of 2D vector graphics drawings with diffusion curves. In sharp contrast to the conventional interface that requires tedious working steps for constructing diffusion curves, our interface supports a family of new multi-touch gestures, enabling users to simultaneously sketch multiple curves, edit curve geometry, and interactively adjust the multi-parameter in curve colors at the spot. Using the working system we developed, we conducted a user study to show the practical value of this interface design with both professional artists and novice users. Results show that our interface not only brings new painting experience, but also provides a practical and effective tool for vector graphics design, typically useful for styles like silk painting, cartoon, art poster, and photo-realistic effects.

Surface painting with textures. We presented *texture brush*, an interactive tool for surface painting and texturing. We proposed an extended exponential map to construct high quality local parameterization. Compared to the conventional exponential map, our method has low distortion and naturally supports the anisotropic texture mapping. We proposed a set of intuitive texturing tools, such as texture painting, texture cloning, texture animation and texture map editing. Our interface integrates texture mapping and editing in a seamless manner and enables a smoother and more natural work flow in which the user can focus on the design task itself without switching back and forth among different tools or stages. The encouraging experimental results and positive evaluation by artists demonstrate the efficacy of our Texture Brush for interactive texture mapping.

Multitouch based building modelling. We proposed a novel multi-touch interface comprising of a rich set of direct finger gestures to facilitate rapid prototyping of

modern building structures. Using our interface, users can quickly mock up in the order of minutes reasonably-complex building structures that can be found in many renowned modern buildings in the world. In our approach, we divide the prototyping process into three stages, and devise effective multi-touch gestures dedicated for relevant modeling and editing needs in each working stage. Among them, our work contributes particularly to a number of novel modeling and editing techniques, including symmetry snapping for 2D editing, the select-and-edit approach for manipulating local structures, as well as a collection of effective gestures for modeling and editing modern buildings like copy and collapse, non-uniform deformation, profile curve editing, and twisting. These interaction gestures are not only intuitive, efficient, and relatively easy to learn and use, but also has potential for supporting 2D/3D modeling and editing in other application scenarios as well. In the end, we develop a working system and also experiment it with a number of users to evaluate its effectiveness. Compared to existing multi-touch techniques to interact with 3D contents, previous techniques focus their concerns mainly on 3D navigation, 3D object manipulation, physical simulation, and virtual scene design, our work advances the multi-touch interaction technology with a comprehensive set of novel gestures. By this, we can increase our capability to model and edit 2D shapes as well as 3D models with nontrivial structures like twisting, profiling, and local non-uniform scaling. The expert review justified that our interface has the potential for conceptual phase of contemporary building design. Lastly, we envision that our interface can also be further generalized for constructing a larger variety of conceptual designs, in particular, man-made objects formed by generic operations like sweeping and extrusion. We will investigate this direction in our future work.

6.2 Future Work

Painting with 2D Diffusion Curves. The following work will be painting with diffusion curves on 3D surfaces. Although a lot of research works have been done in rendering diffusion curves in 3D space, there is no work in painting diffusion curves in 3D model. We want to design a novel interface for real-time painting diffusion curves on 3D surfaces. Also, we plan to conduct a user study for performing quantitative comparison with standard interfaces by asking the participants to reproduce a given image using the interface and recording the number of different operations used, timing, etc. In addition, we plan to conduct a user study for the performance in generating complicated drawings by asking the users to reproduce a given image by using both traditional and multi-touch based interfaces.

Surface painting with textures. There are two aspects that we can dig deeper about this project. One is to solve the limitation and improve the final effects of our system. As mentioned in my thesis, it's not straightforward of texturing brush system to handle self-intersection strokes. Naive method, such as simply cutting a long stroke into several non-self-intersection pieces, leads to distortion or incoherent artefact to the final result. The other one is to design a multi-touch based texturing tiling system for 3D models. We plan to use both pen and fingers as inputs.

Architectural building. There are two following projects we will work on in the future. One is to design a more powerful interface for architectural building modelling. Current work is more like a prototyping system. There are some issues that we haven't solved, such as how to build a building with holes, how to combine multiple buildings into a whole building and so on. Another future work is to do research on pen based sketching system. The challenge is how to make a 2D sketching building into 3D models. To solve this challenge, we need both novel design of interface and brilliant design of algorithms.

