COMPARISON OF FOUR APPROXIMATING SUBDIVISION SURFACE SCHEMES

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by
Tekin Kabasakal
August, 2002

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Bülent Özgüç

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür Ulusoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

COMPARISON OF FOUR APPROXIMATING
SUBDIVISION SURFACE SCHEMES

Tekin Kabasakal
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. Uğur Güdükbay
August, 2002

The idea of subdivision surfaces was first introduced in 1978, and there are many methods proposed till now. A subdivision surface is defined as the limit of repeated recursive refinements. In this thesis, we studied the properties of approximating subdivision surface schemes. We started by modeling a complex surface with splines that typically requires a number of spline patches, which must be smoothly joined, making splines burdensome to use. Unlike traditional spline surfaces, subdivision surfaces are defined algorithmically. Subdivision schemes generalize splines to domains of arbitrary topology. Thus, subdivision functions can be used to model complex surfaces without the need to deal with patches.

We studied four well-known schemes Catmull-Clark, Doo-Sabin, Loop and the $\sqrt{3}$-subdivision. The first two of these schemes are quadrilateral and the other two are triangular surface subdivision schemes. Modeling sharp features, such as creases, corners or darts, using subdivision schemes requires some modifications in subdivision procedures and sometimes special tagging in the mesh. We developed the rules of $\sqrt{3}$-subdivision to model such features and compared the results with the extended Loop scheme. We have implemented exact normals of Loop and $\sqrt{3}$-subdivision since using interpolated normals causes creases and other sharp features to appear smooth.

Keywords: computational geometry and object modeling, subdivision surfaces, Loop, Catmull-Clark, Doo-Sabin, $\sqrt{3}$-subdivision, modeling sharp features.

ÖZET

DÖRT YAKLAŞIMSAL YÜZEY ALT BÖLÜMLEME YÖNTEMİNİN KARŞILAŞTIRILMASI

Tekin Kabasakal
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Uğur Güdükbay
Ağustos, 2002

Yüzey alt bölümleme fikri ilk olarak 1978 yılında öne sürülmüş ve o günden beri bir çok yöntem ortaya konmuştur. Yüzey alt bölümleme, bir sınır değerine kadar kendi kendini yineleyen bir iyileştirme olarak tanımlanmaktadır. Bu araştırmada yaklaşımsal yüzey alt bölümleme yöntemlerinin özellikleri incelenmiştir. Karmaşık yüzeylerin kama eğrileri ile modellenmesi, çok sayıda yivli kama yamasının incelikle birleştirilmesini gerektirdiğinden kullanılmaları zahmetlidir. Geleneksel yüzey kama yamalarının tersine yüzey alt bölümleme yöntemleri iyi tanımlanmıştır ve karalıdırlar. Yüzey bölümleme yöntemlerinin kama eğrilerini düzensiz yapılara uygulanacak şekilde genellemesi, karmaşık yüzeylerin yamalarla uğraşmadan modellenebilmesine imkan vermektedir.

Bu çalışmada, Catmull-Clark, Doo-Sabin, Loop ve $\sqrt{3}$-bölümleme yöntemleri incelenmiştir. Yöntemlerden ilk ikisi dörtgen, diğer ikisi üçgen yüzeyler için önerilmiştir. Yüzey bölümleme ile kırışıklar, kıvrımlar ve bükülme noktaları gibi keskin hatlı yüzeylerin modellenmesi yüzey bölümleme yöntemlerinin düzenlenmesi ve ağın işaretlenmesini gerektirmektedir. $\sqrt{3}$-bölümleme yönteminin kuralları keskin hatlı yüzeyleri modelleyecek şekilde geliştirilmiş ve sonuçları geliştirilmiş Loop yönteminin sonuçları ile karşılaştırılmıştır. Ara kestirim normallerinin kullanılması keskin hatların yumuşak görünmesine neden olduğundan Loop ve $\sqrt{3}$-bölümleme yöntemlerinin kesin normal değerleri hesaplanarak kullanılmıştır.

Anahtar Sözcükler: sayısal geometri ve nesne modelleme, yüzey alt bölümleme, Loop, Catmull-Clark, Doo-Sabin, $\sqrt{3}$-bölümleme, keskin hatların modellenmesi.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Assist. Prof. Dr. Uğur Güdükbay, for his guidance, suggestions, and invaluable encouragement throughout the development of this thesis.

I am also indebted to Prof. Dr. Bülent Özgüç and Assoc. Prof. Dr. Özgür Ulusoy for showing keen interest to the subject matter and accepting to read and review this thesis.

I am grateful to my family and my friends for their infinite moral support and help. I owe special thanks to my friend Cenk Şen for reading this thesis. Special thanks to Turkish General Staff for giving me the opportunity to complete this study.

Finally, I would like to thank to my wife Kadriye for her endless patience while I spent untold hours in front of my computer. Her support in so many ways deserves all I can give.

to my beloved wife

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

CAGD	: Computer Aided Geometric Design
CG	: Computer Graphics
OS	: Operating System
SS	: Subdivision Surfaces
λ_i	: eigenvalue
x_i	: eigen vector
$N_n(t)$	: B-spline basis function of degree n
$\otimes$	: convolution
$U(t)$	: characteristic function
S	: subdivision matrix
$r(u)$	: curve function
C^i	: tangent plane continuation of degree i
t_i	: tangent vector
L	: corner-lath element
F	: faces
E	: edges
V	: vertices

Chapter 1

Introduction

1.1 Overview

Subdivision Surfaces have become a valuable tool in geometric modeling for Computer Graphics (CG) and Computer Aided Geometric Design (CAGD). In recent years, subdivision surfaces have received a lot of attention from both academics and industry professionals. Movie industry professionals apply subdivision techniques to create complex characters and to produce highly detailed, smooth animation. For example, in *Geri's game* from Pixar Animation Studios [4], Geri's hands, head, jacket, and pants were each modeled using a single subdivision surface. The high quality graphics, light maps, and character animations in the game Quake 3 are very impressive. Although they have done an excellent job in painting the textual details, most of the characters consist of only several hundred triangles, which cannot capture highly detailed geometry. Because of the recursive nature, subdivision naturally accommodates level-of-details control through subdivision. This allowed the narrator of the game Quake 3 to spent triangle budgets in regions where more details are needed by subdividing further.

The idea of subdivision surfaces is first introduced by Catmull and Clark [3], and Doo and Sabin [5] independently in 1978. Unlike traditional spline surfaces, subdivision surfaces are defined algorithmically. Simplicity, efficiency, and ease of implementation are the main advantages of the subdivision surfaces. The shape of a subdivision surface is determined by a structured mesh of control points and a set of subdivision rules

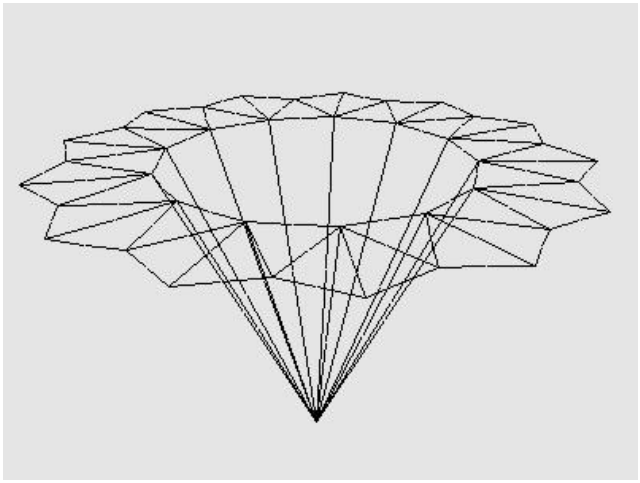
prescribing a procedure for refining the mesh to a finer approximation. The subdivision surface itself is defined as the limit of repeated recursive refinements. Subdivision surfaces satisfy all the usual requirements for surface representation that the computer graphics practitioners are confronted with.

A subdivision surface is the limit of a recursive refinement process. Starting with an initial polygonal mesh of arbitrary topology, a subdivision scheme is used to generate a new mesh that is the initial mesh for the next refinement. A sample subdivision surface is given in Figure 1.1. The repetitive application of this process will generate a sequence of polygonal meshes whose limit may be a smooth surface, assuming appropriate conditions are satisfied.

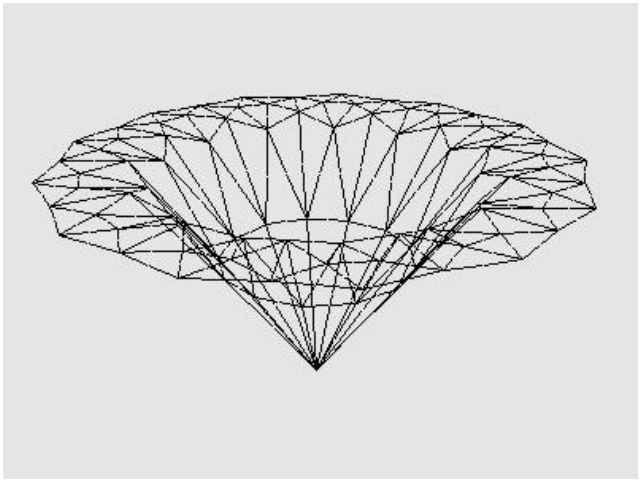
Subdivision surfaces lie somewhere in between polygon meshes and patch surfaces, and offer most of the best attributes of each. The well defined surface normal allows them to be rendered smoothly without the faceted look of low polygon count polygonal geometry, and they can represent smooth surfaces with arbitrary topology (with holes or boundaries) without the restriction in patches where the number of columns and rows has to be identical before two patches can be merged. Secondly, subdivision surfaces are constructed easily through recursive splitting and averaging: splitting involves creating new faces by removing one old face, averaging involves taking a weighted average of neighboring vertices for the new vertices. Because the basic operations are so simple, they are very easy to implement and efficient to execute. Besides, subdivision naturally accommodates level-of-details control through adaptive subdivision because of its recursive nature.

The simple splitting process usually starts from a coarse control mesh, iterating this process several times will produce so-called *semi-regular meshes*. A vertex is regular if it has six neighbors (in triangle mesh) or four neighbors (in quadrilateral mesh). Vertices which are not regular are called *extraordinary*. Meshes that are coming from standard modeling packages or 3D scanning devices usually do not have a regular structure, hence there is a need to convert these irregular meshes into semi-regular meshes, a process known as *remeshing*.

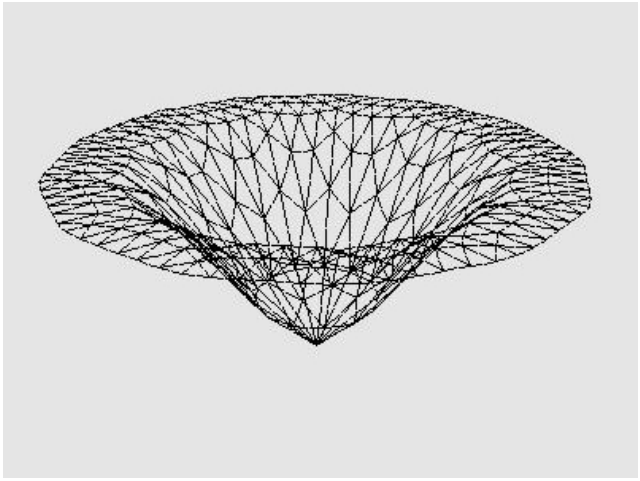
Recursive nature of subdivision surfaces lacks in modeling sharp features, such as creases, corners and dart vertices. By making some modifications to the subdivision rules and adding some new rules for special cases, this problem can be overcome and



(a)



(b)



(c)

Figure 1.1: An example subdivision surface. (a) initial coarse mesh, (b) mesh after first subdivision step, (c) mesh after second subdivision step.

sharp features could be modeled.

The main focus of this work will be on implementing and comparing some subdivision schemes in a common framework, including sharp features on the triangular meshes where one of the methods, $\sqrt{3}$-subdivision, extended to represent sharp features.

1.2 The Organization of the Thesis

The organization of the thesis is as follows: In Chapter 2, we define the basic principles behind subdivision, how subdivision rules are constructed; how their analysis approached. We first define uniform B-spline surfaces and explain how we can extend B-spline curves to B-spline surfaces. Then, subdividing a B-spline curve and extending the method to B-spline surfaces are shown. We take subdivision as a method for defining curves and surfaces. Topological restrictions and computational difficulties of B-splines are mentioned and how subdivision surfaces eliminate these disadvantages are discussed.

In Chapter 3, the approximating subdivision schemes that we deal are introduced. Their subdivision methods and smoothing rules are described with their normal calculations. Here, we also describe the developments we did for the $\sqrt{3}$-subdivision scheme rules to model sharp features and to calculate exact normals.

In Chapter 4, we discuss the implementation. First, we outline the underlying reasons for choosing the corner lath data structure, and explain the chosen data structure. Second, we briefly explain the implementation of the data structure and modifications we did to manage subdivision and tagging knot spacings. Finally, we describe the algorithms we have implemented in order to support each of the schemes. The main ideas behind each of the implementations are mentioned here.

In Chapter 5, the results of applying the implemented subdivision surface schemes to a variety of meshes and their behaviours at these mesh conditions are compared by examining the images of rendered example surfaces. We highlight certain properties of the schemes by examining a number of surfaces rendered using the implemented *Interface* program. We base our comparisons on three criteria: *smoothness of the surfaces produced*, *computational work to be done*, and *modeling the sharp features.*

The sharp features are compared only for the triangular subdivision schemes.

