Subdivision Models in a Freeform Sketching System

Wai Kit Addy Ngan

Advisor: Professor Adam Finkelstein Second Reader: Professor David Dobkin

Submitted in Partial Fulfillment of the Requirements of the Degree of Bachelor of Science in Engineering

To the Department of Computer Science of Princeton University

May 7, 2001

I pledge my honor that this thesis represents my own work in accordance with Princeton University regulations.

Dedicated to my loving mum, dad and brother.

Abstract

This thesis presents the usage of subdivision models in our 3D sketching system. Subdivision surfaces are extremely suitable for our system because of its support for arbitrary topology, its natural smoothing through refinement, its ability to model creases and corners, etc. We give a brief presentation of some important properties of subdivision surfaces, before describing the Freeform Sketch system. Freeform Sketch is a direct modeling system we develop that utilize gestural input for both modeling commands and geometric descriptions. It is part of an effort to greatly simplify the modeling process, by sacrificing some precise control on the surface. However, with a support of some basic primitives, in addition to editing tools including *oversketching*, *trimming* and *joining*, the class of shapes we are able to model exceed that of previous work, with higher quality. One main focus of the thesis is on the trimming operation, which is non-trivial for subdivision surface. We propose an algorithm that support approximate trimming through remeshing and fitting, while adding no extra details to the surface.

Acknowledgement

First, I would like to thank Professor Adam Finkelstein for being my thesis advisor and for all of his help and support over the year. Also I would like to thank Professor David Dobkin for being my second reader. I want to thank Lee Markosian for getting me excited about this project, giving me valuable guidance all the way along, and teaching me everything from computer graphics to coding skills. And thanks go to Lena Petrovic, my graduate student partner, for all her hard work and help. I would also like to take this chance to thank Professor Thomas Funkhouser, whom I worked with throughtout my junior year and the following summer, for all of his excellent ideas and advices, and for all the meetings we had that went over 6 o'clock in the evening.

Of course, this work would not be possible without the support of my best friends, Yee Wai, Irene and Kueh, for always coming to the graphics lab to give me company, to assure that I am not alone. Thanks to long-time graphics lab warrior Matt, for all the good time, good music, and good movie trailers we share in the lab. I want to thank my parents and brother for their love and support. Finally I would like to thank Hang Seng Bank (Hong Kong) for the generous scholarship over the past four years, without which I would not be able to enjoy the most fruitful and exciting education in Princeton. Thank you all!

List of Figures

1.1	Screen shot from 3ds max 4. [1]	2
1.2	Class of shapes modeled in (a) <i>Teddy</i> and (b)SKETCH	3
2.1	Subdivision curve. [19, Chapter 2]	6
2.2	Subdivision Surface.	7
2.3	Refinement of one triangle by splitting each edge	8
2.4	The neighborhood around a vertex v_r of valence n	9
2.5	Masks for Loop subdivision	10
2.6	Masks for creases and corners. [8]	11
2.7	Geri's game, depicting models with creases of variable sharpness (Pixar)	11
2.8	Different triangulations of the same shape give different results after subdivi- sion. [14]	12
2.9	Ripples on a surface generated by the Loop scheme near a vertex of large valence. [19]	13
3.1	Inflation in Teddy. [10]	15
3.2	Steps to build a mug with a handle	16

3.2	Steps to build a mug with a handle (2)	17
3.3	Pen and tablet input for our system.	19
3.4	Four class of gestures recognized	20
4.1	Oversketching curve	23
4.2	Primitive surfaces.	26
5.1	Result from [11]	31
5.2	Creating the hole	32
5.3	Remeshing	33
5.4	Edge operations include swap, split and collapse.	35
5.5	Edge swap.	36
5.6	Error angle.	36
5.7	Oversampling effect if no fitting is performed	38
5.8	Error due to reposition.	39
5.9	Improved approximation with fitting.	40
6.1	Models we made in Freeform Sketch (1)	43
6.2	Trimming of a surface.	44
6.3	A <i>face</i> built using trim and join on a tube.	44
7.1	Bumps after remeshing	46

Contents

Ac	Acknowledgement iv			
1	Intro	oductio	n	1
2	Bacl	kground	I	5
	2.1	Subdiv	ision	5
		2.1.1	Introduction	5
		2.1.2	Basic Concept	6
		2.1.3	Properties	7
		2.1.4	Creases and Corners	10
		2.1.5	Dependency on triangulations	12
3	Mod	leling by	y Sketching	14
	3.1	An Illu	strated Example	15
	3.2	Gestur	al Input	18
	3.3	Primiti	ve Shapes	20

CONTENTS

	3.4	Trim and Join	20
	3.5	Oversketching	20
	3.6	Dependency Hierarchy	21
4	Prin	nitives	22
	4.1	Curves	22
		4.1.1 Creation	22
		4.1.2 Oversketching	23
		4.1.3 Issues on resolution	24
	4.2	Surfaces	25
5	Trin	nming and Joining	28
	5.1	Introduction	28
	5.2	Related Works	30
		5.2.1 Trimming Subdivision Surfaces - Litke <i>et al.</i>	30
		5.2.2 Approximate Boolean operations - Kristjansson <i>et al.</i>	31
	5.3	Creating the hole	32
	5.4	Remeshing	33
		5.4.1 Repositioning	34
		5.4.2 Modify Connectivity	34
	5.5	Fitting	37
	5.6	Alternatives to fitting	41

viii

CC	CONTENTS	
	5.7 Joining	41
6	Results	42
7	Discussion and Conclusion	45
A	Mathematical Properties of Subdivision Surfaces	48

Chapter 1

Introduction

3D modeling has always been important in computer graphics and CAGD¹. Animation, rendering, physical simulation, mechanical and architectural designs are all heavily dependent on the ability of modeling shapes in 3D. Well-known representations for 3D models include polygonal meshes, patches(commonly NURBS² patches), quadrics, implicit surfaces, and other variations. While these techniques have been developed extensively in the past few decades and commonly adopted in different disciplines, they all have some notable shortcomings. For example, polygonal meshes have limited resolution and a large number of polygons are required to represent smooth surfaces; representations using patches require special handling along boundaries of the patches to enforce continuity; and local editing is hard for implicit surfaces. On the other hand, the comparably new techniques of subdivision surfaces, combining the strength of polygonal and patch representation, have started to get wide attention in the past few years and have become the newest feature in commercial packages. 3 However, previous work on subdivision surfaces has primarily focused on specifying subdivision rules and analysis of their properties, provided with a control mesh. For example, Hoppe et al. presented work on fitting subdivision surfaces to pre-existing sampled data points [8]. However, not much work has been done for building subdivision surfaces from *scratch*. In 3ds max 4, for example, subdivision is supported primarily as a finishing tool that provides

¹Computer Assisted Geometric Design

²Non-Uniform Rational B-spline

³Subdivision surfaces are supported in 3ds max 4 and Maya 3 Unlimited.

CHAPTER 1. INTRODUCTION

smoothing and details addition, rather than as a modeling tool. Indeed, the user is required to put considerations on the triangulations of the mesh to ensure subdivision can give a nice result. This thesis is focused on methods of building up subdivision models, in the context of our Freeform Sketch system, which is described in the next paragraph. More details on the properties of subdivision surfaces will be described in Chapter 2.