References

- [1] Maneesh Agrawala, Andrew C. Beers, and Marc Levoy. 3D painting on scanned surfaces. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95, 1995.
- [2] Omer Akin and Hoda Moustapha. *Strategic use of representation in architectural massing*. Elsevier, 2003.
- [3] Pär-Anders Albinsson and Shumin Zhai. High precision touch screen interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 105–112. ACM, 2003.
- [4] Ronald E Anderson. Social impacts of computing: Codes of professional ethics. *Social Science Computer Review*, 10(4):453–469, 1992.
- [5] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *ACM UIST '08*, pages 151–160, 2008.
- [6] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. Everybodylovesketch: 3d sketching for a broader audience. In *ACM UIST '09*, pages 59–68, 2009.
- [7] Ravin Balakrishnan and Ken Hinckley. The role of kinesthetic reference frames in two-handed input performance. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, UIST '99, pages 171–178. ACM, 1999.
- [8] W.C. Barnert. A comparison of one-handed and two-handed direct and indirect computer interaction. Technical report, Technical report, Department of Computer Science, Tufts University, Medford, Mass., 11, 2005.

- [9] Olivier Bau, Ivan Poupyrev, Ali Israr, and Chris Harrison. TeslaTouch: Electrovi-
bration for touch surfaces. In *ACM UIST '10*, pages 283–292, 2010.
- [10] Patrick Baudisch, Edward Cutrell, Ken Hinckley, and Adam Eversole. Snap-and-
go: helping users align objects without the modality of traditional snapping. In
Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,
CHI '05, pages 301–310. ACM, 2005.
- [11] Edward W. Ishak Benko, Hrvoje and Steven Feiner. Cross-dimensional gestural
interaction techniques for hybrid immersive environments. In *Virtual Reality, 2005.*
Proceedings. VR 2005. IEEE, pages 209–216. IEEE, 2005.
- [12] Hrvoje Benko, Andrew D. Wilson, and Patrick Baudisch. Precise selection tech-
niques for multi-touch screens. In *ACM CHI '06*, pages 1263–1272, 2006.
- [13] Xiaojun Bi, Tovi Grossman, Justin Matejka, and George Fitzmaurice. Magic Desk:
Bringing multi-touch surfaces into desktop work. In *ACM CHI '11*, pages 2511–
2520, 2011.
- [14] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose.
Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th*
annual conference on Computer graphics and interactive techniques, SIGGRAPH
'93, pages 73–80. ACM, 1993.
- [15] Renaud Blanch, Yves Guiard, and Michel Beaudouin-Lafon. Semantic pointing:
improving target acquisition with control-display ratio adaptation. In *Proceedings*
of the SIGCHI Conference on Human Factors in Computing Systems, CHI '04,
pages 519–526. ACM, 2004.
- [16] John C. Bowers, Jonathan Leahey, and Rui Wang. A ray tracing approach to
diffusion curves. In *Proceedings of the Twenty-second Eurographics conference on*
Rendering, EGSR'11, pages 1345–1352. Eurographics Association, 2011.
- [17] Colin G. Butler and Robert St. Amant. Habilisdraw dt: a bimanual tool-based di-
rect manipulation drawing environment. In *CHI '04 Extended Abstracts on Human*
Factors in Computing Systems, CHI EA '04, pages 1301–1304. ACM, 2004.