Conclusions and future research directions are given in Chapter 6.

In Appendix A, the basis for the B-spline functions and curves are explained briefly. We define the uniform B-spline basis functions and use them to construct piecewise polynomial curves. We choose repeated convolution to derive B-splines, since, we can see from it directly how splines can be generated through subdivision. Techniques for analysis of subdivision surfaces are then examined.

To model objects using subdivision surfaces we have to use some utilities to support input handling and the rendering of the modeled surfaces. The utilities except the representation and manipulation of data, such as data formats supported, rendering process of surfaces, etc., are discussed in Appendix B.

Chapter 2

B-spline and Subdivision Surfaces

2.1 B-spline Surfaces

Subdivision surface schemes are usually originated from spline curves and surfaces [18]. Splines are piecewise polynomial curves of some chosen degree. Uniform B-spline basis functions are used to construct piecewise polynomial curves. B-splines can be derived in many ways. One of them is repeated convolution and briefly explained in Appendix A.

B-spline curves can be extended to surfaces by some methods. Here, we will briefly discuss tensor product B-spline surfaces and triangular box spline surfaces. Then, we will expand our view on subdivision to define curves and surfaces as the limit of repeated application of subdivision rules.

2.1.1 Tensor Product B-spline Surfaces

A surface representation can be obtained through the concept of a tensor product. We start with a curve.

$$r(u) = \sum_i F_i(u) c_i, \tag{2.1}$$

where $F_i(u)$ are some basis functions. Then, we move the curve while simultaneously allowing it to alter shape. The motion and deformation in shape can be described by

c_i where it is also a function of v as

$$c_i(v) = \sum_j G_j(v) p_{ij}, \tag{2.2}$$

where $G_j(v)$ is another set of basis functions. This process swept out a surface, which has the parameterization

$$r(u, v) = \sum_i \sum_j F_i(u) G_j(v) p_{ij}, \tag{2.3}$$

The control points p_{ij} can be organized in a mesh structure. For a finite surface, there are a finite number of control points. The parameter values run over a rectangular domain.

Replacing the arbitrary functions in above equation with B-spline functions produces tensor product B-spline surfaces.

$$r(u, v) = \sum_i \sum_j N_i(u) N_j(v) p_{ij}, \tag{2.4}$$

The construction of tensor product B-spline surfaces indicates that subdivision of a surface is closely related to subdivision of curves. We can apply the subdivision process for B-spline curves in Equation A.20 to the control points c_i.

Rules for subdivision surfaces can easily be described by *masks*. The mask is supposed to be applied to all parts of the mesh and for each application a new control point calculated using the coefficients in the masks.

2.1.2 Triangular Box Spline Surfaces

The spline surfaces considered in the previous section are defined on a rectangular parameter domain. It is also possible to define a spline surface over a regular triangulation of the parameter plane.

Triangular splines can be defined by using the piecewise linear box spline, which is a hexagonal pyramid with the apex at height 1. The higher degree splines are then can be obtained through convolution along the three directions simultaneously.

The triangular splines share many of the properties of B-splines. They have local support, they are non zero, and their translates forms a partition which can be in any of the three directions. It is possible to derive a subdivision equation for the triangular splines similar to B-splines.

2.2 Curves and Surfaces Defined by Subdivision

In the previous chapter we examined subdivision of spline curves and surfaces. We examined under repeated subdivision how the control polygon converged to the spline curve and similar results can be shown for surfaces. The subdivision equation is geometric interpretation of subdivision process: new control points are calculated by using the old control points. This geometric approach provides the inspiration for using subdivision to define curves and surfaces.

The problem is choosing coefficients for subdivision that converges to a smooth curve or surface. In other words, given the averaging masks, what are the properties of the limit curve/surface? This question includes determining, whether a limit object exists, and, whether that object is smooth. And another problem is also identifying those masks that allow the mesh approach to a limit curve or surface.

2.3 Arbitrary Topology Surface Schemes

In Section 2.1, we briefly described two types of surfaces —*tensor product B-spline surfaces* and *triangular box spline surfaces*. For tensor product B-spline surfaces the control mesh must be a regular quadrilateral grid, while for triangular box spline surfaces the control mesh must be a regular triangular mesh. This imposes a restriction on the topology of the surface that can be modeled.

The topological restrictions on spline based surface modeling methods are overcame with subdivision surfaces. The early subdivision surface schemes such as those proposed by Doo and Sabin, Catmull and Clark, and Loop, generalize spline surface schemes to arbitrary topology by specifying additional mask for irregular vertices in the control mesh.

An important property of subdivision surface schemes is that the number of irregular vertices and faces remains constant after first subdivision. As subdivision progresses the irregular features are separated by larger patches of regular faces. This allows us to analyze isolated irregular features. The limit position of the control point associated with an irregular vertex is called an *extraordinary* point.

2.4 Classification of Subdivision Surface Schemes

In literature there are many known stationary subdivision schemes generating continuous surfaces on arbitrary meshes. There are wholly different classes of subdivision surfaces. At first glance the variety of existing schemes may appear confusing. On the other hand the schemes can be classified straightforwardly based on four criteria

- **Refinement rule:** The refinement rule is an algorithm to produce a finer tiling of the same type, from a given tiling. The most common refinement rules are the *face split* and *vertex split*.The face split rule can be applied to both triangular and quadrilateral meshes. Usually, each face is split into four faces of the same type —in $\sqrt{3}$-subdivision scheme faces split into three. The vertex-split rule can only be applied to quadrilateral meshes. With this rule every vertex is split into four new vertices. Subdivision schemes using the face split rules are called *primal*, while schemes using the vertex split rules are called *dual*. See Figure 2.1 for face split and vertex split refinement rules.

- **Mesh type:** Regular subdivision schemes act on regular control meshes. The regular mesh should be homomorphic to a tiling of the plane. For regular meshes it's natural to use faces that are identical. If the faces are also required to be regular polygons, there are only three possibilities: triangular, quadrilateral and hexagonal tiling. Meshes consisting of hexagons are not very common and most subdivision schemes operate on regular triangular or regular quadrilateral tiling. These lead to two types of regular subdivision schemes: those defined for quadrilateral tilings and those defined for triangular tilings.

- **Interpolating or approximating:** A subdivision scheme is interpolating if all control points in the original mesh are also the control points in all the meshes

obtained through subdivision, otherwise the scheme is approximating. In interpolation control points defining the initial mesh are also the points of the limit surface, which allows one to control it in an intuitive manner. This property also allows some algorithms to be considerably simplified. Unfortunately, the quality of these surfaces are not good as the quality of the surfaces produced by approximating schemes, and these schemes do not converge to the limit surface as fast as the approximating schemes.

- **Smoothness:** The order of continuity on the regular parts of the subdivision surface depends on the scheme used. Most schemes produce C^1 or C^2 order surfaces. The order of continuity near extraordinary points is usually lower than that of regular parts of the surface.

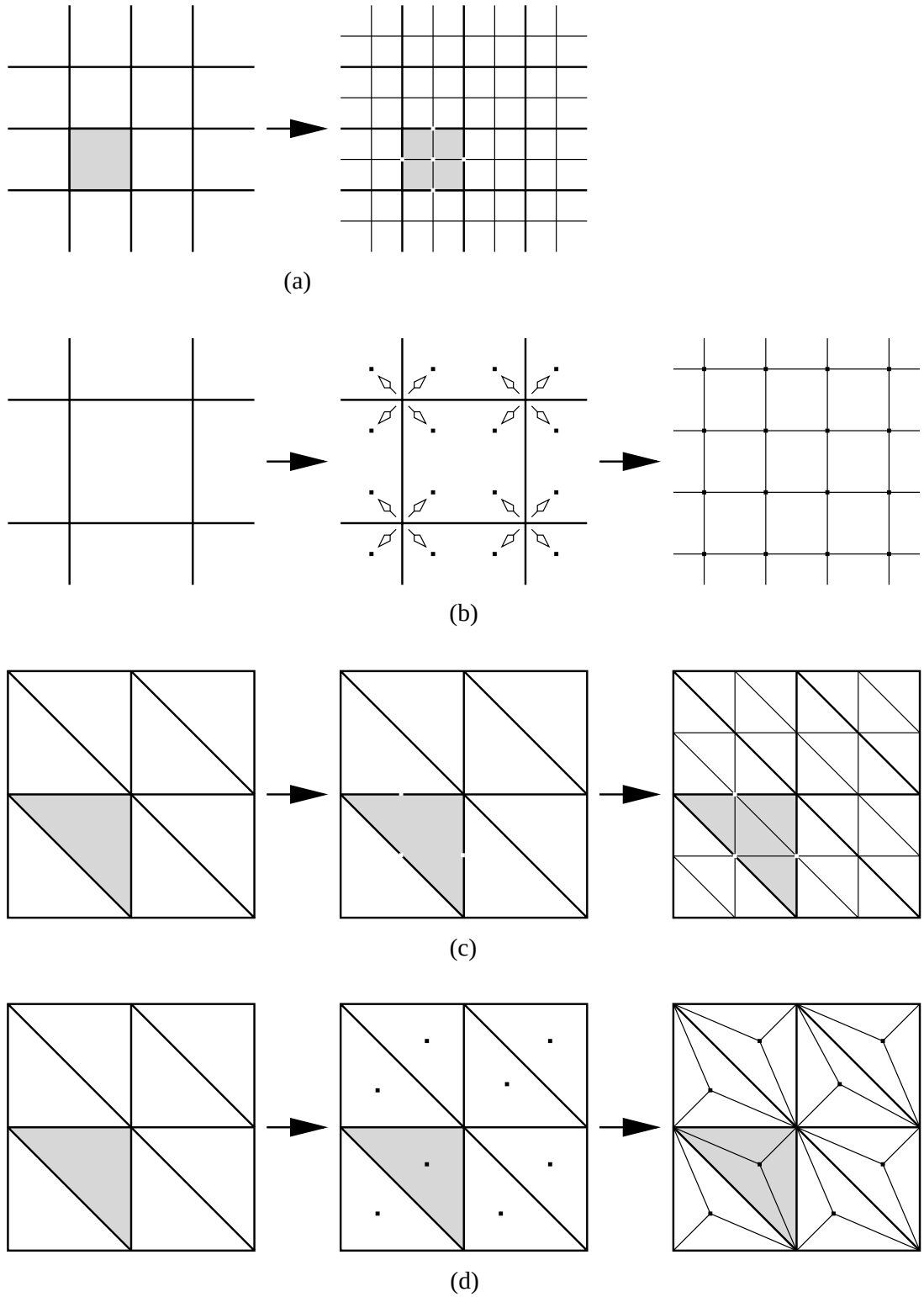


Figure 2.1: Face split and vertex split refinement rules. (a) quadrilateral face split, (b) quadrilateral vertex split, (c) triangular 1-to-4 split, and (d) triangular 1-to-3 split.

Chapter 3

Approximating Subdivision Schemes

As we have seen the subdivision schemes are a big family. In our study, we focused on four different approximating stationary subdivision schemes. Two of them quadrilateral and the other two triangular schemes. Modeling sharp features using subdivision schemes make these schemes valuable. But this requires some modifications in subdivision rules and tagging in the mesh. Loop subdivision scheme was extended by Hoppe et al. [9] and further by Schweitzer [15]. We developed subdivision rules for $\sqrt{3}$-subdivision scheme [10] proposed by Leif Kobbelt to model sharp features.

The schemes considered in our work are categorized as follows:

- **$\sqrt{3}$-subdivision:** triangular, face split, approximating, C^2.
- **Loop subdivision:** triangular, face split, approximating, C^2.
- **Doo-Sabin subdivision:** quadrilateral, vertex split, approximating, C^1.
- **Catmull-Clark subdivision:** quadrilateral, face split, approximating, C^2.

3.1 $\sqrt{3}$-Subdivision Scheme

$\sqrt{3}$-subdivision scheme is an approximating face split scheme defined on triangular meshes presented by Leif Kobbelt [10]. $\sqrt{3}$-subdivision scheme generates C^2 surfaces for regular control meshes. For arbitrary control meshes Kobbelt show that the limit surface to be C^2 almost everywhere except for the extraordinary vertices (valence $\neq 6$) where the smoothness is at least C^1.

In $\sqrt{3}$-subdivision scheme a 1-to-3 split operation performed for every triangle by inserting a new vertex at its center. This introduces three new edges connecting the new vertex to the surrounding old ones. In order to re-balance the valence of the mesh vertices, every original edge that connects two old vertices flipped. Applying this split operation to a uniform grid generates a (rotated and refined) uniform grid. Applying the same refinement operator twice splits every original triangle into nine subtriangles *tri-adic split*. When $\sqrt{3}$-subdivision applied to arbitrary triangle meshes, all newly inserted vertices have exactly valence six. The valence of old vertices are not changed, also the number of extraordinary vertices stay the same during every step of subdivision.

The stencil for the relaxation of the old vertices is the 1-ring neighborhood containing the vertex itself and its direct neighbors. The labelling of the vertices is shown in Figure 3.1. For symmetry reasons the same weight is assigned to each neighbor. For a vertex p, the relaxation is calculated by

$$S(p) = (1 - \alpha_n)p + \frac{\alpha_n}{n} \sum_{i=0}^{n-1} p_i, \tag{3.1}$$

where the he value of α_n is calculated as

$$\alpha_n = \frac{4 - 2\cos\left(\frac{2\pi}{n}\right)}{9}, \tag{3.2}$$

where n is the valency of the vertex. The masks used for all types of vertices in $\sqrt{3}$-subdivision are shown in Figure 3.2.