Figure 1.1: Screen shot from 3ds max 4. [1]

During the past academic year I have been working on a modeling system named Freeform Sketch, with Lee Markosian, Lena Petrovic and Adam Finkelstein. Freeform Sketch has a different goal from popular commercial packages like 3ds max, Maya, SoftImage, etc. These packages commonly focus on giving the user ability to precisely manipulate models with maximal flexibility. They often expose the controls of the underlying representations to the user and give a large variety of tools, often with numerous menus, toolboxes and dialog boxes. On the favorable side, a very wide range of operations are supported, however, the complicated user interface (See Figure 1.1) often trade off the ease of use. Users are also required to have substantial knowledge about the underlying representation of the 3D models in order to make

good use of the tools. Indeed, even for experts, it is quite a time-consuming and difficult task to create relatively simple 3D models (e.g.mug,table,*teddy bear*,etc.) in these packages. Also, a system that supports high accuracy and precise control is only desirable in some applications like CAGD. On the other hand, for the casual users, or for cases when stylish models are preferred to realistic ones, it is often desirable to be able to *sketch* models in an imprecise way. More importantly, the ability to do 3D sketching is useful for fast prototyping before using more sophisticated tools to build the final refined model. Some representative systems in this direction include SKETCH [18] and *Teddy* [10]. These two systems provide interfaces that are simple to learn and use by specializing the modeling process for certain kinds of shapes – roundish, teddy-bear-like objects in *Teddy*; and boxy, geometric objects in SKETCH (Figure 1.2). Using these systems it is indeed surprisingly easy to model relatively sophisticated shapes, however the classes of shapes for which it is possible (roundish or boxy, respectively) is restricted. Modeling many kinds of free-form shapes in either SKETCH or *Teddy* would be a challenge.



Figure 1.2: Class of shapes modeled in (a)Teddy and (b)SKETCH

Our system is aimed at a goal similar to *Teddy* and SKETCH but with the ability to draw a much larger class of objects. The idea is to retain much of the directness and simplicity of those systems, while leveraging more of the users' drawing ability. We provide a direct,

gestural interface for modeling. We start by drawing planar and non-planar curves in 3D using a technique like the one described by Cohen *et al.* [4]. These 3D curves can then be used to define a variety of primitives, including ruled surfaces, generalized cylinders, and extruded shapes (See Chapter 4). These shapes can then be further refined by *oversketching*. In some respects Freeform Sketch strongly resembles both SKETCH and *Teddy*, but by adapting and generalizing the methods of those systems, and incorporating a more sophisticated model representation based on subdivision surfaces, Freeform Sketch is targeted to produce significantly more complex, detailed and compelling models.

In our system all curves and surfaces are represented using subdivision techniques, appealing to its various advantages (See Chapter 2) which are especially suitable for our system. For example, subdivision naturally provides smoothing, which is important for sketching as we often desire a smooth surface that approximates but not interpolates the user input. Also, its support of arbitrary topology is handy for building sophisticated models. For example, a mug with a handle, which has genus one, is already a non-trivial object to be represented by patches or implicit surfaces, while no special handling is required using subdivision surfaces. Last but not least, its multi-resolution representation supports level-of-detail techniques naturally, which make our system usable even on low-end computer systems.

While I have been involved mainly in the implementation of the system during the past year, I have particular interest on the use of subdivision techniques in our system. This thesis is thus centered on how we build up and perform various operations on the subdivision curves/surfaces, including redefinition by oversketching, trimming and joining; and the advantages and challenges of using subdivision techniques. We will give a brief introduction to our sketching system in Chapter 3, before we delve into details of our geometric primitives(Chapter 4) and our trimming procedure(Chapter 5). We conclude with the results we obtained, and a discussion of potential future works.

Chapter 2

Background

As this thesis is focused on subdivision models, we give a brief qualitative discussion of subdivision in this chapter, while leaving some of the important mathematical properties to Appendix A. For more in-depth information on subdivision, see [13, 19, 17, 15].

2.1 Subdivision

2.1.1 Introduction

Recently, subdivision surfaces have gained wide attention because of its numerous advantages over other modeling techniques:

- Arbitrary Topology: A subdivision surface uses a polygonal mesh as the base domain, so it can naturally support surfaces with arbitrary topology. It is not necessary to use trim curves which is common for NURBS surfaces, and also avoids the cumbersome boundary constraints of joined patches.
- Natural Representation: 3D models are usually converted to a polygonal representation before rendering and graphics hardware are often optimized for rendering polygons. Subdivision surfaces are represented by polygonal meshes and require no conversion

before rendering.

- Multiresolution support: Subdivision surfaces are intrinsically associated with multiresolution representation. Multiresolution rendering/editing can be supported by storing detail vectors on subdivided levels.
- Scalability: Because of its multiresolution structure, it can readily support adaptive approximation depending on available system resources.

2.1.2 Basic Concept

In a brief way, subdivision curve or surface is defined by the limit of recursive refinement of an initial set of *control vertices*. Figure 2.1 shows the case for subdivision curve. On the left we have 4 control vertices, joined by straight line segments. Next to it is a refined version, with new vertices inserted on each line segments of the previous level, and all vertices are repositioned with respect to some rules. It is intuitive that if the rules are chosen *appropriately*, the refinement converges and its limit would be a *smooth* curve. A subdivision curve is defined to be this smooth limit curve. Indeed, B-spline curves with arbitrary degree can be shown to be just a special case of subdivision curves, away from the endpoints [2].¹



Figure 2.1: Subdivision curve. [19, Chapter 2]

Similarly, the same notion can be defined for surfaces (Figure 2.2). With a *control mesh* and a set of refinement rules, a subdivision surface is defined to be the limit of recursive refine-

¹For example, a uniform cubic B-spline can be constructed by using the subdivision masks (1,6,1) and (1,1) on even and odd vertices respectively.



Figure 2.2: Subdivision Surface.

ment on the control mesh. However, it is more involved than the case of curves, because the refinement rules are dependent on the connectivity of the vertices, while for curves the connectivity is always the same (i.e.all vertices have valence² two). Thus in general, closed-form expression of subdivision surfaces does not exist. As subdivision curves could be regarded as a subset of subdivision surfaces, we will focus only on surfaces in the remaining sections.

2.1.3 Properties

There are several different rules that operate on different kind of meshes, including Loop[13], Catmull-Clark[3], Butterfly[6], to name a few.³ In our system we employ the Loop scheme, including the rules for defining creases and corners described in [8] and [15], one of the simplest scheme defined on triangular meshes and the most commonly used.

Before going into the details of the Loop scheme, we should note some of the important properties among all the subdivision schemes.

• Local Support : A certain point on the surface is only affected by vertices in a finite neighborhood. (e.g.vertices in one-ring⁴ or two-ring neighborhood) This is desirable as it provides ability for local editing, and it also allows very efficient computation during

²Number of neighboring vertices.

³See [19, Chapter 4] for a more complete classification.

⁴By one-ring neighborhood of a vertex we mean all the vertices that share an edge with the vertex. A two-ring neighborhood is the union of one-ring neighborhoods of all the one-ring neighbor, and so on.



Figure 2.3: Refinement of one triangle by splitting each edge.

refinement.

• **Convergence and Smoothness** : To be able to define a subdivision surface, the scheme of course need to be converging on arbitrary control meshes. Moreover, it is desirable for it to converge quickly so that a small number of refinements can approximate the limit surface with small error bound. Also, the limit surface is piecewise smooth, away from predefined *creases*(See next section).

Now we start our discussion on the Loop scheme on the general setting, ignoring creases at the moment. The Loop scheme is defined on a triangular mesh and it is an *approximating* scheme as the limit surface does not interpolate the control mesh. All vertices with valence six are called *regular* vertices. The Loop subdivision surface is C^2 continuous away from extraordinary vertices, and C^1 continuous at the extraordinary vertices.