- [18] Bill Buxton, Ralph Hill, and Peter Rowley. Issues and techniques in touch-sensitive tablet input. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '85, pages 215–224. ACM, 1985.
- [19] Bill Buxton and Brad A. Myers. A study in two-handed input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '86, pages 321–326. ACM, 1986.
- [20] Samuel Hsiao-Heng Chang, Lachlan Stuart, Beryl Plimmer, and Burkhard Wünsche. Origami simulator: A multi-touch experience. In *ACM CHI Extended Abstracts*, pages 3889–3894, 2009.
- [21] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proceedings of the sixth annual symposium on Computational geometry*, SCG '90, pages 360–369, 1990.
- [22] Patrick Chiu, Chunyuan Liao, and Francine Chen. Multi-touch document folding: Gesture models, fold directions and symmetries. In *ACM CHI '11*, pages 1591–1600, 2011.
- [23] Aurélie Cohé, Fabrice Décle, and Martin Hachet. tBox: A 3D transformation widget designed for touch-screens. In *ACM CHI '11*, pages 3005–3008, 2011.
- [24] Lawrence D. Cutler, Bernd Fröhlich, and Pat Hanrahan. Two-handed direct manipulation on the responsive workbench. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, I3D '97, pages 107–114. ACM, 1997.
- [25] David (grue) DeBry, Jonathan Gibbs, Devorah DeLeon Petty, and Nate Robins. Painting and rendering textures on unparameterized models. *ACM Trans. Graph.*, 21(3):763–768, 2002.
- [26] Son Do-Lenh, Frédéric Kaplan, Akshit Sharma, and Pierre Dillenbourg. Multi-finger interactions with papers on augmented tabletops. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*, TEI '09, pages 267–274. ACM, 2009.

- [27] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95*, pages 173–182, 1995.
- [28] Leah Findlater, Jacob O. Wobbrock, and Daniel Wigdor. Typing on flat glass: Examining ten-finger expert typing patterns on touch surfaces. In *ACM CHI '11*, pages 2453–2462, 2011.
- [29] Clifton Forlines and Chia Shen. DTLens: multi-user tabletop spatial data exploration. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 119–122. ACM, 2005.
- [30] Clifton Forlines, Daniel Wigdor, Chia Shen, and Ravin Balakrishnan. Direct-touch vs. mouse input for tabletop displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 647–656. ACM, 2007.
- [31] Mathias Frisch, Sebastian Kleinau, Ricardo Langner, and Raimund Dachsel. Grids & Guides: Multi-touch layout and alignment tools. In *ACM CHI '11*, pages 1615–1618, 2011.
- [32] Chi-Wing Fu, Wooi-Boon Goh, and Junxiang Allen Ng. Multi-touch techniques for exploring large-scale 3D astrophysical simulations. In *ACM CHI '10*, pages 2213–2222, 2010.
- [33] Chi-Wing Fu, Jiazhi Xia, and Ying He. Layerpaint: a multi-layer interactive 3d painting interface. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 811–820, 2010.
- [34] Christian Gänschert. *Tools for Ideas: An Introduction to Architectural Design*. Birkhäuser GmbH, 2007.
- [35] Ismael Garcia, Jiazhi Xia, Ying He, Shi-Qing Xin, and Gustavo Patow. Interactive applications for sketch-based editable polycube-map. *IEEE Transactions on Visualization and Computer Graphics*, 19:preprints, 2013.

- [36] Yotam I. Gingold, Philip L. Davidson, Jefferson Y. Han, and Denis Zorin. A direct texture placement and editing interface. *UIST '06*, pages 23–32, 2006.
- [37] Gridlogics. [urlhttp://www.patentinsightpro.com/](http://www.patentinsightpro.com/), 2010. Technology Insight Reports. Multi-Touch Technology.
- [38] Tovi Grossman, Daniel Wigdor, and Ravin Balakrishnan. Multi-finger gestural interaction with 3D volumetric displays. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, *UIST '04*, pages 61–70. ACM, 2004.
- [39] Y. Guiard. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Journal of motor behavior*, 19:486–517, 1987.
- [40] Y. Guiard and T. Ferrand. Asymmetry in bimanual skills. *Manual asymmetries in motor performance*, pages 175–195, 1996.
- [41] Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, *UIST '05*, pages 115–118. ACM, 2005.
- [42] Jefferson Y. Han. Multi-touch interaction wall. In *SIGGRAPH 2006 Emerging technologies*, page 25, 2006.
- [43] Mark Hancock, Sheelagh Carpendale, and Andy Cockburn. Shallow-depth 3D interaction: Design and evaluation of one-, two- and three-touch techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, *CHI '07*, pages 1147–1156. ACM, 2007.
- [44] Mark Hancock, Thomas ten Cate, and Sheelagh Carpendale. Sticky tools: full 6DOF force-based interaction for multi-touch tables. In *ITS '09*, pages 133–140, 2009.
- [45] Pat Hanrahan and Paul Haeberli. Direct wysiwyg painting and texturing on 3d shapes. *SIGGRAPH Comput. Graph.*, 24(4):215–223, 1990.