$\sqrt{3}$-subdivision has many benefits. First, it is very natural to subdivide triangular faces at their center instead of splitting all three edges. Second, the $\sqrt{3}$-refinement is slower than the usual dyadic split operators since the number of vertices and faces

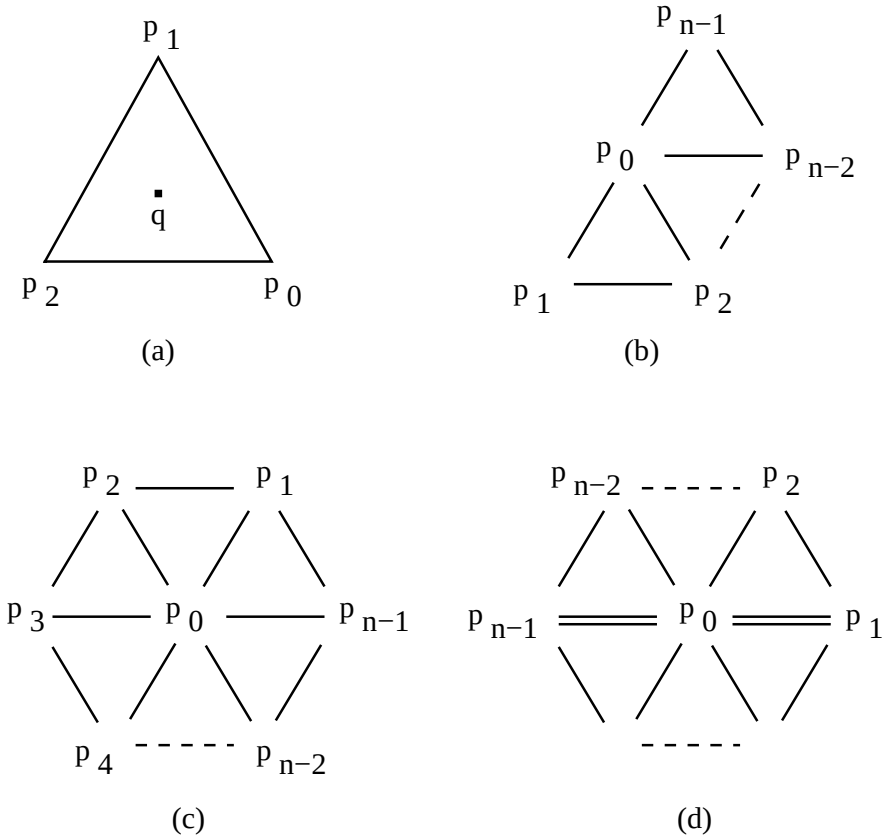


Figure 3.1: Labeling control points for $\sqrt{3}$-subdivision scheme. (a) face vertex, (b) boundary vertex, (c) smooth or dart vertex, and (d) crease vertex.

increases by the factor of 3 instead of 4. Third, it enables a very simple implementation of adaptive refinement with no inconsistent intermediate state, and it's easy to implement. These properties makes $\sqrt{3}$-subdivision refinement suitable for many applications.

3.1.1 Piecewise Smooth Surfaces by $\sqrt{3}$-Subdivision

Fitting smooth surfaces to non-smooth objects often produces unacceptable results. Since the $\sqrt{3}$-subdivision useful for many applications, we developed new subdivision rules to accurately model objects with tangent discontinuities. These subdivision rules produce commonly occurring sharp features that we call *creases, corners* and *darts* as Hoppe et al. [9] did for the Loop scheme. A *crease* is a tangent line smooth curve

along which the surface is C^0 but not C^1 or C^2; a *corner* is a point where three or more creases meet; finally a *dart* is a point of surface where a crease terminates. We extended Kobbelt's $\sqrt{3}$-subdivision scheme [10] to model sharp features.

Evaluation of piecewise subdivision surfaces are done by the eigenvalue analysis. By the way, Zorin at. al. [22] extended the work of Stam [17] by considering the subdivision rules for piecewise smooth surfaces with boundaries depending on parameters. They used a different set of basic vectors for evaluation, which, unlike eigenvectors, depend continuously on the coefficients of the subdivision rule. The advantages of this approach is that it becomes possible to define evaluation for parametric families of rules without considering an excessive number of special cases and while improving the numerical stability of calculations.

To model sharp features a subset L of edges in the initial mesh M_0 is tagged as *sharp*. We used a tagging facility as Hoppe et al. [9] and Sederberg et al. [16] did for tagging vertices and edges in the control net, in order to determine which rule to use. In the subdivision process, edges created from the refinement of the sharp edges are marked as sharp.

Tangent plane continuity relaxed by modifying the smoothing rules across sharp edges. To ensure surface to have a well-defined tangent-plane at each point along the crease, subdivision rules at crease edges must be chosen very carefully. Similar attention must be paid to the corner or dart vertices.

We classified vertices as *smooth vertex, dart vertex, crease vertex* and *corner vertex* into four types. A *smooth vertex* is the one where the number of incident sharp edges is zero, a *dart vertex* is the one where the number of incident sharp edges is one, a *crease vertex* is the one where the number of incident sharp edges is two and a *corner vertex* is the one where the number of incident sharp edges is more than two. Figure 3.2 shows our subdivision masks.

We used two types of masks. A face vertex mask which does not depend on the property of the vertices of the corresponding triangle. The vertex masks used for smoothing the vertices on the sharp edges are chosen as explained above for smooth, dart, crease, and corner vertices.

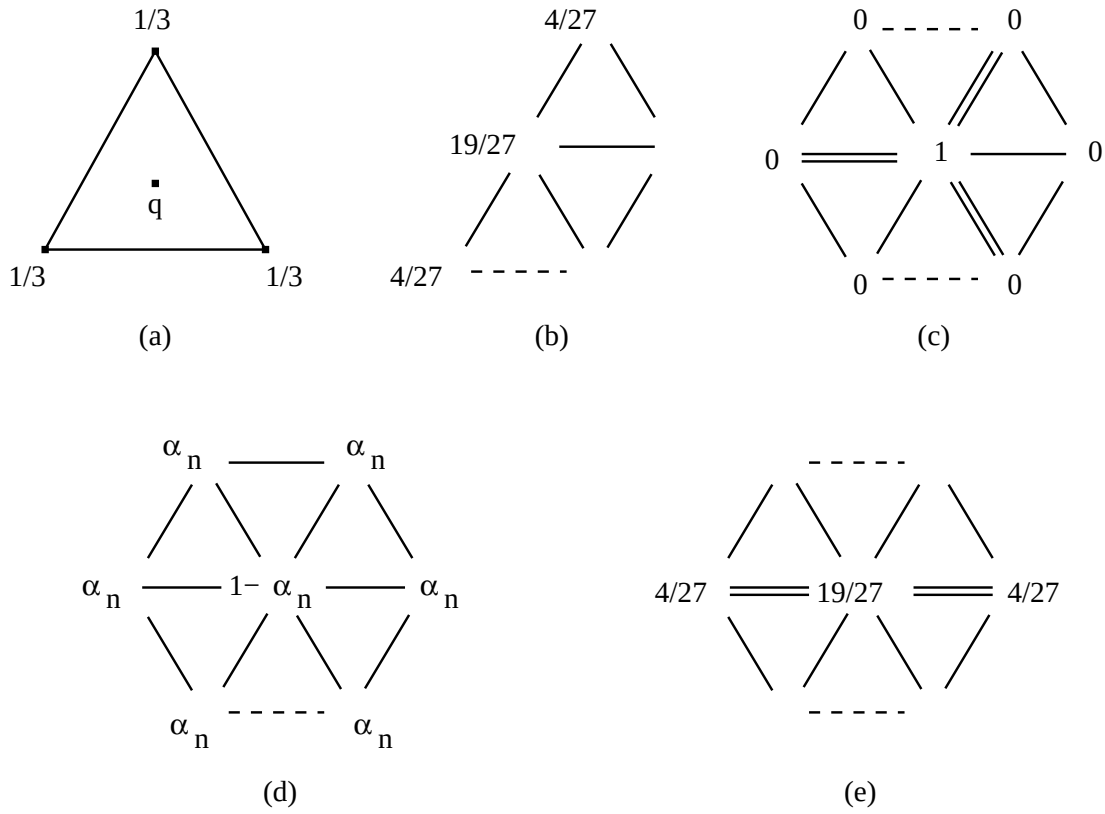


Figure 3.2: Masks for the $\sqrt{3}$-subdivision scheme. Double marked edges are the sharp edges. (a) face vertex, (b) boundary vertex, (c) corner vertex, (d) smooth or dart vertex, and (e) crease vertex.

3.1.1.1 Adapting Surface Smoothing Rules

Limiting points and tangent planes can be computed by using masks, as explained in Section 2.1.1. These masks are determined by the eigen structure of the local subdivision matrices.

Smooth and dart vertices: In the case of smooth or dart vertices the local subdivision matrices describe the whole region and every vertex in the 1-ring neighborhood equally affects the vertex.

Crease vertices: Crease vertices modeled as two sided boundaries. This divides subdivision matrix into two separate matrices each describing a smooth region of the surface. To prevent gaps and cracks we choose to use only the vertices along the crease

in 1-neighborhood of the vertex, to smooth the vertex.

Corner vertices: Corner vertices can also be treated as crease vertices where more than two different regions to be smoothed by different separate decomposed matrices. Corner vertices do not move during subdivision.

3.1.2 Normals of the $\sqrt{3}$-Subdivision

In computer graphics the normal vectors of surfaces are used to calculate lighting effects. The normal vectors of surfaces are usually calculated by interpolating the normals of the faces joining at the vertex. But in the case of piecewise subdivision surfaces which is piecewise smooth this approach will not suffice, since the interpolation will cause creases and other sharp features to appear smooth.

As we mentioned in the previous Chapter, tangent planes can be computed by using masks. Halstead et al. [7] show that the tangent vectors to the surfaces can be computed using the two left eigen vectors of S_n corresponding to the largest eigen value as,

$$t_1 = c_1 v_1^0 + c_2 v_2^0 + \ldots + c_n v_n^0 \tag{3.3}$$

$$t_2 = c_2 v_1^0 + c_3 v_2^0 + \ldots + c_1 v_n^0 \tag{3.4}$$

For $\sqrt{3}$-subdivision, $c_i = \cos(2\pi i/n)$ and their cross product gives an exact normal to the surface. The tangent masks in Figure 3.3 are derived by the Equation 3.3 and 3.4.

At the limit position of a smooth or dart vertex with valency n, the tangent space is spanned by Equation 3.3. At the limit position of a crease vertex, the tangent along the crease is

$$t_{along} = p_1 - p_n, \tag{3.5}$$

where n is the crease valency, i.e. the number of edges between and including the two edges marked as crease edges. The cross tangent vector depends on the valency. For a valency 4 vertex it is

$$t_{across} = (p_2 + p_3)/2 - (p_1 + 2p_0 + p_3)/4. \tag{3.6}$$

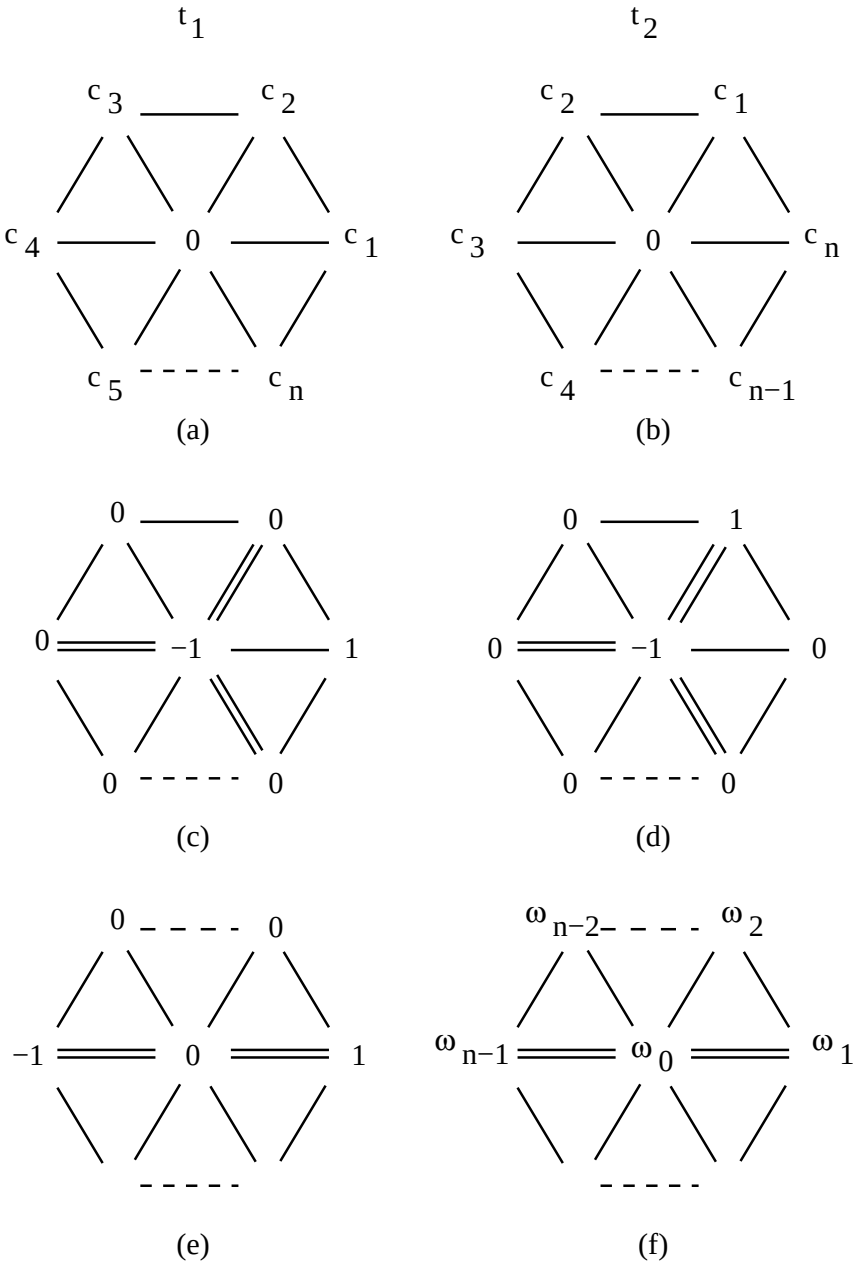


Figure 3.3: Tangent masks for calculating the exact normals of the $\sqrt{3}$-subdivision scheme. (a), (b) smooth or dart vertex, (c), (d) corner vertex, and (e), (f) crease vertex.