During one step of refinement, each triangle is split into four triangles by inserting three new vertices, by splitting each of the three edges. (Figure 2.3) The new vertices are called *odd* vertices while the ones that correspond to vertices from the previous level are called *even* vertices. It is easy to observe that odd vertices are always regular, and irregular vertices from the control mesh remain irregular(the degree stay unchanged) in all subdivision level. The positions of the vertices after the refinement are dependent on vertices in the one-ring neighborhood, and are given by the following formulas:

For an even vertex v^{r+1} at subdivision level r + 1 with valence n (see Figure 2.4),

$$v^{r+1} = (1 - n\beta)v^r + \beta v_1^r + \beta v_2^r + \ldots + \beta v_n^r$$



Figure 2.4: The neighborhood around a vertex v_r of valence n.

where $\beta = \frac{1}{n} (5/8 - (\frac{3}{8} + \frac{1}{4} \cos \frac{2\pi}{n})^2)$. For the odd vertices,

$$v_i^{r+1} = \frac{3v^r + 3v_i^r + v_{i-1}^r + v_{i+1}^r}{8}$$

The position of the refined vertices are indeed given by weighted averages of the neighboring vertices in the previous level, so the equations above can be nicely represented with *masks* diagrams (Figure 2.5) that describes the weights. It can be directly seen from these masks that the implementation for Loop refinement is extremely simple.

The Loop scheme is developed based on the *three-directional box spline*, which means in *regular* regions the Loop subdivision surface is exactly a spline patch. Thus in the regular region it can be evaluated exactly in the same way as a spline surface. However, because of the irregular vertices and the arbitrary topology of the surface, a closed-form expression of the surface does not exist in general. Nonetheless, the limit position and normal to the surface at locations corresponding to control vertices can be computed in an extremely simple way, as a result of eigenanalysis on the subdivision matrix (See Appendix A). Also, recent research has established methods for exact evaluation of the limit surface at arbitrary parameters. [17, 16]



Figure 2.5: Masks for Loop subdivision. $\beta = \frac{1}{n} (5/8 - (\frac{3}{8} + \frac{1}{4} \cos \frac{2\pi}{n})^2)$ where *n* is the valence of the vertex

For some of the important mathematical details of the Loop scheme see Appendix A. For even more details on subdivision in general, see [13, 19, 17, 15].

2.1.4 Creases and Corners

One of the advantages of using subdivision surface is its ease of modeling creases, which is a very difficult task for NURBS surfaces and implicit surfaces.

To model creases and corners, we simply tag edges and vertices as sharp, and specify special rules for these creases, as suggested in [8]. See Figure 2.6 showing these crease rules. With these rules, the creases actually converge to uniform cubic B-spline curves, and we can see from the masks that it decouples the surface on the two side of the crease. Thus the surface is C^0 continuous across creases.

DeRose *et al.* [5] further proposed variable sharpness for creases with a slight modification to the subdivision rules. Figure 2.7 shows the famous Geri from Pixar's short animation *Geri's Game*, who is modeled by subdivision surfaces with creases of variable sharpness. With the support of variable sharpness, subdivision surface is a simple yet very robust way for supporting a wide class of models.



Figure 2.6: Masks for creases and corners. [8]



Figure 2.7: Geri's game, depicting models with creases of variable sharpness (Pixar).



Figure 2.8: Different triangulations of the same shape give different results after subdivision. [14]

2.1.5 Dependency on triangulations

While analysis proves mathematical continuity for the limit surface, in practice this does not always correspond to the intuitive sense of smoothness, at least *visually*. See Figure 2.8. The first and the third figures are two control meshes that triangulate the same shapes. The second and the last figures are their limit surface respectively. We can see that the limit surface of the first control mesh have a lot of unexpected wiggles and bumps, while the latter is much smoother. While the first surface looks unacceptable in its resemblance to the original shape, mathematically it is still at least C^1 continuous everywhere.

Thus, as a general rule, we want to avoid skinny triangles like those in the first figure. Also, we want to avoid vertices with large valence. With the Loop scheme, vertices with large valence often lead to undesired creases and rippling effect(Figure 2.9).



Figure 2.9: Ripples on a surface generated by the Loop scheme near a vertex of large valence. [19]

Chapter 3

Modeling by Sketching

Commercial modeling systems commonly focus on getting as much functionality as possible, often at the expense of convoluted user-interface (Figure 1.1). However, if we were to support CAD-like accuracy, there does not seem to exist much room for a simpler interface. For example, users need to be able to see arrays of control vertices for surfaces if they were to manipulate them directly; and a huge interface for specifying parameters could not be avoided if the user were allowed to define shapes numerically.

However, if we are willing to sacrifice some accuracy, and restrict the class of models we support, then we claim we can greatly simplify our interface, at the same time abstract away many of the underlying model representations details from the users.

Works with similar goals include SKETCH and *Teddy*. These two systems provide interfaces that are simple to learn and use by specializing the modeling process for specific kinds of shapes. SKETCH supports primitives including polygons, rectangular boxes, cylinders, and it allows extrusion and Boolean operations. While the class of primitives is small, the Boolean operations and extrusions allow it to model shapes without general curved surface like furniture and other artificial objects very well. Nonetheless, it is not intended and impossible for modeling most biological shapes, or any shapes that cannot be represented well without freeform surfaces. *Teddy* addressed this problem by offering a system that specialize in teddy-bear-like shapes instead. Its main idea is to use a special kind of inflation (Figure 3.1). However, it becomes inflexible when the user desires otherwise. Also, the shapes it

CHAPTER 3. MODELING BY SKETCHING

can support is always topologically equivalent to a sphere(i.e. without holes), which make it challenging or impossible to model objects with higher complexity.



Figure 3.1: Inflation in Teddy. [10]

Our system is aimed at a class of models that expands on the union of the above two. We retain the simplicity of the interface, while leveraging more of the user's drawing ability. The direct gestural interface we support remove most of the GUI controls necessary in commercial systems, while giving an efficient way to define shapes. With trimming and joining, we are able to build up more sophisticated models with arbitrary topology. Also, we support creases with variable sharpness by subdivision surfaces.

Before going into any details of our modeling system, we begin by showing how a mug with reasonable quality could be built with our system in a short amount of time.

3.1 An Illustrated Example

The mug we are trying to model is composed of a handle and a main body. The handle can be made from a tube described by its axis, and the main body can be built from extruding a hollow tube.

Here is a step-by-step illustration for the process of building up this mug (Figure 3.2):

1. Draw the base of the mug. (a)(b)(c)



Figure 3.2: Steps to build a mug with a handle



(i) Dot and cross to attach point and get axis



(k)Handle attached



(j)Draw axis curve describing the handle



(l)Draw three strokes to smooth creases.



(o)Control mesh

(p)Mesh refined twice.

Figure 3.2: Steps to build a mug with a handle (2)

- 2. Draw the silhouette of a cylinder that describe the general shape of the main body of the mug. (d)(e)(f)
- 3. Remove the top face of the cylinder to make it hollow.(g)
- 4. Extrude the surface to give it thickness.(h)
- 5. Draw a curve attached to the surface that describe the axis of the handle.(i)(j)
- 6. Draw the cross section of the handle and have the handle merged with the body of the mug.(k)
- 7. Use a smoothing gesture to smooth the top of the mug.(l)(m)

The entire process takes about one minute, for an user with average experience with our system.