- [46] Ken Hinckley, Randy Pausch, Dennis Proffitt, and Neal F. Kassell. Two-handed virtual manipulation. *ACM Trans. Comput.-Hum. Interact.*, 5(3):260–302, September 1998.
- [47] Ken Hinckley, Randy Pausch, Dennis Proffitt, James Patten, and Neal Kassell. Cooperative bimanual action. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, CHI '97, pages 27–34. ACM, 1997.
- [48] Qi-Xing Huang, Bart Adams, Martin Wicke, and Leonidas J. Guibas. Non-rigid registration under isometric deformations. In *SGP '08*, pages 1449–1457, 2008.
- [49] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM TOG*, 24(3):1134–1141, 2005.
- [50] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78:1–78:8, 2011.
- [51] Stefan Jeschke, David Cline, and Peter Wonka. A gpu laplacian solver for diffusion curves and poisson image editing. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 116:1–116:8. ACM, 2009.
- [52] Stefan Jeschke, David Cline, and Peter Wonka. Rendering surface details with diffusion curves. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 117:1–117:8. ACM, 2009.
- [53] Paul Kabbash, William Buxton, and Abigail Sellen. Two-handed input in a compound task. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pages 417–423. ACM, 1994.
- [54] J Karat, J E McDonald, and M Anderson. A comparison of menu selection techniques: touch panel, mouse and keyboard. *Int. J. Man-Mach. Stud.*, 25(1):73–88, July 1986.
- [55] Rubaiat Habib Kazi, Kien Chuan Chua, Shengdong Zhao, Richard Davis, and Kok-Lim Low. SandCanvas: A multi-touch art medium inspired by sand animation. In *ACM CHI '11*, pages 1283–1292, 2011.

- [56] Rubaiat Habib Kazi, Takeo Igarashi, Shengdong Zhao, and Richard Davis. 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*, CHI '12, pages 1727–1736, 2012.
- [57] Ji-Sun Kim, Denis Gracanin, Kresimir Matkovic, and Francis Quek. iPhone/iPod touch as input devices for navigation in immersive virtual environments. In *IEEE VR '09*, pages 261–262, 2009.
- [58] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. In *Proc. Natl. Acad. Sci. USA*, pages 8431–8435, 1998.
- [59] Kenrick Kin, Maneesh Agrawala, and Tony DeRose. Determining the benefits of direct-touch, bimanual, and multifinger input on a multitouch workstation. In *Proceedings of Graphics Interface 2009*, GI '09, pages 119–124, 2009.
- [60] Kenrick Kin, Bjorn Hartmann, and Maneesh Agrawala. Two-handed marking menus for multitouch devices. In *ACM CHI'12*, pages 33–40, 2012.
- [61] Kenrick Kin, Tom Miller, Björn Bollensdorff, Tony DeRose, Björn Hartmann, and Maneesh Agrawala. Eden: A professional multitouch tool for constructing virtual organic environments. In *ACM CHI '11*, pages 1343–1352, 2011.
- [62] David S. Kirk, Shahram Izadi, Otmar Hiliges, Stuart Taylor, Abigail Sellen, and Richard Banks. At home with surface computing? In *ACM CHI'12*, pages 159–168, 2012.
- [63] Jan J. Koenderink and AJ van Doorn. The internal representation of solid shape with respect to vision. *Biological cybernetics*, 32(4):211–216, 1979.
- [64] Vladislav Kraevoy, Alla Sheffer, and Craig Gotsman. Matchmaker: constructing constrained texture maps. *ACM Trans. Graph.*, 22(3):326–333, 2003.
- [65] Haeyoung Lee, Yiying Tong, and Mathieu Desbrun. Geodesics-based one-to-one parameterization of 3d triangle meshes. *IEEE MultiMedia*, 12:27–33, 2005.