For an extraordinary vertex with valency $n > 4$ it is

$$t_{across} = \sum_{i=1}^{n} \omega_i p_i, \tag{3.7}$$

where $\omega_1 = \omega_n = \sin\left(\frac{\pi}{n-1}\right) / \left(2\cos\left(\frac{\pi}{n-1}\right) - 2\right)$ and $\omega_i = \sin\left(\frac{\pi(i-1)}{n-1}\right)$ for $i = 2, \ldots, n-1$. For $n = 2$

$$t_{across} = p_1 + p_2 - 2p_0, \tag{3.8}$$

At a corner vertex the tangent vectors are

$$t_1 = p_1 - p_0, \tag{3.9}$$

$$t_2 = p_n - p_0. \tag{3.10}$$

3.2 Loop Subdivision Scheme

The Loop subdivision scheme is a simple approximating face split scheme for triangular meshes proposed by Charles Loop [12]. The scheme is based on the three-directional box spline, which produces C^2 continuous surfaces over regular surfaces and C^1 at extraordinary points. The scheme can be applied to arbitrary polygonal meshes, after the mesh is converted to a triangular mesh. for example by triangulating each polygonal face.

Hoppe et al. extended the scheme [9] to incorporate sharp features, such as, creases, darts and corner points. Creases are smooth curves across which the surface is C^0 continuous instead of C^1 or C^2 continuous. Corners are points where three or more creases join. A dart is a crease that transitions into a smooth part of the surface.

Schweitzer [15] further extended the scheme with conical and cusp points where the limit surface is C^0. At a conical point, the space spanned by the limiting tangent vectors has dimension three. At a cusp point, the tangent vectors become parallel.

Later the masks for Loop subdivision surface algorithm are modified by Loop [13] to result in surfaces with bounded curvature and the convex hull property.

The full set of masks for the Loop scheme including the described extensions are shown in Figure 3.4. To implement this scheme there must be a facility for tagging edges and vertices in the control net, in order to determine which rule to use.

Loop showed tangent plane continuity of the original scheme. Schweitzer [15] proved C^1 continuity at extraordinary vertices up to valency 100 for the original scheme, and also examined the behaviour of the extended scheme. Zorin [20] proved C^1 continuity at extraordinary vertices for all valences for the original scheme.

3.2.1 Normals of the Loop Scheme

In computer graphics the normal vectors of surfaces are used to calculate lighting effects. In order to make a polygonal model look smooth the normal at a vertex is usually computed by interpolating the normals of the faces joining at the vertex. For the extended Loop scheme, this approach is not satisfactory, since the interpolation will cause creases and other sharp features to appear smooth. One solution to this problem is to use exact normals of the limit surface.

Usually the tangent vectors are used to compute a normal. The rules for computing the tangent vectors of the Loop scheme are especial simple and can be obtained by

$$t_1 = \sum_{i=0}^{k-1} \cos\frac{2\pi i}{k} p_{i,1}, \tag{3.11}$$

$$t_2 = \sum_{i=0}^{k-1} \sin\frac{2\pi i}{k} p_{i,1}. \tag{3.12}$$

These formulas can be applied to the control points at any subdivision level. The normal obtained as the cross product t_1 x t_2 can be interpreted geometrically. The geometric nature of the normals obtained in this way suggests that they can be used to compute approximate normals for other schemes. The exact normals require more complicated expressions. Schweitzer [15] derived equations for the tangent vectors in the extended scheme using eigen analysis. The equations are described in the sequel.

The labeling of the vertices is shown in Figure 3.5(a) for smooth, dart, conical, cusp vertices and in Figure 3.5(b) for crease and corner vertices. The double edges are edges marked as sharp.

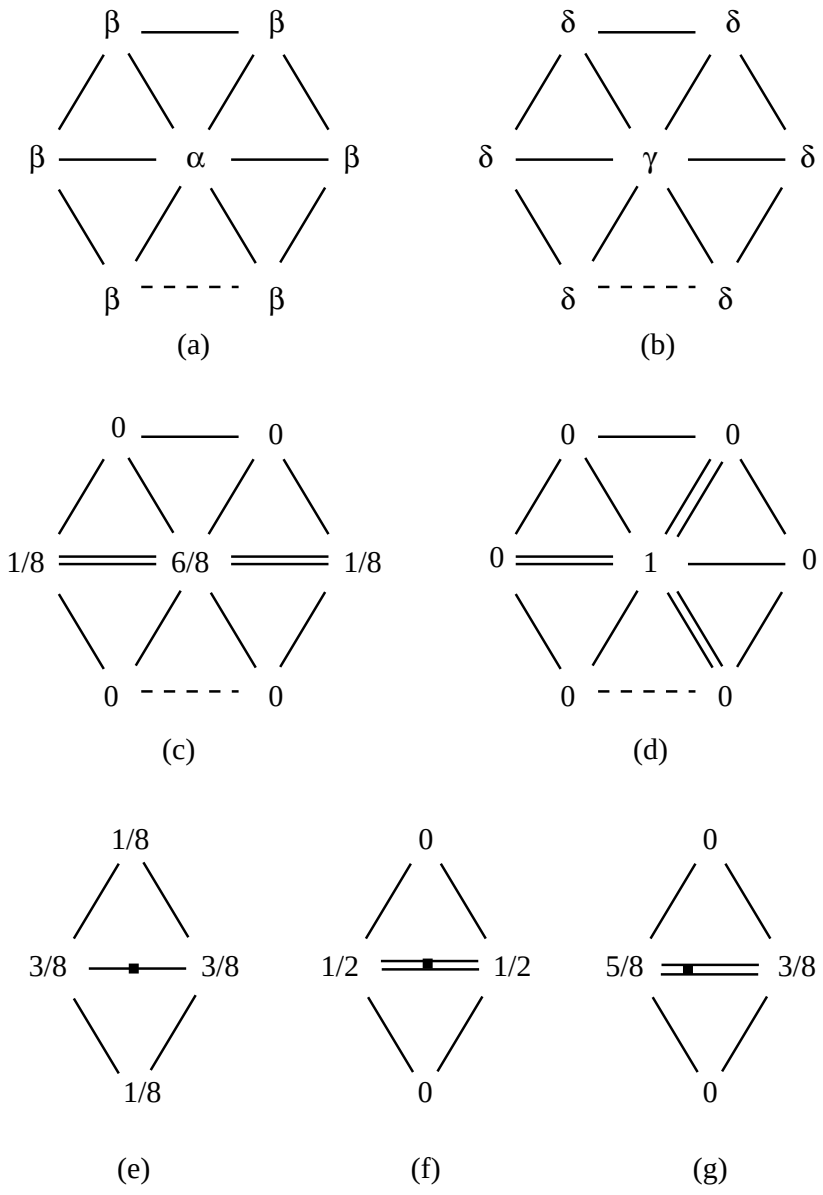


Figure 3.4: Masks for the Loop subdivision scheme. (a) smooth or dart vertex, (b) conical vertex, (c) crease vertex, (d) corner or cusp vertex, (e) smooth edge, (f) ordinary crease edge, and (g) special crease edge.

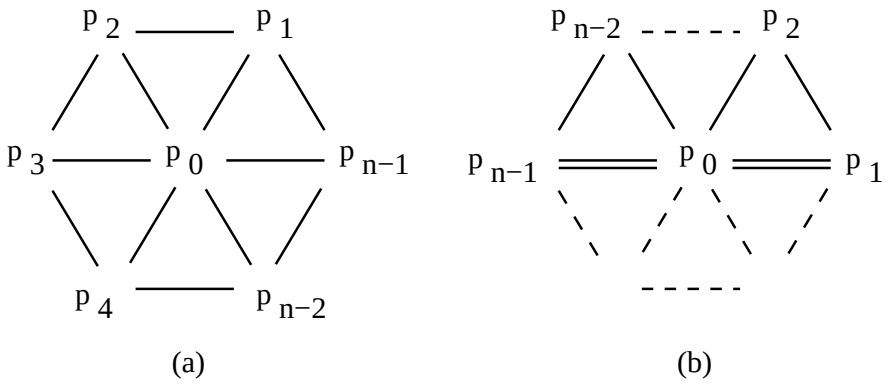


Figure 3.5: Labeling of the 1-neighborhood of (a) a smooth, dart, conical or cusp vertex, and (b) a crease or corner vertex.

At the limit position of a smooth or dart vertex with valency n, the tangent space is spanned by Equation 3.12. At the limit position of a crease vertex, the tangent along the crease is

$$t_{along} = p_1 - p_n, \tag{3.13}$$

where n is the crease valency, i.e. the number of edges between and including the two edges marked as crease edges.

The cross tangent vector depends on the valency. For a valency 4 vertex it is

$$t_{across} = (p_2 + p_3)/2 - (p_1 + 2p_0 + p_3)/4. \tag{3.14}$$

For an extraordinary vertex with valency $n > 4$ it is

$$t_{across} = \sum_{i=1}^{n} \omega_i p_i, \tag{3.15}$$

where $\omega_1 = \omega_n = \sin\left(\frac{\pi}{n-1}\right) / \left(2\cos\left(\frac{\pi}{n-1}\right) - 2\right)$ and $\omega_i = \sin\left(\frac{\pi(i-1)}{n-1}\right)$ for $i = 2, \ldots, n-1$. For $n = 2$

$$t_{across} = p_1 + p_2 - 2p_0, \tag{3.16}$$

while for $n = 3$ the eigen analysis fails because the subdivision matrix does not have a complete set of eigen vectors. The solution is to use generalized eigen vectors, and Schweitzer skipped this case. The Equation above assumes the neighbor vertices are ordinary. If this is not the case, the equations can be applied, after subdividing once.

At a corner vertex the tangent vectors are

$$t_1 = p_1 - p_0, \tag{3.17}$$

$$t_2 = p_n - p_0. \tag{3.18}$$

At a conical point there is no tangent plane. At a cusp point the tangent plane degenerates into a line, which is spanned by

$$t = p_0 - (p_1 + \ldots + p_n)/n. \tag{3.19}$$

3.3 Doo-Sabin Scheme

The Doo-Sabin subdivision scheme [5] is a generalization of uniform biquadratic tensor product B-spline surfaces to arbitrary topology. It is an approximating vertex split (dual) scheme. The subdivision is quite simple since there is only one mask used to compute the new vertices. A special rule is required only for the boundaries. Figure 3.6 shows the general mask, which reduces to the ordinary mask for quadratic tensor product B-spline when applied to a quadrilateral face. The coefficients are defined by the equations $\alpha_0 = 1/4 + 5/4k$ and $\alpha_i = \left(3 + 2\cos\left(\frac{2i\pi}{k}\right)\right)/4k$ for $i = 1, \ldots, k-1$. Another choice of coefficients proposed by Catmull and Clark: $\alpha_0 = 1/2 + 1/4k$, $\alpha_1 = \alpha_{k-1} = 1/8 + 1/4k$ and $\alpha_i = 1/4k$ for $i = 2, \ldots, k-2$. In the subdivision a new face is created for every face, edge and vertex in the control mesh. After one step of subdivision, every vertex has valence four, and the number of extraordinary faces does not change in subsequent subdivisions.

Doo and Sabin initiated analysis of the Doo-Sabin scheme. Later Peters and Reif [14] proved C^1 continuity of a more general class of schemes of which Doo-Sabin is a special case.

Sederberg et. al. [16], introduced features such as cusps, creases, and darts to model sharp features with Doo-Sabin scheme.