3.2 Gestural Input

We primarily use the pen and tablet as the input for our system, along with a few keystrokes that corresponds to command which no intuitive gestures could be associated (e.g. refine and unrefine). Modeling with the pen corresponds naturally to drawing on paper, while it is awk-ward and difficult to use the mouse for the same task. Also, with the additional flexibility of the pen, we can interpret more gestures than it is possible with the mouse.

There are five basic gestures we use in our system (Figure 3.4):

- Tap Make contact with the tablet at a location instantaneously. Similar to a click for the mouse. Used for selection and in compound gestures.
- Slash A fast, short and straight stroke. Used for several commands depending on context.
- Dot Circle around a spot several times. Used for creating a point in space.
- Cross Self-explanatory. Used for displaying the axis widget.



Figure 3.3: Pen and tablet input for our system.

• Stroke - Any gesture that does not classify to any of the above. Usually used for describing geometry.

While the *tap* and *stroke* are possible with the mouse, *slash*, *dot* and *cross* are very difficult for it. For more involved operations like smoothing and hole cutting, we employ compound gestures, which is a sequence of several simple gestures with a particular configuration. For example, the gesture for cutting a hole is to first draw the hole on the surface, then tap and slash inside the hole. Whenever the gesture is not recognized as a single command, we put the gesture in a gesture list, and keep the gesture on screen. Every time a new gesture is entered into the list, we interpret the entire list. If the configuration is valid and complete, we carry out the operation. We clear the list if the user is idle for some predefined amount of time, or the configuration fails to be contained in any possible interpretation.



Figure 3.4: Four class of gestures recognized

3.3 Primitive Shapes

Like SKETCH and unlike *Teddy*, we support primitive shapes. The primitives we support include curves, generalized cylinders, extrusions, ruled surfaces and panels¹.

3.4 Trim and Join

The set of primitives we have is obviously not enough for creating interesting shapes. We support trimming and joining in order to build up more complicated shapes. At this moment, however, we do not support Boolean operations between shapes. See Chapter 5 for a detailed description of the trimming and joining operations.

3.5 Oversketching

We support oversketching for shape editing. To redefine the shape of a curve, the users can simply overdraw on the existing curve, instead of manipulating control vertices directly as in most modeling system. More details on oversketching of curve will be discussed in the next chapter.

For primitive surfaces, in particular *tubes*, we also support silhouette oversketching. By silhouette we mean the boundary between the object and the background. For the tubes, as we have well-defined parameterization, we can reasonably infer user's intention when they overdraw the silhouette.

¹By panel we mean a surface that is enclosed by a boundary

CHAPTER 3. MODELING BY SKETCHING

Also, for surfaces like ruled surfaces and panels that are directly defined by its parent curves, the surface will be modified automatically by oversketching the associated curves.

Unfortunately, we are not able to support oversketching for general surface at this moment. This is due to two main reasons:

- For more complex shapes, in particular shapes that have self-occlusion, the silhouette can be complicated. When the user do an oversketch drawing, it is often ambiguous which set of silhouette edges he is trying to edit.
- A general surface lacks a natural parameterization likes a tube. While it is always possible to move the vertices on the silhouette to fit the oversketch visually, the result is not always satisfactory. More importantly, without a parameterization, it is hard to decide how to move vertices neighboring the silhouette accordingly to give a smooth falloff for the deformation.

3.6 Dependency Hierarchy

When an object is created based on another one, (for example a tube built from the base, which is a panel built from some boundary curves), we keep the old object and store it in a dependency tree. This is useful as we can later go up the dependency tree and modify shapes that were used to define the children. The changes to the parent will then be passed to the children in an appropriate way, if possible. For example, if we modify the shape of the base by oversketching, it will be carried forward to the tube to modify its cross section shape.

However, the inheritance is only preserved when possible. For operations like trimming which change the topology of the underlying mesh, the dependency is no longer obvious and hence is not supported.

Chapter 4

Primitives

To build models in our system, we always start from some *primitive* shapes. In particular, we usually start from curves, and then build surface primitives from them. More complex shapes can then be obtained by oversketching, trimming and joining.

4.1 Curves

Curves are represented using subdivision in our system, and contrary to commercial packages, we give the user the impression that a curve is a single entity defined by itself, instead of something controlled by a set of control vertices (e.g.NURBS curve). Thus we are able to create and edit the curves *directly*, without manipulating any control vertices.

4.1.1 Creation

During the creation stage, the user can use the pen to draw on any predefined plane. The trail of the screen space pixels will first be projected to the plane. The next step is to find a set of vertices to build the control mesh of the curve so that the limit curve approximate the drawing. To do this, we first try two vertices for the control curve. Then we iteratively double the number of control vertices until the *error* is smaller than ε . During each iteration, we

distribute the control vertices evenly by length on the drawing, and the error is measured by the average of the distance of each drawing sample point to the polyline defined by the control vertices. We typically do not want ε to be too small, otherwise we will pick up unwanted noise from user's drawing input. In our system we take ε to be the world space equivalent of 2 to 3 pixels.

When the iterations are finished, we obtain a polyline approximating the drawing, with each of the vertices interpolating. As a final step, we apply fitting to the control vertices, using the same algorithm described in Section 5.5, so that the limit curve instead of the control curve is approximating the drawing.

Apart from drawing curves on a predefined plane, we also support drawing curves in space with respect to a shadow. This technique was proposed by Cohen *et al.*in [4], and it prove to be a very powerful and easy way to specify curves in 3D.

4.1.2 Oversketching



Figure 4.1: Oversketching curve.

Using the Loop subdivision rule, a subdivision curve is indeed a B-spline curve. So we could possibly support editing in the standard approach, by directly manipulating the control vertices. However, as part of our goal is to achieve a direct interface, we hide the control points from the user, (unless the user explicitly unsubdivide to the base level), and we do not support direct manipulation of the control points. Instead, editing is achieved by *oversketching*. Oversketching is defined by some drawing that start and end on the curve, using the new drawing trail to replace the original segment between the starting point and the ending point

(Figure 4.1).

The algorithm Now assume the user draw a trail trying to oversketch the curve. Let the trail be $\{d_1, d_2, \ldots, d_n\}$, which is in screen coordinates. Project the vertices of the curve at the current subdivision level to screen space and denote the list of 2D points by $\{c_1, c_2, \ldots, c_m\}$. Find the closest points on $\{c_m\}$ to d_1 and d_n and name them c_j and c_k respectively. If the distances are bigger than some threshold, we reject the oversketch, as we cannot identify where the trail is joining the curve.

If the oversketch is not rejected, let j < k without loss of generality, and for this moment assume the original curve is open¹. Find l <= j and r >= k such that c_l and c_r are vertices that correspond to vertices on the control curve. Redefine the oversketch trail to be $\{e_n\} =$ $\{c_l, c_{l+1}, \ldots, c_j, d_1, d_2, \ldots, d_n, c_k, \ldots, c_r\}$. This trail will be used to redefine the control vertices in between and including the ones that correspond to c_l and c_r , denoting it by $\{q_n\}$. Then we parametrize both polylines $\{q_n\}$ and $\{e_n\}$ by length, and finally at each of q_i with parameter t_i , we reposition q_i to the polyline $\{e_n\}$ evaluated at t_i .

If the curve is closed, we have two possible candidates for redefinition provided two joining points, namely the two halves of the curve by cutting it at the two joining points. We resolve the ambiguity by computing the average distance between $\{d_n\}$ and the two halves and choose the closer one. The algorithm described above can then be applied to the chosen half.