REFERENCES

- [66] SK Lee, William Buxton, and K. C. Smith. A multi-touch three dimensional touch-sensitive tablet. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '85, pages 21–25. ACM, 1985.
- [67] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Texture sprites: texture elements splatted on surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 163–170, 2005.
- [68] G. Julian Lepinski, Tovi Grossman, and George Fitzmaurice. The design and evaluation of multitouch marking menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2233–2242. ACM, 2010.
- [69] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21(3):362–371, 2002.
- [70] Steven Longay, Adam Runions, Frédéric Boudon, and Przemyslaw Prusinkiewicz. TreeSketch: interactive procedural modeling of trees on a tablet. In *SBIM '12*, pages 107–120, 2012.
- [71] Hao Lü and Yang Li. Gesture coder: A tool for programming multi-touch gestures by demonstration. In *ACM CHI'12*, pages 2875–2884, 2012.
- [72] Shahzad Malik, Abhishek Ranjan, and Ravin Balakrishnan. Interacting with large displays from a distance with vision-tracked multi-finger gestural input. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 43–52. ACM, 2005.
- [73] J. Matejka, T. Grossman, J. Lo, and G. Fitzmaurice. The design and evaluation of multi-finger mouse emulation techniques. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1073–1082. ACM, 2009.
- [74] Nobuyuki Matsushita and Jun Rekimoto. HoloWall: Designing a finger, hand, body, and object sensitive wall. In *ACM UIST '97*, pages 209–210, 1997.
- [75] MERL-DiamondTouch. [urlhttp://www.merl.com/projects/DiamondTouch](http://www.merl.com/projects/DiamondTouch).

- [76] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, 1987.
- [77] Jon Moeller and Andruid Kerne. ZeroTouch: an optical multi-touch and free-air interaction architecture. In *ACM CHI'12*, pages 2165–2174, 2012.
- [78] Tomer Moscovich and John F. Hughes. Multi-finger cursor techniques. In *GI '06*, pages 1–7, 2006.
- [79] Tomer Moscovich and John F. Hughes. Indirect mappings of multi-touch input using one and two hands. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1275–1284. ACM, 2008.
- [80] Alex Olwal and Steven Feiner. Rubbing the fisheye: Precise touch-screen interaction with gestures and fisheye views. In *In Conf. Supp. of UIST'03*, pages 83–84, 2003.
- [81] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 92:1–92:8. ACM, 2008.
- [82] Daniel Ostroff and Ben Shneiderman. Selection devices for users of an electronic encyclopedia: an empirical comparison of four possibilities. *Inf. Process. Manage.*, 24(6):665–680, November 1988.
- [83] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch, and Przemysław Prusinkiewicz. Self-organizing tree models for image synthesis. *ACM Trans. Graph.*, 28(3):58:1–58:10, 2009.
- [84] Wai-Man Pang, Jing Qin, Michael Cohen, Pheng-Ann Heng, and Chol Kup-Sze. Fast rendering of diffusion curves with triangles. *Computer Graphics and Applications, IEEE*, 32(4):68–78, 2012.

- [85] James Patten, Hiroshi Ishii, Jim Hines, and Gian Pangaro. Sensetable: a wireless object tracking platform for tangible user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '01, pages 253–260. ACM, 2001.
- [86] Hans Køhling Pedersen. A framework for interactive texturing on curved surfaces. SIGGRAPH '96, pages 295–302, 1996.
- [87] Gabriel Peyré and Laurent D. Cohen. Geodesic remeshing using front propagation. *Int. J. Comput. Vision*, 69(1):145–156, 2006.
- [88] Claudio Pinhanez, Rick Kjeldsen, Lijun Tang, Anthony Levas, Mark Podlaseck, Noi Sukaviriya, and Gopal Pingali. Creating touch-screens anywhere with interactive projected displays. In *Proceedings of the eleventh ACM international conference on Multimedia*, MULTIMEDIA '03, pages 460–461. ACM, 2003.
- [89] Andrei D. Polyanin and Valentin F. Zaitsev. *Handbook of exact solutions for ordinary differential equations*. Chapman & Hall/CRC, 2002.
- [90] R. L. Potter, L. J. Weldon, and B. Shneiderman. Improving the accuracy of touch screens: an experimental evaluation of three strategies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '88, pages 27–32. ACM, 1988.
- [91] Jason L. Reisman, Philip L. Davidson, and Jefferson Y. Han. A screen-space formulation for 2D and 3D direct manipulation. In *ACM UIST '09*, pages 69–78, 2009.
- [92] Jun Rekimoto. Smartskin: an infrastructure for freehand manipulation on interactive surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '02, pages 113–120. ACM, 2002.
- [93] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. “Grabcut”: interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.