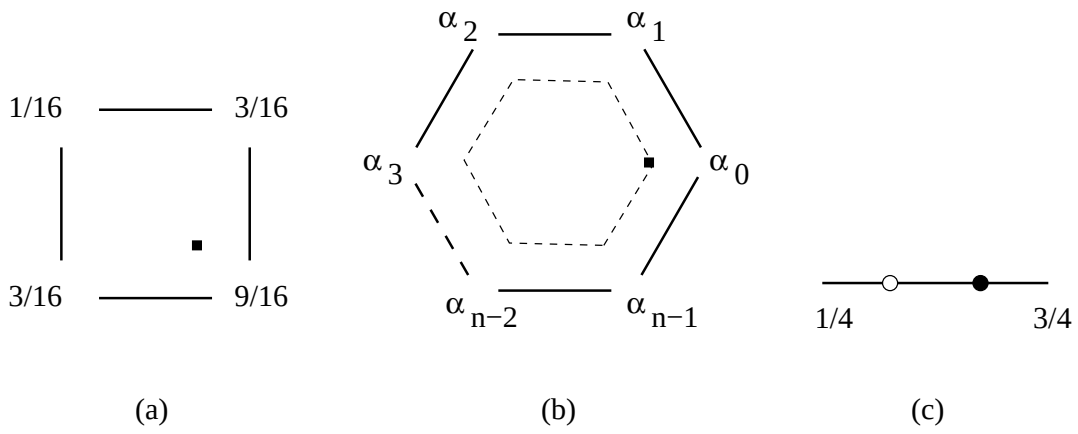


Figure 3.6: Doo-Sabin subdivision scheme masks. (a) regular vertex, (b) extraordinary vertex, and (c) boundary vertex.

3.4 Catmull-Clark Scheme

The Catmull-Clark subdivision scheme [3] was proposed at about the same time as the Doo-Sabin subdivision scheme. Catmull-Clark generalize uniform bicubic B-spline surfaces to arbitrary topology. The masks shown in Figure 3.7 generate new face, edge and vertex points. At extraordinary vertices, Figure 3.7(c), Catmull and Clark use the coefficients $\alpha = 1 - \frac{7}{4n}$, $\beta = \frac{3}{2n}$, $\gamma = \frac{1}{4n}$ where n is the valency.

The Catmull-Clark scheme is an approximating face split, *primal*, scheme, splitting each face with n sides into n quadrilaterals. After one step of subdivision all faces are quadrilaterals. The number of vertices with valence other than four remains constant in subsequent subdivisions. The masks in Figure 3.7 cannot be applied to control meshes with irregular faces. To handle such meshes the rules are extended.

The rules of Catmull-Clark scheme are defined for meshes with quadrilateral faces. Arbitrary polygonal meshes can be reduced to a quadrilateral mesh using a more general form of Catmull-Clark rules [3].

- New face points are computed as the average of the control points associated with the vertices in the original face.
- New edge points are computed as the average of the end points of the edge and

the new face points for the two faces sharing the edge.

- New vertex points can be calculated as

$$P_0^{j+1} = \frac{k-2}{k} P_0^j + \frac{1}{k^2} \sum_{i=0}^{k-1} P_{i,1}^j + \frac{1}{k^2} \sum_{i=0}^{k-1} P_{i,2}^{j+1} \tag{3.20}$$

Ball and Story [1] proved tangent plane continuity of the Catmull-Clark scheme. Peters and Reif [14] proved C^1 continuity for valences up to 10,000. The scheme produces surfaces that are C^2 continuous away from extraordinary vertices where they are C^1.

Sederberg et. al. [16], developed new subdivision rules for generating nun-uniform subdivision surfaces of arbitrary topology with Catmull-Clark scheme. He introduced features such as cusps, creases, and darts, while maintaining the same order of continuity as their counterparts. Havemann [8] did researches on the interactive display of Ctmull-Clark surfaces. He used a three level *caching* strategy so that the display routine can make use of static geometry, but it can also efficiently handle moving control vertices, and it even changes the connectivity of the control meshes.

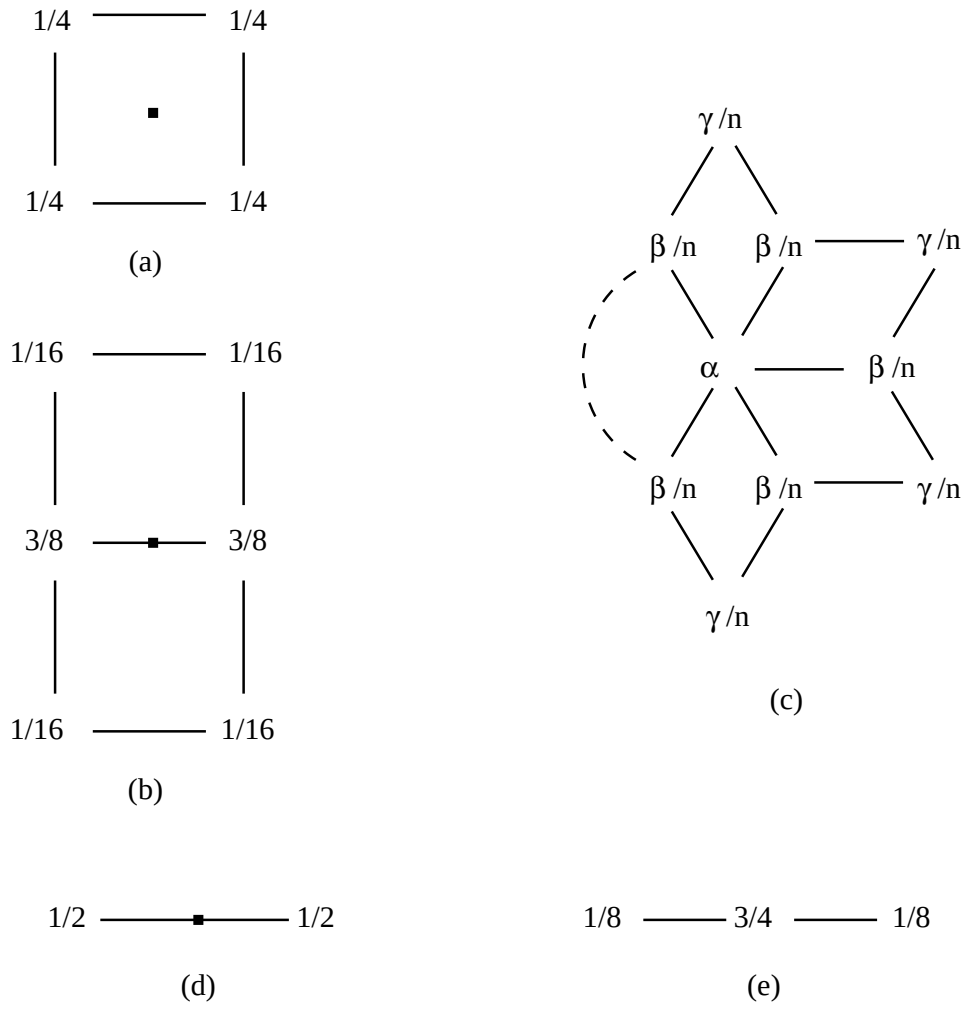


Figure 3.7: Masks for the Catmull-Clark subdivision scheme. (a) face vertex mask, (b) edge vertex mask, (c) even vertex rule, (d) boundary odd vertex mask, and (e) boundary even vertex mask.

Chapter 4

Implementation of Subdivision Surfaces

4.1 Data Structures

One of the fundamental steps we had to cover before starting to implement any of the subdivision algorithm was to design a suitable data structure. A major challenge is to find a data structure that could handle the large and constantly changing data sets involved in subdivision. The chosen structure should, besides representing the data in its native form, also show the connectivity of the mesh in question, while keeping the storage at minimum.

Specifically for our purposes it was important that the data structure could represent almost arbitrary meshes since we worked with different types of subdivision schemes. The schemes operate on mesh types that consist of primarily either triangular or quadrilateral faces. It would be very impractical to have separate data structures for the different mesh types, so an important criterion is the support for almost arbitrary meshes.

There are some restrictions on the meshes since we deal with subdivision. First, no edge can be shared by more than two faces. Secondly, the faces sharing a vertex can be ordered such that every two consecutive faces also share an edge. Thirdly, each face

has an orientation and faces sharing an edge must have same orientation.

The connectivity of the mesh should be represented easily since these changes often during subdivision when new additional vertices are introduced and therefore a new connectivity structure has been generated. A good implementation of connectivity also ensures easy traversal of the data structure. This is again important in our implementation since we recur over the mesh when subdividing.

We have chosen to focus less on other interesting and important areas of a good data structure such as computational cost and memory consumption, the reason being that today's computers can rather effortlessly support large data sets. Nevertheless, our data structure still ensures that basic traversal of the mesh is reasonable efficient, but there is certainly room for improvement in other areas of our implementation.

4.1.1 The Corner Lath Data Structure

We use the *Corner Lath* data structure proposed by Legakis et al. [11]. This data structure is based on the *Lath* concept, illustrated in Figure 4.1. First, a link to the vertex information associated with *Lath*; second, a link (the *face-clockwise* link) to a lath that represents the next lath in a clockwise traversal of laths of the face that *Lath* represents; and third, a link (the *vertex-clockwise* link) to the lath that is the next lath in a clockwise traversal of the vertex that *Lath* represents are established. Here, the lath elements are easily identified with each vertex-face pair. The edge identified with each lath *Lath* is that edge bounded by the vertex of *Lath* and the vertex of the lath identified by the *face-clockwise* link.

This data item can encapsulate the structure of the mesh, and is fairly easy to implement. The lath is used to represent all the important element of a mesh namely the vertices, edges, faces and the connectivity among them. Furthermore the lath supports the development of a basic set of traversal functions that are used during subdivision.

The article describes several variations of the lath. The one we choose is the corner lath. This type of lath, like the others, is associated with exactly one vertex, one edge and one face of the mesh. Each lath contains a link to the vertex, one to the lath

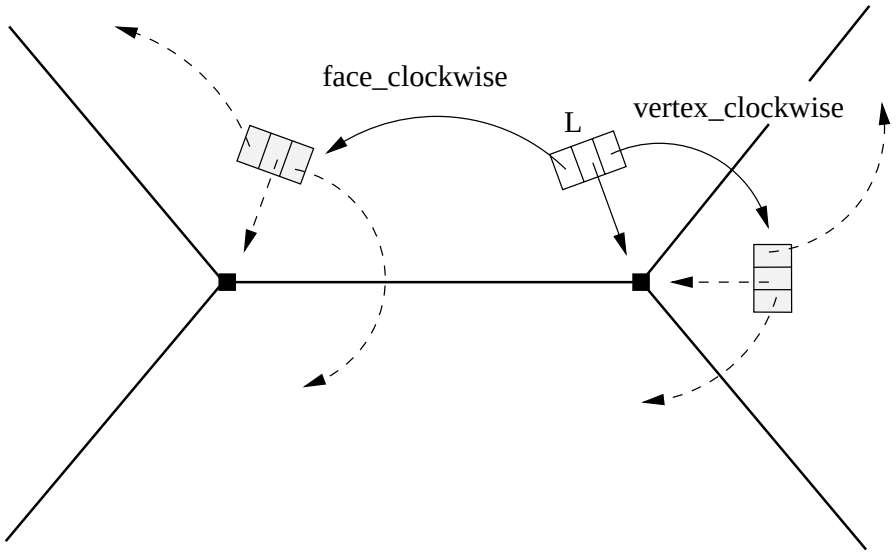


Figure 4.1: The lath data element.

contained in the same face but in the counter-clockwise direction and one to the next lath contained in a counter-clockwise traversal of the associated vertex. A visualization of the corner lath data structure is shown in Figure 4.2. The two things that must be stored are the laths and the vertices. The vertices contain the coordinates of the associated control points.

By focusing on a large portion of the mesh two loops formed by the lath pointers are easily identified. The first loop allows a counter-clockwise traversal of the laths in a face and the second loop allows a counter-clockwise traversal of the laths around a vertex. Applying simple functions that use the lath pointers to navigate it is possible to traverse these loops. Given a lath **L** , the basic functions are,

- **L.ccwv()** — returns the lath that follows **L** in a counter-clockwise traversal around the vertex that **L** points to.
- **L.ccwf()** — returns the lath that follows **L** in a counter-clockwise traversal of the face that contains **L** .
- **L.cwv()** — returns the lath that follows **L** in a clockwise traversal around the vertex that **L** points to. This can be computed either by using **L.ccwv()** until

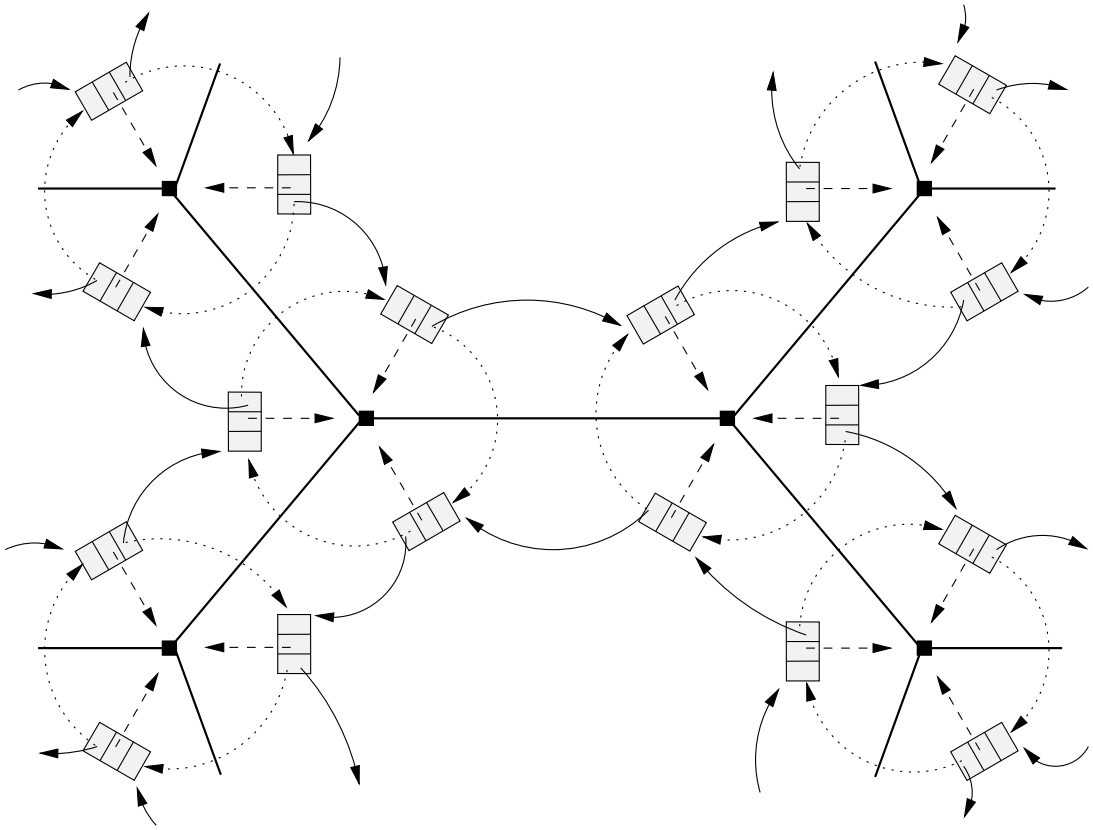


Figure 4.2: Corner lath data structure pointer loops. The continuous arrows show the loop on a face while the dotted ones show the loop around the vertex.

the lath **L.ccwv()** equals **L**, or by using the sequence **L.ccwf().ccwv().ccwf()**. Special considerations have to be taken when working on the boundary.