4.1.3 Issues on resolution

It is obvious that the algorithm presented above will not be able to handle an oversketch that requires higher resolution than the original curve. In other words, by redefining k control points we only have k degrees of freedom and is not possible to capture components with high details. To overcome this problem in the case of a stand-alone curve, we calculate the error of approximation after redefining the control points. If the error exceeds some predefined threshold ε , we repeatedly subdivide and reapply the oversketching until the error is smaller

¹If this was not true, we can reverse the list $\{d_n\}$.

than ε . While we can store the the editing as multiresolution detail on the different subdivision levels, we choose to simply throw away all the coarser ones and treat the control vertices at the refined level as the new control curve. If the curve has children in the dependency hierarchy, however, we cannot switch the control level freely. What we can do on this case is dependent on the children it has, and is handled by the children curves or surfaces.

4.2 Surfaces

We have four types of primitive surfaces (Figure 4.2): Tube, Extrusion, Ruled Surface and Panel

- Tube a generalized cylinder, with user-defined cross section, and different scaling between cross sections. Cone is a special case of a tube.
- Extrude give thickness to an open surface. We make a copy of the surface by translating each vertices along its normal by some user-defined amount. The new copy is then stitched to the original.
- Ruled Surface defined by a pair of curves, or a curve and a point. Surface with gridlike parameterization.
- Panel surface defined by boundary curves. For non-planar boundaries, it produces a surface with minimal *energy*.

Of these four, the tube is the most flexible shape and can be used to build a lot of shapes. We have two main ways of building a tube:

- 1. Use a closed curve as the cross section shape, describe the silhoutte. A tube will then be built with cross sections of varying size defined by the silhouette. (Figure 4.2(b)(g))
- 2. Use a curve as the axis of the tube, describe a cross section. The tube will then have uniform cross sections along the entire axis. (Figure 4.2(e))





Figure 4.2: Primitive surfaces.

When a tube is built, we build a control mesh for a subdivision surface that describe the mesh. If the cross section curve already exists, as in case 1, then we simply make copies of its control curve to build up the mesh for the tube. Otherwise, we build up the cross section curve first, in the way described in Section 4.1. The spacing between the cross sections are chosen so that: (1)it captures the resolution of the axis curve, and (2)it produces triangles with good aspect ratios. (Figure 4.2(c))

For the extrusion, we assume the *depth* we extrude is small compared to the size of the object, and hence we do not insert any new vertices during an extrusion. For the ruled surface, we insert new isoparameter curves between the two curves that define the surface, so that the triangles are of good aspect ratios.

The panel is one of the other more interesting primitives. Given a list of boundary curves that form a closed loop, not necessarily planar, we can get some form of triangulations that produce a surface which interpolate the curves. However, a naive triangulation would not give a good result when subdivided, as discussed in Section 2.1.5. Hence, we perform a remeshing to produce triangles with good aspect ratio. The algorithm is a simplified version of the one described in Section 5.4, without any tracking. So during repositioning we can simply move the vertices to the centroid of its neighbor, and the connectivity modification is purely based on the aspect ratios of the triangles. Indeed, the surface we would get is approximating the surface with minimum energy that interpolates the given boundary, and this is often the desired surface.

Chapter 5

Trimming and Joining

5.1 Introduction

Trimming and joining are important primitive operations in modeling. In order to build up compound objects, we often need to join several parts together. For example, a handle to a mug, or limbs to a torso. In some cases, it may be acceptable to just put the parts together in the correct position, with the parts remaining separate entities and intersecting each other. A more elegant way is to use CSG¹ [7, Chapter 12], which represents the object using Boolean operations (e.g. union, intersection, subtraction) of sub-objects. However, it is often more convenient and sometimes essential to have one geometric entity that correspond to the actual compound object. With the parts remained as separate objects, we do not have a single well-defined model, so it is tricky to perform any geometric operations on the joining region. With CSG, the Boolean operations are often stored in a tree structure, and again no explicit representation of the final object is constructed. Thus we do not have a single representation of the final model either, and it is difficult to apply subsequent operations like editing (especially around the region of the join), collision detection, rendering, etc. Hence, we are often interested in explicitly computing the result of a trimming or/and joining and represent the result with a single surface. To achieve that, we will need to be able to trim a surface.

¹Constructive Solid Geometry

CHAPTER 5. TRIMMING AND JOINING

For modeling with NURBS patches, trimming is indeed notoriously difficult and remains a challenge for even high end systems. Usually trimming is done by computing the pre-image of the trim curve in the parameter space. Then the pre-image in the parameter space can be used to bound regions for which the patch is evaluated. As the control vertices of the patches remain unchanged, the trimming ensures that the trimmed surface is coincident with the original surface. However, computing the pre-image is often difficult if not impossible, and also the representation for the result is more complicated than the original, as extra trim curves are required to define the trimmed surface. It becomes difficult to handle the surface when more and more trim curves are added.

In our system, we support the *trim* and *join* operation on subdivision surfaces. The result surface is represented by a modified subdivision surface, with no extra information stored (e.g. trim curves, multiresolution details). During a *trim*, the user describes the hole by drawing, then the surface is remeshed² with the hole removed. The user can then join new primitives to the hole by drawing shapes attached to the curve. In some cases like attaching a tube to a surface, the trim and join can be done in one step.

Before going into the details of our algorithm, we will first discuss the desired properties of a good trimming method.

A good trimming is achieved if the following three conditions are satisfied:

- Condition 1 The trimmed boundary match the trimming curve given.
- Condition 2 The surface away from the trimming region should be unaffected (i.e. trimming is a localized operation)
- Condition 3 The surface around the hole should stay close to the original surface.

Condition 1 is quite obvious. Also, as none of the current work can guarantee exact coinicidence between the original and the trimmed subdivision surface, condition 2 is important to restrict the region affected locally. Finally, condition 3 restricts the deviations from the original surface in this region.

²The triangulation of the surface is modified to adapt to the shape of the hole

As subdivision surface is relatively new, the trimming operations are not studied well until this year, when two works related are published, which we will discuss in the next section. Both works are published this year and our work is done in parallel with them and take a different approach.

5.2 Related Works

5.2.1 Trimming Subdivision Surfaces - Litke *et al.*

Litke *et al.* proposed in [12] a trimming method for subdivision surface. Their method is attractive in the sense that the trim boundary it creates exactly match the trim curve provided. (Condition 1). They also satisfy condition 2 as the algorithm is local, and the error of approximation for the region around the hole is bounded by *arbitrary* ϵ , which satisfies (and in some sense stronger than) condition 3.

Here is an overview of their algorithm:

- 1. Find approximate pre-image of the trim curve for a correspondence between the control mesh and the trim curve.
- 2. Adaptively refine³ triangles in the neighborhood of the pre-image.
- 3. Remove triangles that contain the hole.
- 4. Add a strip of triangles to fill the gap between the desired hole and the *bigger* hole created in the previous step.
- 5. *Relax* the vertices on the desired hole to give triangles with better aspect ratios.
- 6. Optimize correspondence.⁴

³according to subdivision refinement rules.

⁴While the adaptive refinement give natural correspondence to domain of the original surface, the correspondence is poor because the connectivity has changed. Hence they require a step to optimize the correspondence by doing a local search

7. Approximate surface by applying quasi-interpolation stencils to produce detail vectors at several levels, depending on the user-desired ϵ .