- [94] P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. Multi-chart geometry images. In *SGP '03*, pages 146–155, 2003.
- [95] Munehiko Sato, Ivan Poupyrev, and Chris Harrison. Touché: enhancing touch interaction on humans, screens, liquids, and everyday objects. In *ACM CHI'12*, pages 483–492, 2012.
- [96] Ryan Schmidt. Stroke parameterization. *Comp. Graph. Forum*, 32(2):(to appear), 2013.
- [97] Ryan Schmidt, Cindy Grimm, and Brian Wyvill. Interactive decal compositing with discrete exponential maps. *ACM Trans. Graph.*, 25(3):605–613, July 2006.
- [98] Andrew Sears and Ben Shneiderman. High precision touchscreens: design strategies and comparisons with a mouse. *Int. J. Man-Mach. Stud.*, 34(4):593–613, April 1991.
- [99] Anirudh Sharma, Sriganesh Madhvanath, Ankit Shekhawat, and Mark Billinghurst. Mozart: a multimodal interface for conceptual 3d modeling. In *ICMI '11*, pages 307–310, 2011.
- [100] Qian Sun, Juncong Lin, Chi-Wing Fu, Sawako Kaijima, and Ying He. A multi-touch interface for fast architectural sketching and massing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page to appear. ACM, 2013.
- [101] Qian Sun, Long Zhang, Minqi Zhang, Xiang Ying, Shi-Qing Xin, Jiazhi Xia, and Ying He. Texture brush: an interactive surface texturing interface. In *Proceedings of the 2013 symposium on Interactive 3D graphics and games*, I3D '13, page to appear, 2013.
- [102] Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo. Diffusion curve textures for resolution independent texture mapping. *ACM Trans. Graph.*, 31(4):74:1–74:9, July 2012.
- [103] Microsoft Surface. [urlhttp://www.microsoft.com/surface/index.html](http://www.microsoft.com/surface/index.html).

REFERENCES

- [104] Kenshi Takayama, Ryan Schmidt, Karan Singh, Takeo Igarashi, Tamy Boubekeur, and Olga Sorkine. Geobrush: Interactive mesh geometry cloning. *Comput. Graph. Forum*, 30:613–622, 2011.
- [105] Kenshi Takayama, Olga Sorkine, Andrew Nealen, and Takeo Igarashi. Volumetric modeling with diffusion surfaces. In *ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA '10, pages 180:1–180:8. ACM, 2010.
- [106] Barbara Tversky. *The Cambridge Handbook of Visuospatial Thinking*, chapter Functional Significance of Visuospatial Representations. Cambridge University Press, 2005.
- [107] Yochay Tzur and Ayellet Tal. Flexistickers: photogrammetric texture mapping using casual images. *ACM Trans. Graph.*, 28(3):45:1–45:10, 2009.
- [108] Daniel Vogel and Gery Casiez. Hand occlusion on a multi-touch tabletop. In *ACM CHI '12*, pages 2307–2316, 2012.
- [109] Julie Wagner, Stéphane Huot, and Wendy Mackay. BiTouch and BiPad: designing bimanual interaction for hand-held tablets. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 2317–2326. ACM, 2012.
- [110] Daniel Wigdor, Hrvoje Benko, John Pella, Jarrod Lombardo, and Sarah Williams. Rock & Rails: Extending multi-touch interactions with shape gestures to enable precise spatial manipulations. In *ACM CHI '11*, pages 1581–1590, 2011.
- [111] Daniel Wigdor, Joe Fletcher, and Gerald Morrison. Designing user interfaces for multi-touch and gesture devices. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, pages 2755–2758. ACM, 2009.
- [112] Andrew D. Wilson. Playanywhere: a compact interactive tabletop projection-vision system. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 83–92. ACM, 2005.