- **L.cwf()**— returns the lath that follows **L** in a clockwise traversal around the face that contains **L** . This can be computed either by using **L.ccwf()** until the lath **L.ccwf()** equals **L**, or by using the sequence **L.ccwv().ccwf().ccwv()**. On the boundary the second option will not work.

On the boundary, we should devote special care when traversing around a vertex or in other words when switching from one face to another. The destination face might not exist so this has to be clearly identified in the data structure. By storing a *NULL* pointer in the vertex link of a lath, and checking whether such one exists or not, when traversing the data structure.

4.1.2 Implementing the Data Structure

Our implementation of the corner lath data structure is contained in a class. The basic functions discussed above are implemented just described as member functions of the class. Here is an outline of certain details differing form the above discussion.

During subdivision reversal from and to different levels arise. Consequently, a very simple but an important extension to our implementation of the corner lath structure is needed to connect the subdivision levels that arise during subdivision. Each lath therefore has a pointer to a lath in the next subdivided level, which is accessed by the function

- **L.nextlevel()**— returns the lath in the next subdivided level

To handle the non-uniform subdivision schemes discussed in Section 3.1 and 3.2 we need a special tagging of the vertices and edges. Biermann et. al. [2] also used tags in the same manner as we did for modeling sharp features. For this reason, we added two member variables to the lath class to store two knot spacings for the vertex it points to. One knot spacing is associated with the edge between the vertex and its counter-clockwise neighbor in the face, and the other knot spacing is associated with the edge between the vertex and its clockwise neighbor in the face.

Recall that each edge has two knot spacings associated with it – one in each end of the edge. Table 4.1 lists the available knot spacing values and their interpretation. If a point has conflicting markings e.g. it is marked as both a conical and cusp point, the rule used to, is determined by the precedence: corner, conical, cusp, dart/ordinary.

4.2 Implementation of Subdivision Schemes

This will describe the algorithms we have implemented in order to support each of the schemes. The main ideas behind each of the implementations will be mentioned here. Each of the schemes are implemented as a class that all inherited from a base class, which enables us to switch methods at run time.

Knot spacing value	Interpretation
0.0	Crease edge. The two knot spacings on the edge should have the same value. A point is a dart if exactly one edge emanating from its corresponding vertex is marked as a crease. The dart vertex rule is identical to the same value.
1.0	Ordinary edge. The two knot spacings on the edge should have the same value.
2.0	Conical point. A point is conical if at least one edge emanating from its corresponding vertex is marked as conical.
3.0	Cusp point. A point is cusp if at least one edge emanating from its corresponding vertex is marked as cusp.
4.0	Corner point. A point is corner if at least one edge emanating from its corresponding vertex is marked as corner. or if there are more creases are incident upon a vertex. The edge rule will treat edges marked as corners as though they were marked as creases.

Table 4.1: Special knot spacing values for marking sharp features which are used in MDL file format.

4.2.1 $\sqrt{3}$-Subdivision Scheme

Subdivision

The $\sqrt{3}$-subdivision works on triangular meshes and the first assumption of the scheme is that it is applied to meshes consisting of entirely triangles. The implemented scheme extends Kobbelt's [10] scheme with the sharp features such as crease edge, or corner, cusp or dart points. Again the knot spacing markings are used in here as explained in Section 4.1.2.

The $\sqrt{3}$-subdivision process is implemented in the same recursive style as the other schemes. The recursion, traverses vertices in the mesh building the subdivided 1-ring neighbourhood of each vertex, and connecting these at the end of recursion.

The order of events when processing a single vertex is outlined below and illustrated in Figure 4.3.

1. If the vertex has already been processed, nothing is done.
2. It is checked whether the vertex is on the boundary of the mesh. If so pointers to the clockwise most and counter-clockwise most laths are saved.
3. The control points of the neighbor vertices are checked and the knot spacings are inspected to determine the type of vertex rule to use.
4. The new vertex point is calculated by selecting the appropriate vertex rule based on the findings in the previous step.
5. A new point is calculated for each triangle adjacent to the vertex. The midpoints of the triangles are calculated as the average of the corner vertices of the triangle.
6. If the triangle has a counter-clockwise mate with respect to the vertex being processed edge flipping is done. If not or if these triangles are marked as crease then the boundary rule is applied.
7. If the subdivision level being processed is an odd level the knots pacing on the counter-clockwise vertex of the triangle and if it is on the boundary or a crease edge then boundary rule is applied.
8. The triangles comprising the 1-neighbourhood of the start vertex on the subdivided level are constructed and connected. The knot spacings on the original edges are inherited by the corresponding edges on the subdivided level.
9. Connections from the coarser mesh to the newly generated laths in the next level are made.
10. The subdivision procedure is applied recursively to all the neighboring vertices.
11. The final step is making the connections between the newly generated triangles with the corresponding one of the old vertices.

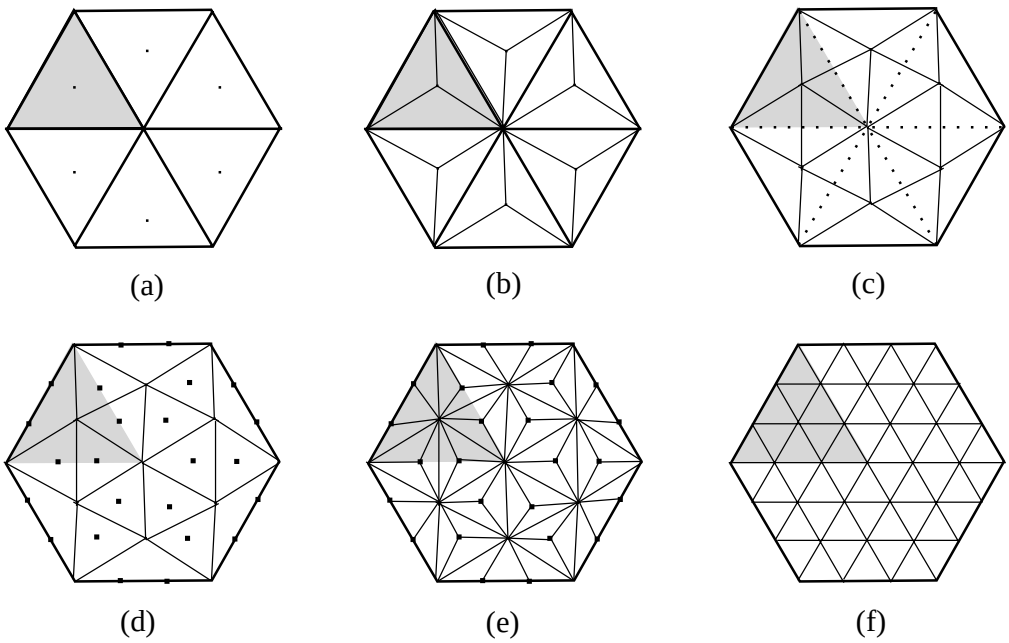


Figure 4.3: Illustration of $\sqrt{3}$-subdivision, upper row odd and bottom row even subdivision steps. (a)new face vertices, (b) new faces (c) old edges flipped, (d) new face and boundary vertices, (e) new faces, and (f) old edges flipped.

Normals

When $\sqrt{3}$-subdivision surfaces with creases and special points are visualized, the use of interpolated polygonal normals is unsatisfactory as in the case of Loop, since this will tend to smooth out these features. To get the correct visual appearance, the exact normals of the limit surface are used instead. The equations for calculating the exact normals were given in Section 3.1.2.

4.2.2 Loop Subdivision Scheme

Subdivision

The Loop subdivision scheme works on triangular meshes so the first assumption for the implementation of the scheme is that it is applied to meshes consisting entirely triangles. If this assumption is not satisfied for a model, it must be triangulated before subdividing.

We implemented extended Loop [9] scheme with facilities for creating crease edges, cusp, conical, corner and dart points as discussed in Section 3.2. These features are marked in the data structure as special knot spacings.

The Loop scheme subdivision process is implemented in the same recursive fashion as the other schemes. The recursion traverses vertices in the mesh, building the subdivided 1-neighbourhood of each vertex, and connecting these at the end of the recursion.

The order of events when processing a single vertex is outlined below and illustrated in Figure 4.4.

1. If the vertex has already been processed, nothing is done.
2. It is checked whether the vertex is on the boundary of the mesh. If so pointers to the clockwise most and counter-clockwise most laths are saved and regular valence for the vertex is recorded as 4.
3. The control points of the neighbor vertices are collected and the knot spacings are inspected to determine the type of vertex rule to use.
4. The new vertex point is calculated by selecting the appropriate vertex rule based on the findings in the previous step.
5. A new point is calculated for each edge emanating from the vertex. The vertex at the opposite end of the edge is inspected to find its valence and check whether it is a corner. The edge rule is selected based on the findings.
6. The triangles comprising the 1-neighbourhood of the start vertex on the subdivided levels are constructed and connected. The knot spacings on the original edges are inherited by the corresponding edges on the subdivision level.
7. The subdivision procedure is applied recursively to all the neighboring vertices.
8. The 1-neighbourhood triangle patches are connected within every old triangle. These are the triangles at the center of the original triangles.

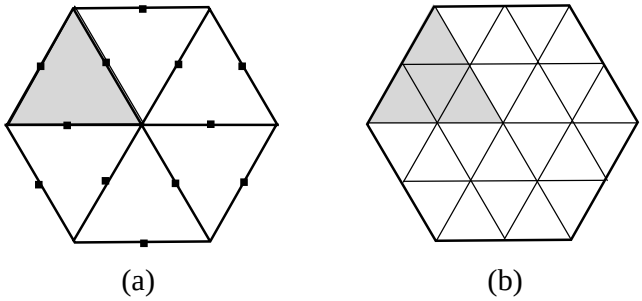


Figure 4.4: Illustration of Loop subdivision. (a) new edge vertices, and (b) new faces.

Normals

When Loop surfaces with creases and special points are visualized, the use of interpolated polygonal normals is unsatisfactory since this will tend to smooth out these features. To get the correct visual appearance, the exact normals of the limit surface are used instead. The equations for calculating the exact normals were given in Section 3.2.1.

4.2.3 Doo-Sabin Subdivision Scheme

Subdivision

As mentioned in Section 3.3 the Doo-Sabin scheme is relatively easy to use and implement. There is essentially only one mask that is applied to all faces in the control mesh. The subdivision process is illustrated in Figure 4.5 and includes the following steps.

1. If the vertex has already been processed, nothing is done.
2. The first step in the refinement calculates the new face vertices. In general it takes each vertex of each face in the mesh and applies the mask shown in Figure 3.6 in Section 3.3 to calculate the new vertex. (See Figure 4.5(a)).
3. Then the generated vertices are connected to form the new faces inside the old faces. (See Figure 4.5(b)).