This algorithm is especially suitable for CAD applications because of the exactness they provide. However, to achieve the exactness, they require to put multi-resolution details to the surface, and in most cases the details need to be added to several subdivision levels to give a *good*-looking surface.

For our sketching system, however, this algorithm is less than ideal. We do not require the kind of accuracy they give, and at the same time we would like to keep our representation simple during this stage of modeling. For example, the dependency hierarchy is tricky to implement with multi-resolution details. While we do not preclude using multi-resolution details in our system, we would like to keep them to the final stage of the modeling to add user-desired details, like embossing.

5.2.2 Approximate Boolean operations - Kristjansson et al.



Figure 5.1: Result from [11]

Kristjansson *et al.*'s work involved more general boolean operations like intersection, union and subtraction on subdivision surfaces (Figure 5.1). They propose an algorithm for finding approximate intersection curves between two subdivision surfaces. With the intersection



Figure 5.2: Creating the hole

curves represented in the parametric domain, they subdivide the triangles around the intersection curve adaptively to create the hole, similar to Litke *et al.*'s work. After merging the domains of the two surfaces, fitting is performed to produce multi-resolution details to approximate the original surface.

5.3 Creating the hole

In the following sections we present our algorithm. The input is a subdivision surface and a trim curve, and the output is a trimmed subdivision surface which satisfies all three conditions in Section 5.1. Specifically, in our sketching system, the trim curve is given by a screen space pixel trail (closed), and a projection direction. We rephrase the trimming problem as the following:

Given an original surface defined by a control mesh M_0 and a trim curve, find a new control mesh S_0 such that its subdivision limit surface S_{∞} approximates M_{∞} with the hole removed, and such that the three conditions set in Section 5.1 are satisfied.

The first step is to create the boundary on the mesh. Contrary to the works discussed above, we directly split the faces along the boundary (Figure 5.2(b)). While the splitting requires extra work, we do not need to perform stitching and relaxation as is necessary in [12], or the snapping and refinement in [11]. During the splitting, we maintain correspondence of any new vertices to its position in the original control mesh, represented by barycentric coor-



Figure 5.3: Remeshing

dinates, which will be used later when we remesh the region. After the splitting is finished, we discard the triangles inside the hole. (Figure 5.2(c))

5.4 Remeshing

After removing the inner triangles, we are left with a mesh that is different from the original mesh *locally*. Triangles around the hole in general have bad aspect ratios (Figure 5.2(c)), and this is bad for subdivision surfaces as discussed in Section 2.1.5. In order to fix this problem , we need to remesh the region. See Figure 5.3. The remeshing has two main goals: (1) to produce triangles with better aspect ratios, (2) to smoothly interpolate the difference in resolution between the hole and M_0 .

Our remeshing algorithm resembles the one in [14]. Initially, we put vertices in the tworing neighborhood of the hole in a *hot list*. Each of the vertex in the hot list h has a corresponding tracking position T_h in the original control mesh M_0 , which is initialized during the splitting in the previous step. Then we do the following:

repeat

Step 1 - Reposition vertices in hot listStep 2 - Modify connectivity for edges in hot list

until hot list empty

Intuitively, step 1 try to distribute the vertices evenly and provides smoothing, while step 2 try to form triangles with better aspect ratio.

5.4.1 Repositioning

The goal of the repositioning step is to distribute the vertices more evenly. In this step, each vertex in the hot list is visited. Let h be one of the vertex and its position be x_h . To distribute the vertices evenly, we try to make every vertex close to the centroid of their neighbor. We set h's target position to be $t_h = \alpha x_h + (1 - \alpha)c_h$, where c_h the centroid of its neighbor, and α is a constant, typically set to 0.3. However, we do not reposition h to t_h directly, as t_h does not lie on M_0 in general. Instead, we do a local search around T_h in M_0 , to find a closest point to t_h . We then reposition h to this closest point and update T_h . If the repositioning move h less than ϵ distance, we remove h from the hot list.

5.4.2 Modify Connectivity

In this step, the connectivity in the neighborhood of the hole is modified trying to achieve three goals:

- 1. Triangles with better aspect ratio.
- 2. Closer to the original surface.
- 3. Vertices with valence six.

While these three goals are not directly opposed to each other, in many cases they are competing. During this step, we look at each of the edge that connect hot vertices. We try



Figure 5.4: Edge operations include swap, split and collapse.

to apply one of the edge operation as shown in Figure 5.4, if certain conditions are satisfied. Also, if any edge operations are performed, we put the affected vertices in the hot list.

Each vertex has an associated target length, which is initialized to the average adjacent edge length at the beginning. We set the target length of an edge the average of those of its two vertices. The conditions for the split and collapse are as follows: We split an edge if it is longer than 1.5 times the target length, and collapse the edge if it is less than 0.5 times the target length.

To decide whether to swap or not it is slightly more involved. First we check if the swap is valid as discussed in [9]. If the swap is valid, we want to evaluate (1) the improvement in aspect ratios of the triangles involved $(\Delta\theta)$, (2) the improvement of the distance to the original surface. (Δd)

Referring to Figure 5.5, assume that we are considering to swap edge BC, and BC is adjacent to two triangles, ABC and BCD. Then we calculate $\Delta \theta$ by looking at the change of minimum angle within the two triangles if BC is swapped.

 $\Delta \theta = min(minangle(ABD), minangle(ADC)) - min(minangle(ABC), minangle(BCD))$



Figure 5.5: Edge swap.



Figure 5.6: Error angle.

As all vertices are constrained to be on M_0 during repositioning, we measure the distance from the surface at the mid-point of an edge, which in general does not lie on the surface. So, let E be BC's mid-point, and F be AD's midpoint, then

$$\Delta d = error_angle(E) - error_angle(F)$$

where $error_angle()$ computes an angle that represents the error from the original surface M_0 . Refer to Figure 5.6. To compute error angle of E, we find the closest point T on M_0 to E using a local search. Then we define the error angle to be the angle between BT and BE.

Finally, we take a weighted sum of the two quantities to estimate the *improvement*.

$$I = \alpha \Delta d + \Delta \theta$$

In our system we choose $\alpha = 2$ and we carry out the swap operation if I > 0.

5.5 Fitting

After the remeshing step, we obtain a control mesh, call it S_0 , that have *good* triangles around the region, and identical to M_0 away from the trim region. The reposition procedure also maintains that all vertices of S_0 are lying on M_0 .

However, this does not necessarily give a good approximation of the original surface. As discussed in Section 2.1.5, the limit surface is dependent on the topology of the mesh. Indeed, when the hole we are trimming have a higher resolution than M_0 , which is common, our remeshing will produce triangles of varying size that interpolates between the size of M_0 and that of the hole. Hence, when tracking on M_0 we will suffer from oversampling. See Figure 5.7. Here we cut a small hole from a surface that has a much coarser control mesh. We do the cutting and remeshing as described in the previous sections, tracking on the coarse control mesh. During the retriangulation, small triangles are created around the hole and many new vertices are inserted because of the short target edge length around the hole.

However, these new vertices, which are relatively more densely populated than other vertices, are tracking on a control mesh with much lower resolution, and this contributes to oversampling artifacts. We can see from Figure 5.7 that the limit surface of the trimmed mesh inappropriately interpolates the big triangles around the hole in the original control mesh.

There is another issue which is independent of resolution. For simplicity, we can look at a two dimensional case. In Figure 5.8, we have a polygon on the left, let us take it be M_0 , a control mesh that we track on. At the right, we repositioned the vertices while staying on the control mesh. We can easily see that the subdivision limit of the repositioned polygon is smaller than that of the original. Of course, extreme cases like this are prevented by the surface



Limit surface before trimming Limit surface after trimming Figure 5.7: Oversampling effect if no fitting is performed



Figure 5.8: Error due to reposition.

error measure during the remeshing, but they are not totally guaranteed against because the remeshing is optimizing on a set of potentially competing conditions.