- [113] Andrew D. Wilson, Shahram Izadi, Otmar Hilliges, Armando Garcia-Mendoza, and David Kirk. Bringing physics to the surface. In *ACM UIST '08*, pages 67–76, 2008.
- [114] Chunlin Wu and Xue-Cheng Tai. A level set formulation of geodesic curvature flow on simplicial surfaces. *IEEE Trans. Vis. Comput. Graph.*, 16(4):647–662, 2010.
- [115] Mike Wu and Ravin Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, UIST '03, pages 193–202. ACM, 2003.
- [116] Jiazhil Xia, Ismael Garcia, Ying He, Shi-Qing Xin, and Gustavo Patow. Editable polycube map for gpu-based subdivision surfaces. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 151–158, 2011.
- [117] Shi-Qing Xin, Ying He, and Chi-Wing Fu. Efficiently computing exact geodesic loops within finite steps. *IEEE Trans. Vis. Comput. Graph.*, 18(6):879–889, 2012.
- [118] Shi-Qing Xin and Guo-Jin Wang. Improving Chen and Han’s algorithm on the discrete geodesic problem. *ACM Trans. Graph.*, 28(4):1–8, 2009.
- [119] Shi-Qing Xin, Xiang Ying, and Ying He. Efficiently computing geodesic offsets on triangle meshes by the extended xin-wang algorithm. *Comput. Aided Des.*, 43(11):1468–1476, 2011.
- [120] Shi-Qing Xin, Xiang Ying, and Ying He. Constant-time all-pairs geodesic distance query on triangle meshes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 31–38, 2012.
- [121] Xiang Ying, Shi-Qing Xin, Qian Sun, and Ying He. Parallel and accurate Poisson disk sampling on arbitrary surfaces. In *SIGGRAPH Asia 2011 Sketches*, SA '11, pages 18:1–18:2, 2011.
- [122] Steve Zelinka and Michael Garland. Similarity-based surface modelling using geodesic fans. In *SGP '04*, pages 204–213, 2004.

REFERENCES

- [123] Hong Zhang, Xing-Dong Yang, Barrette Ens, Hai-Ning Liang, Pierre Boulanger, and Pourang Irani. See me, see you: a lightweight method for discriminating user touches on tabletop displays. In *ACM CHI'12*, pages 2327–2336, 2012.
- [124] Kun Zhou, Xi Wang, Yiyang Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Texturemontage. *ACM Trans. Graph.*, 24(3):1148–1155, 2005.
- [125] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3D: an interactive system for point-based surface editing. *ACM Trans. Graph.*, 21(3):322–329, 2002.

Publications

- **Qian Sun**, Juncong Lin, Chi-Wing Fu, Sawako Kaijima, and Ying He. A Multi-touch Interface for Fast Architectural Sketching and Massing. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'13)*. pp. 247-256, 2013.
- **Qian Sun**, Long Zhang, Minqi Zhang, Xiang Ying, Shi-Qing Xin, Jiazhi Xia and Ying He. Texture Brush: An Interactive Surface Texturing Interface. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D'13)*. pp. 153-160, 2013.
- **Qian Sun**, Chi-Wing Fu, and Ying He. An Interactive Multi-touch Sketching Interface for Diffusion Curves. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. pp. 1611-1614, 2011.
- Long Zhang, **Qian Sun** and Ying He. Splatting Lines: An Efficient Method for Illustrating 3D Surfaces and Volumes In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D'14)*. pp. 135-142, 2014.
- Xiang Ying, Shi-Qing Xin, **Qian Sun**, and Ying He. An Intrinsic Algorithm for Parallel Poisson Disk Sampling on Arbitrary Surfaces In *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, Vol. 19, No. 9, pp. 1425-1437, 2013.
- Juncong Lin, **Qian Sun**, Guilin Li, and Ying He. SnapBlocks: a snapping interface for assembling toy blocks with XBOX Kinect In *Multimedia Tools and Applications (MTA)*, accepted, September 2013.

REFERENCES

- Dao T.P. Quynh, Ying He, Xiaoming Chen, Jiazhi Xia, **Qian Sun**, and Steven C.H. Hoi. Modeling 3D Articulated Motions with Conformal Geometry Videos (CGVs). In *Proceedings of the 19th ACM international conference on Multimedia (MM '11)*. pp. 383-392.
- Xiang Ying, Shi-Qing Xin, **Qian Sun**, and Ying He. Parallel and Accurate Poisson Disk Sampling on Arbitrary Surfaces. In *SIGGRAPH Asia 2011 Sketches (SA '11)*. ACM, New York, NY, USA, Article 18.