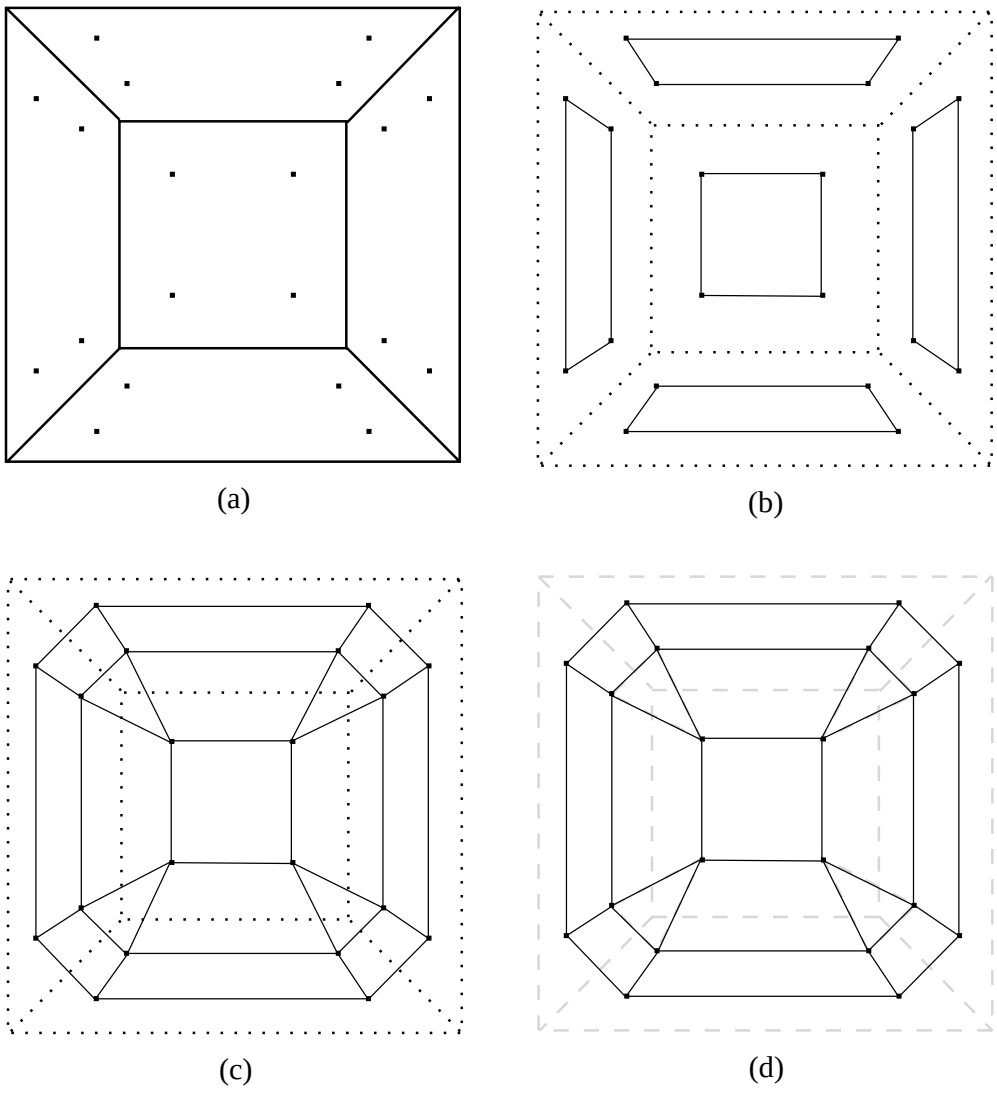


Figure 4.5: Illustration of Doo-Sabin subdivision. (a) new vertices are calculated, (b) new faces are constructed, (c) the faces across the edges are connected, and (d) the faces around faces are connected.

4. Connections from the coarser mesh to the newly created laths in the next level are made.

5. The subdivision procedure is applied recursively to all the neighboring faces.

6. New faces are constructed for each edge in the old face by connecting the four new vertices adjacent to an old edge. The new faces are connected to the faces constructed in step 3. (See Figure 4.5(c)).

7. The final step is to construct a new face for each vertex in the old mesh by connecting the new vertices adjacent to each old vertex. These new faces are connected to the faces constructed in steps 3 and 6. (See Figure 4.5(d)).

Normals

We did not implement the exact normals of the limit surface as in Loop or $\sqrt{3}$-subdivision. Instead, we use interpolated polygonal normals on each vertex.

4.2.4 Catmull Clark Scheme

Subdivision

The procedure for subdividing a mesh with the Catmull-Clark masks is a little more complicated than the case of Doo-Sabin. This is because a 2-neighborhood is necessary to calculate the new points. There are three types of new points: face points, edge points, and vertex points. The newly generated points then have to be connected in a proper way. the following shows the order of events in the Catmull-Clark scheme and the accompanying figures help to clarify each step. Our implementation can also work on irregular polygons as shown in Figure 4.6(a).

1. If the vertex has already been processed, nothing is done.

2. The new face points are calculated in this step. That is we circle our vertex and generate a new face point for each face containing the vertex. The face point is the average of all the original points defining the face. If we hit the boundary and can not circle any further, we start the recursive subdivision here on the next counter-clockwise vertex. (See Figure 4.6(a)).

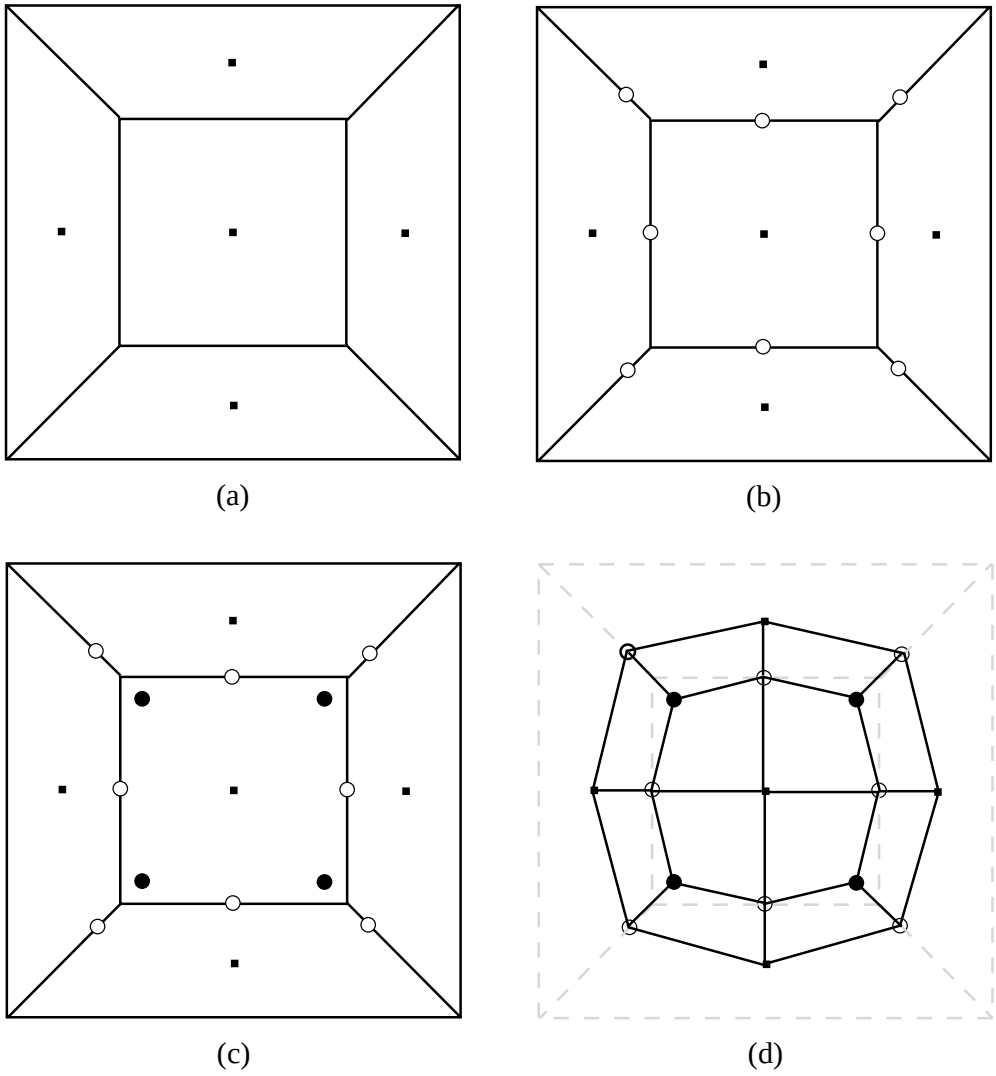


Figure 4.6: Illustration of Catmull-Clark subdivision. (a) new face points calculated, (b) new edge points calculated, (c) new vertex points calculated, and (d) new points are connected to construct the new faces.

3. The new edge points are calculated by averaging the end points of the edge and the newly computed face points in the two faces sharing the edge. The end points are placed on level j and the newly created face points are on level $j+1$; so we are using two levels. (See Figure 4.6(b)).

4. The new center points can be calculated in different ways but we use the formula $\frac{F}{n}+\frac{2E}{n}+\frac{(n-3)V}{n}$, where F is the average of the new face points of all faces adjacent to old vertex V. E is the average of the midpoints of all edges incident upon the old vertex V, and n is the valency of V (See Figure 4.6(c)).

5. The faces surrounding the newly created center vertex are created by using the new edge and face vertices. Each face consist of the new center vertex, the two new adjacent edge vertices and the face vertex created in each face.

6. The subdivision procedure is recursively applied to all the neighboring vertices.

7. The new faces around each vertex are connected to each other as a final step. The result of the scheme is seen in Figure 4.6(d). It should be noted that the refined mesh only contains faces with four edges regardless of the structure of the original mesh.

Normals

We did not implement the exact normals of the limit surface as in Loop or $\sqrt{3}$-subdivision. Instead, we use interpolated polygonal normals on each vertex.

Chapter 5

Subdivision Surfaces Examples and Analysis

This chapter compares discussed subdivision schemes by applying to a variety of meshes and examining images of example surfaces rendered. Smoothness of the surfaces produced, computational work to be done, and modeling the sharp features can be thought of the areas of comparison. Smoothness of the surfaces and computational cost comparison done for considered four subdivision schemes. Instead, we compare the sharp features only for the triangular subdivision schemes – Loop, $\sqrt{3}$-subdivision –, since we just implemented sharp features with these schemes.

5.1 Comparing Smoothness of Produced Surfaces

Figure 5.1 shows surfaces generated by the $\sqrt{3}$-subdivision, Loop, Doo-Sabin and Catmull-Clark subdivision schemes from a common control mesh in the shape of a cube. It can be easily distinguished that Catmull-Clark and Doo-Sabin produces more pleasing surfaces than that of produced by Loop or $\sqrt{3}$-subdivision which are more asymmetric. The reason for this is that the cube had to be triangulated first before applying these schemes. The triangulation has some implications; firstly it raises the valency of the original vertices, possibly making them irregular, secondly the new vertices inserted will be irregular unless the face is a hexagon, and thirdly, the triangles

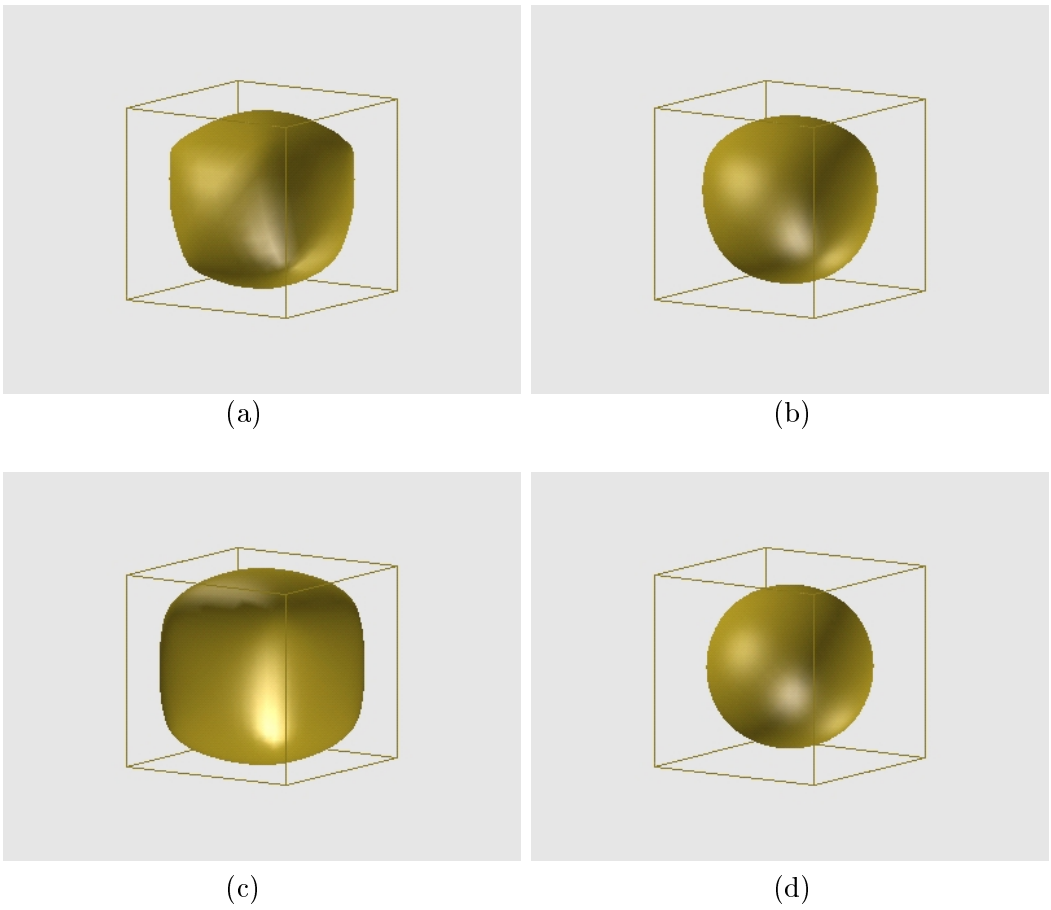


Figure 5.1: Results of applying various subdivision schemes to a cube. (a) $\sqrt{3}$-subdivision, (b) Loop subdivision, (c) Doo-Sabin subdivision, and (d) Catmull-Clark subdivision. The control mesh is the unit cube drawn in wire frame.

will be flat with a very small angle at one side while having a large one on the other side. Ideally, the triangulation should not increase the number of irregular features and all triangles should have approximately equal angels.