To resolve the issues described above, ideally, we should control the limit surface of S_{∞} directly, and optimize by tracking vertices of S_{∞} on M_{∞} . However, tracking on a limit surface is hard, so in our algorithm we do the following instead:

- 1. Remesh with vertices tracking on M_0 .
- 2. For each vertices v_i in the trim region, calculate v'_i using M_{∞} evaluated at the parameter given by the tracking position.
- 3. Use a fitting procedure on the vertices $\{v_i\}$ so that their limit of subdivision tends to $\{v'_i\}$.

First we carry out the remeshing as before by tracking on M_0 . For each vertex v_i in the trim region, we can find the barycentric coordinates of the track position on the track face. This can be regarded as a parameter in the domain surface M_0 . With this parameter, we can evaluate the corresponding position in M_{∞} , possibly using techniques specified in [16]. However, for simplicity, we approximate the evaluation by subdividing the triangles a few times. Evaluating at each vertex v_i , we get a list of geometric positions $\{v'_i\}$ which we hope S_{∞} to interpolate. So, the final step is to fit the v_i s so that their limit positions are at the respective v'_i .



Figure 5.9: Improved approximation with fitting.

To solve the fitting problem, it is equivalent to finding the inverse of S^{∞} , where S is the global subdivision matrix. Rather than solving the inverse directly, (which is possible in practice as S^{∞} is sparse), we apply an iterative algorithm that works well in practice. Let $V = (v_1, v_2, \dots, v_i)^T$, $V' = (v'_1, v'_2, \dots, v'_i)^T$,

repeat

 $V = V + (V' - S^\infty V)$ until $|(V' - S^\infty V)| < \varepsilon$

In practice, during each iteration the error drops geometrically, and we require about 10-15 iterations with a setting of ε equals to 0.001% of the size of the mesh. See Figure 5.9 for a illustration of the improvement for cutting a small hole with the fitting procedure.

However, we should notice that our fitting procedures only optimize V, the set of vertices in the control mesh S_0 , so that their limit positions tend to a set of fixed positions V' on M_{∞} obtained from the evaluations. As a consequence, we have no guarantee about the error at locations other than V'. Nonetheless, in our sketching system we assume all our surfaces are *well-behaved* and a good error bound on the set of control vertices should give similar bound for the entire surface.

5.6 Alternatives to fitting

One important reason for performing the fitting is the variable edge length in the mesh produced from our remeshing algorithm. If instead, we have a relatively uniform triangulation, then fitting is not as crucial. This is possible when our original mesh has uniform triangulation. Then when we have a trim curve with uniform edge length, we can choose to trim on a refined mesh (say level n) of the original surface that has comparable edge length. Now, we can perform a remeshing with uniform target length, while *tracking* on M_n instead of M_0 . Because the edge length is uniform and we are tracking on a mesh with comparable resolution, no oversampling artifacts will occur.

While this approach is simpler by skipping the fitting step, trimming on a refined mesh hinders flexibility for subsequent operations, as we are left with a refined mesh to represent our object. For example, if we want to smooth a crease, we would like to be able to set the smoothness in a very coarse level. If our trimmed mesh is based on a refined mesh, however, smoothing is limited as the region we smooth out is only the one-ring neighborhood of the crease edge.

5.7 Joining

When a hole is created, we also create an associated curve around the hole. New parts can then be attached to the hole by building new shapes from this curve.

At the current stage, we do not support Boolean operations of surfaces. However, it should be relatively straightforward to support them once we acquire the intersection curve of two shapes. With this curve, we can cut the two meshes and join them together. Then we can apply our remeshing algorithm to the joining region for better surface approximation.

Chapter 6

Results

The following are some shapes we are able to model with Freeform Sketch. The process for building them is no more complicated than the description given above. Each of them typically take about a minute to build.

- Mug described in Chapter 3.
- **Teapot** start from a tube that represent the main body, attach a handle to it, and attach another tube to form the spout.
- Mushroom two tubes attached together.
- **House** draw a rectangle for the base, extrude it upwards to form the main body. Cut a hole for the door, extrude it to give it thickness. Cut a hole on the top, attach a tube to make the mosque-like roof. Cut a hole through the wall to make the window.

Please also see http://www.cs.princeton.edu/~waingan/ffs.avi for a video demo showing how a mug can be made in our system.



Figure 6.1: Models we made in Freeform Sketch (1).



Control mesh before trimming



Control mesh after trimming



Limit surface before trimming



Limit surface after trimming





Figure 6.3: A *face* built using trim and join on a tube.

Chapter 7

Discussion and Conclusion

This thesis presents the usage of subdivision models in our 3D sketching system. We see that subdivision surfaces are extremely flexible and suitable for our system. One of the important property is the smoothing it provides, as sketching input is often noisy. Also, while the approximating properties of the Loop scheme is not desired in some applications like CAD, it posed no problem for a sketching system like ours. Its convenience for specifying shapes with arbitrary topology, and its ability to define creases with variable sharpness are very attractive for our system.

One of the main focus of the thesis is a trimming algorithm for subdivision surfaces. We initially aim at a simple implementation as accuracy is not our main concern. However, results show that a naive approach will give very bad-looking results, with wiggles and bumps everywhere. A great amount of effort has been spent on getting trimmed surface with better quality, at least visually, while not adding multi-resolution details to the surface. In two related works [12, 11] we discussed in Chapter 5, they both require putting multi-resolution details to several subdivision levels after trimming. This is undesirable for us as it is cumbersome to apply subsequent operations.

The remeshing and fitting algorithm we present in Chapter 5 is giving good results for most cases, but it is not yet ideal. While we are able to control the errors at the positions corresponding to control vertices, in some occasions very subtle bumps appear, even the triangulations we have in the region are reasonably regular (See Figure 7.1). We do not have an

explanation for why this is happening, but this point out an important fact that the L^{∞} norm is not always a good measure for approximation of surfaces. In our case, we do have a small L^{∞} norm for the distance between the trimmed and the original surface, but subtle bumps can still exist as a coherent region of small deviations can contribute to noticeable disruption, especially when smoothly shaded. In our system, we would actually prefer a smoother surface with a worse L^{∞} norm.



Figure 7.1: Subtle bumps after remeshing, contrast adjusted in the circled region of (b) to emphasize the bump.

For the sketching system itself, our work has shown promising result on the system's potential. We are able to build shapes with complexity previously not possible with similar systems in a very short amount of time. Also, subdivision surfaces give our model higher visual quality, compared to polygon models as in Teddy or SKETCH. Surfaces are in general smooth everywhere, while creases can be specified with variable sharpness when desired. This gives us ability to model both the boxy kind of shapes of SKETCH and the biological shapes of Teddy, and combinations of both.

All the modeling operations in our system are supported through gestural input, which prove to be an easy and efficient way to build shapes, compared to the standard GUI in most commercial systems. It has better correspondence to traditional drawing on paper, and without the complicated GUI, it is much less intimidating for the novice users. Nonetheless, there are still plenty of rooms for improvement on the interface, as currently our gestures are not always obvious to users. More importantly, we should give suitable negative feedbacks or make reasonable guess when a gesture is not recognized.

Indeed, pen interfaces has recently gained much popularity in other application domains like PDA and Tablet PC. Even computer games (e.g.Black and White, 2001, Lionhead Studios.) have started to use gestures for the interface, rather than the standard interface with buttons and keystrokes.

Currently, we are developing new tools to build models from skeletons, which describe the basic structures of the shape like branches. With the skeletons, we can acquire natural parameterization of the models more easily, and this can be used to support oversketching for these shapes. For example, we can oversketch iso-parameter curves to describe deformations to a model.

While we have some notable achievements during the past year, it just seems to open up even more possibilities for 3D modeling in this direction. Some of the more interesting or immediate future research include:

- Further study on the trimming operations We would like to investigate more into the properties of the trimmed surface we obtain, in order to prevent the bumps we show in Figure 7.1. Also it will be interesting to incorporate general Boolean operations which will widen the class of models we can model.
- **Multi-resolution details** It will be desirable to support multi-resolution details as a finishing tool to add features to our model, like embossing.
- Animation by sketching A natural follow-up work for modeling would be animation. In particular, the skeleton-based modeling that we are developing naturally gives the structure for animating. It will be interesting to use a similar gestural interface to describe motions.

Appendix A

Mathematical Properties of Subdivision Surfaces

In this appendix we give a few of the important and well-known mathematical properties of subdivision surfaces without proof. Much more elaborated treatment can be found in [15] and [19].

In the context of subdivision analysis, it is very useful to introduce the notion of the subdivision matrix. The subdivision process can be represented using matrix operations as subdivision rules are linear combinations of vertices. For analysis of continuity, convergence and limit position, we look at the *local* subdivision matrix, which operate on a neighborhood of some vertex.

Let M_r be the array of vertices $(V^r, V_1^r, \ldots, V_n^r)^T$ consisting the one-ring neighborhood of vertex V^r , at subdivision level r (Refer to Figure 2.4). Then, we can characterize subdivision in the neighborhood of V^r by a n+1 by n+1 matrix, S_n , to encode the rules of one step of refinement, which send M_r to M_{r+1} , the neighborhood of the vertex V^{r+1} in the refined mesh. In terms of the matrix, we have

$$\begin{pmatrix} V^{r+1} \\ V_1^{r+1} \\ \vdots \\ V_n^{r+1} \end{pmatrix} = S_n \begin{pmatrix} V^r \\ V_1^r \\ \vdots \\ V_n^r \end{pmatrix},$$

or more concisely,

$$M_{r+1} = S_n \cdot M_r.$$

And so, M_k , the neighborhood of V^0 refined k times, is equal to $S_n \cdot M_{k-1} = S_n^k M_0$. For the Loop scheme,

$$S_n = (1/8) \begin{pmatrix} a & b & b & b & \dots & b & b \\ 3 & 3 & 1 & 0 & \dots & 0 & 1 \\ 3 & 1 & 3 & 1 & \dots & \dots & 0 \\ 3 & 0 & 1 & 3 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & & & & \\ 3 & 0 & \dots & \dots & 1 & 3 & 1 \\ 3 & 1 & \dots & \dots & 0 & 1 & 3 \end{pmatrix},$$

where a = 8 - nb, $b = (1/n)(5 - (3 + 2\cos(\frac{2\pi}{n}))^2/8)$.

For example, when n = 6, $b = \frac{1}{2}$, a = 5

$$S_{6} = (1/8) \begin{pmatrix} a & b & b & b & b & b & b \\ 3 & 3 & 1 & 0 & 0 & 0 & 1 \\ 3 & 1 & 3 & 1 & 0 & 0 & 0 \\ 3 & 0 & 1 & 3 & 1 & 0 & 0 \\ 3 & 0 & 0 & 1 & 3 & 1 & 0 \\ 3 & 0 & 0 & 0 & 1 & 3 & 1 \\ 3 & 1 & 0 & 0 & 0 & 1 & 3 \end{pmatrix},$$

Indeed the local subdivision matrix is just another way of representing the masks shown in Figure 2.5. The first row of the matrix map V^r to V^{r+1} by the mask (a, b, b, ..., b), and the remaining rows map V_i^r to V_i^{r+1} by the mask in Figure 2.5.

For a well-defined scheme like Loop, the neighborhood will converge to a single point in the limit by applying S_n repeatedly. In other words, as $k \to \infty$, each of V_i^k in M^k will converge to the limit point V^{∞} . So, to analyze the local property of a subdivision surface at a vertex, it is sufficient to analyze the local subdivision matrix. By eigenanalysis, it is known that V^{∞} is given by

$$V^{\infty} = \frac{(wV^0 + V_1^0 + \ldots + V_n^0)}{w + n}$$

where w = 3/b.

Also, a well-defined tangent plane at V^{∞} on the limit surface will be spanned by two vectors,

$$v_1 = \sum_{i=1}^n \cos(\frac{2\pi(i-1)}{n}) V_i^0, \ v_2 = \sum_{i=1}^n \sin(\frac{2\pi(i-1)}{n}) V_i^0$$

The simple closed form solution for V^{∞} is the reason that evaluation is possible in constant time, and evaluation on arbitrary parameters is also based on this fact. Indeed, with arbitrary evaluation, a subdivision surface can be regarded as a patch locally. In our system, we use the evaluation methods during the fitting stage, described in Section 5.5.

Bibliography

- [1] Autodesk. 3ds max 4., 2001.
- [2] R.H. Bartels, J.C. Beatty, and B.A. Barsky. An Introduction to Splines for Use in Computer Graphics and Geometric Modeling. Morgan Kaurmann, Los Altos, California, 1989.
- [3] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10*, pages 350–355, 1978.
- [4] Jonathan M. Cohen, Lee Markosian, Robert C. Zeleznik, John F. Hughes, and Ronen Barzel. An interface for sketching 3D curves. 1999 ACM Symposium on Interactive 3D Graphics, pages 17–22, April 1999. ISBN 1-58113-082-1.
- [5] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. *Proceedings of SIGGRAPH* 98, pages 85–94, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.
- [6] Nira Dyn, David Levin, and John A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.
- [7] Foley, van Dam, Feiner, and Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley, 1996.
- [8] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John Mc-Donald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. *Proceedings of SIGGRAPH 94*, pages 295–302, July 1994. ISBN 0-89791-667-0. Held in Orlando, Florida.

- [9] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *Proceedings of SIGGRAPH 93*, pages 19–26, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.
- [10] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. *Proceedings of SIGGRAPH 99*, pages 409–416, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.
- [11] Daniel Kristjansson, Henning Biermann, and Denis Zorin. Approximate boolean operations on free-form solids. *CAGD*, 2000. Held in Oslo, Norway.
- [12] Nathan Litke, Adi Levin, and Peter Schröder. Trimming for subdivision surfaces. *CAGD*, 2001.
- [13] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.
- [14] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John F. Hughes. Skin: A constructive approach to modeling free-form shapes. *Proceedings of SIGGRAPH 99*, pages 393–400, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.
- [15] Jean E. Schweitzer. *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, August 1996.
- [16] Jos Stam. Evaluation of loop subdivision surfaces. SIGGRAPH 98 CDROM Proceedings, 1998.
- [17] Jos Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. *Proceedings of SIGGRAPH 98*, pages 395–404, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.
- [18] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. SKETCH: An interface for sketching 3D scenes. *Proceedings of SIGGRAPH 1996*, pages 163–170, 1996.
- [19] D. Zorin and P. Schröder. Subdivision for modeling and animation. *Course notes of SIGGRAPH 2000*, August 2000.