Figure 5.2 shows surfaces generated by the schemes from a common control mesh in the shape of a tetrahedron. Notice how much the mesh shrinks under Loop, Figure 5.2(b). Also a noticeable shrinkage can be seen in $\sqrt{3}$-subdivision, Figure 5.2(a), and Catmull-Clark, Figure 5.2(d), but slightly less than the Loop scheme. All the cube and tetrahedron figures show that, the shrinkage is a characteristic feature of approximating schemes. Shrinkage increases while the number of the polygons in the

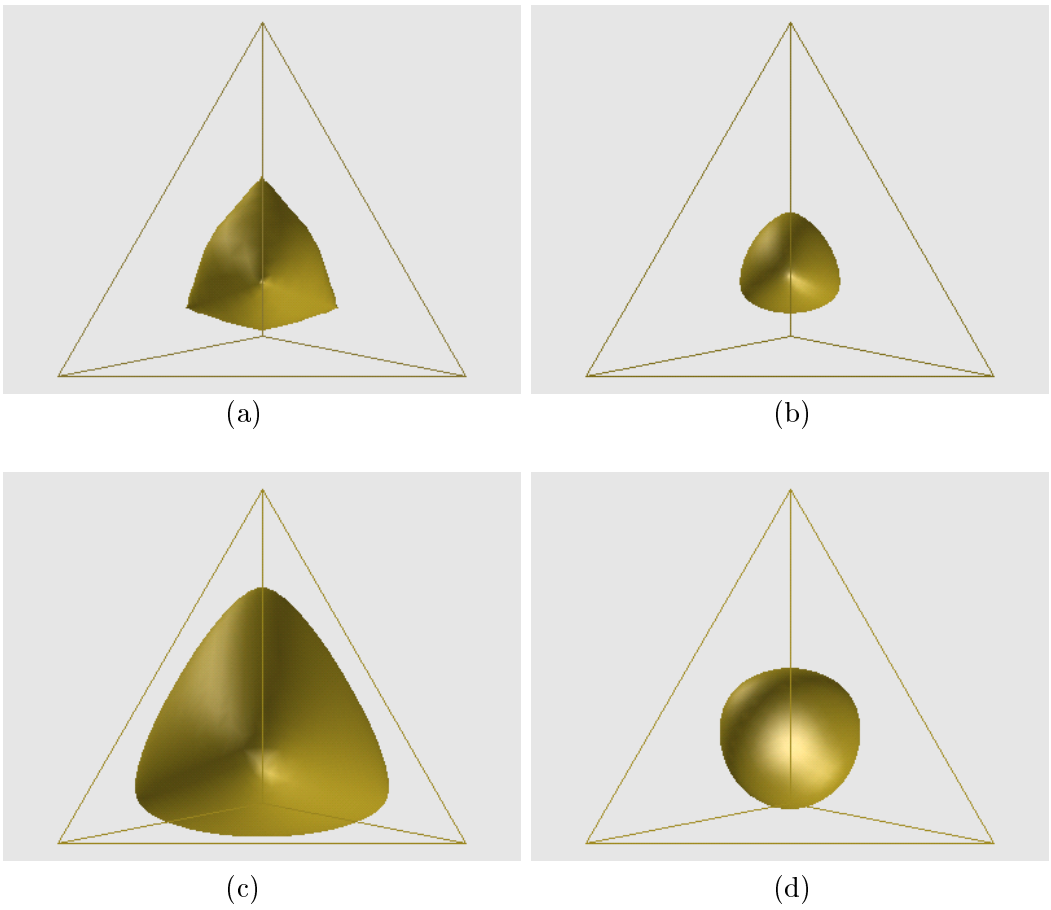
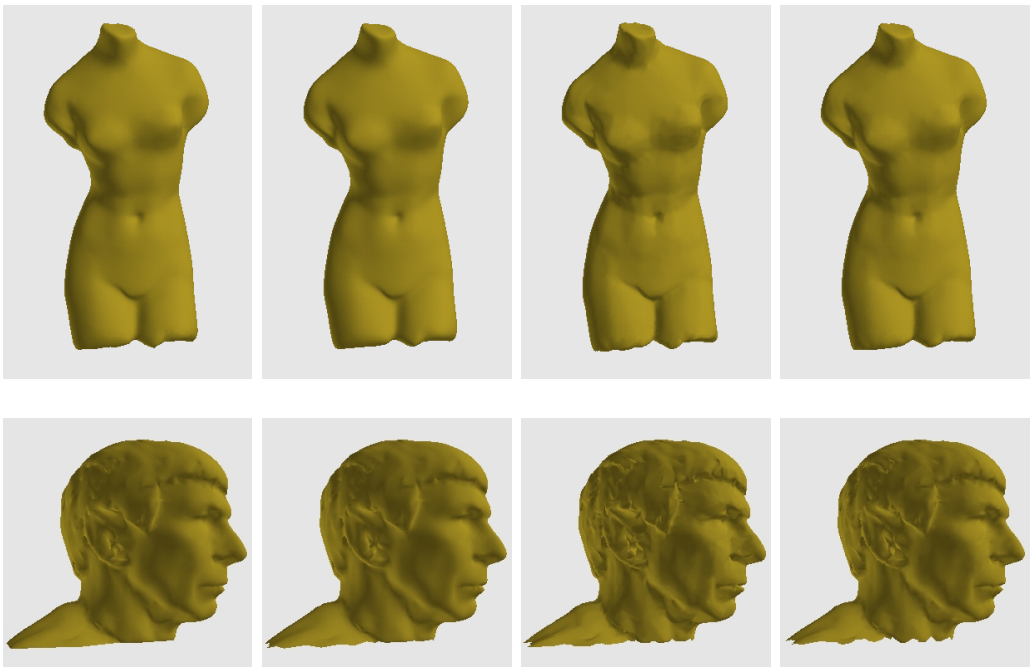


Figure 5.2: Results of applying various subdivision schemes to a tetrahedron. (a) $\sqrt{3}$-subdivision, (b) Loop subdivision, (c) Doo-Sabin subdivision, and (d) Catmull-Clark subdivision. The control mesh is the tetrahedron drawn in wire frame.

coarse mesh decreases, that is for small meshes, the resulting surface is likely to occupy much smaller volume than the original control mesh. Besides, Figure 5.3 shows that for sufficiently smooth coarse meshes, different schemes produce virtually undistinguishable results. Upper row of the Figure 5.3 illustrates the surfaces produced from the same coarse mesh of Venus sculpture and the bottom row illustrates the surfaces produced from the same coarse mesh of head of Mr. Spock by the algorithms we had implemented. A careful analysis will show that, there are little differences between the outputs. Another thing to notice in Figure 5.2 is that the shape of the control polyhedron can be recognized in the Doo-Sabin surface while getting less degree in the Loop and $\sqrt{3}$-subdivision surfaces, while Catmull-Clark bears no resemblance.



(a) (b) (c) (d)

Figure 5.3: Applying various schemes to a sufficiently smooth mesh produce similar results. (a) $\sqrt{3}$-subdivision, (b) Loop, (c) Doo-Sabin, and (d) Catmull-Clark.

5.2 Comparing Computational Cost of the Schemes

Performance of different subdivision schemes can be evaluated by examining the number of the polygons computed per time unit. The absolute values are naturally highly dependent on the specifications of the computer used. We have done our experiments on a 800 Mhz. Pentium III double processor system with 1 GB RAM and a GForce 3 Video Adapter with OpenGL support, running Linux OS. Sample measurements showing the number of polygons processed per second are listed in Table 5.1.

Figure 5.5 shows the results of applying $\sqrt{3}$-subdivision and Loop schemes to a decimated Stanford bunny. The original Stanford bunny was decimated with Michael Garland's [6] `QSlim 2.0 surface simplification tool`. All the other models used in our experiments are also simplified with the same tool. The original Stanford Bunny and the simplified one are shown in Figure 5.5(a) and (b) respectively. It should be noted in Figure 5.5(c),(d),(e) and (f) that how $\sqrt{3}$-subdivision scheme produces

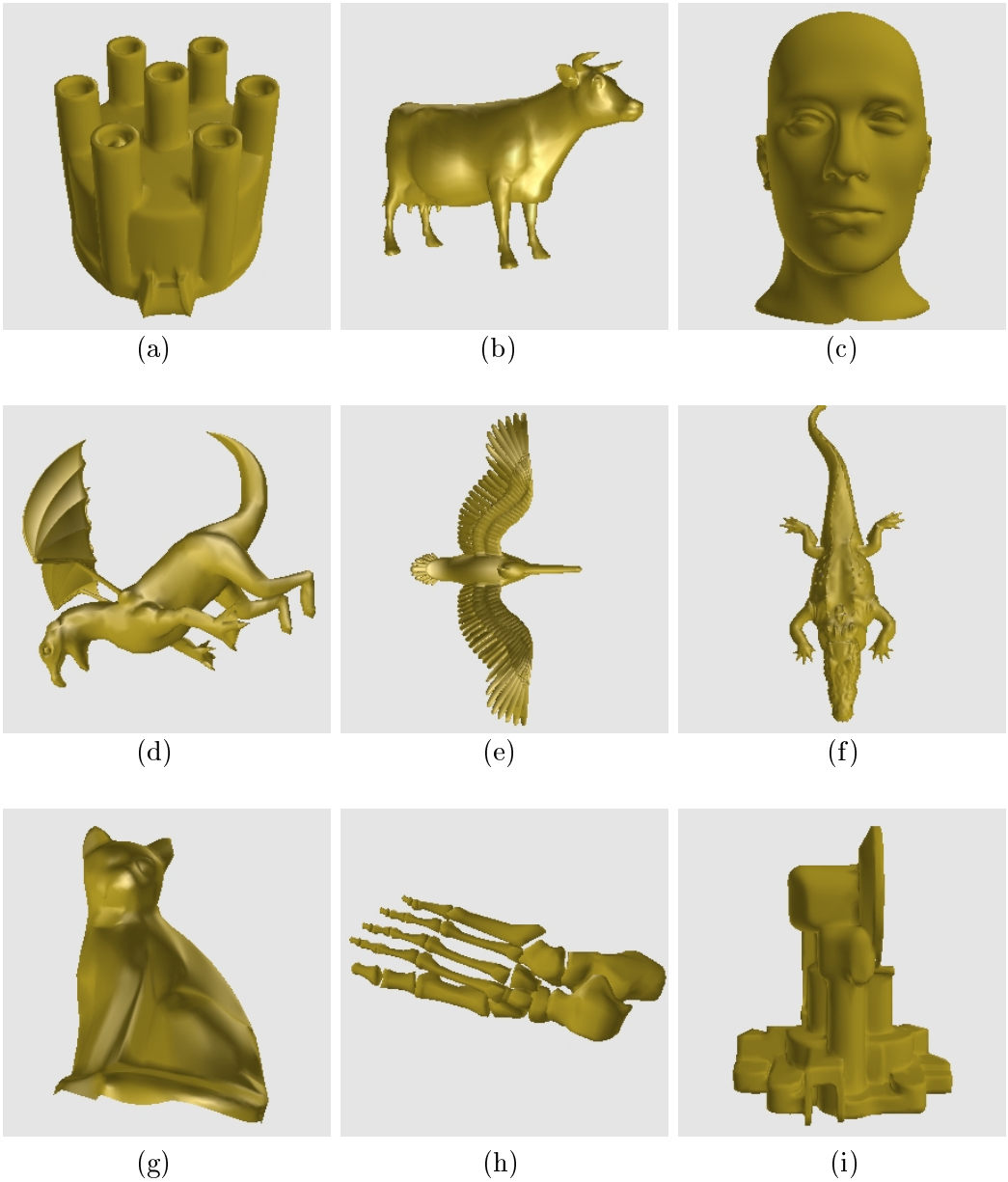


Figure 5.4: Models which are used in our experiments. (a) distcap, (b) cow, (c) mannequin, (d) dragon, (e) pelican, (f) crocodile, (g) cat, (h) bones, and (i) oilpump.

MESHES	$\sqrt{3}$-Subdiv.	Loop	Doo-Sabin	Catm.-Clark
Mannequin(34,567)	58.736,8	86.260,9	61.347,9	50.207,0
Oil-Pump(20,544)	59.328,0	83.638,6	59.671,4	48.391,3
Bunny(69,451)	60.391,3	82.578,5	58.936,4	49.248,3
Cat(8,976)	61.072,7	81.118,5	59.736,4	51.521,4
Dragon(9,832)	68.945,0	90.074,2	61.983,0	48.717,3
Pelican(25,080)	63.319,3	82.119,2	63.580,0	50.508,6
Bones(4,204)	59.327,4	84.925,2	61.936,7	49.729,4
Dist-Cup(14,880)	58.554,2	86.084,7	59.385,7	46.596,8
Crocodile(34,404)	62.694,1	83.527,6	62.589,5	51.060,0
Crater(65,467)	58.521,4	83.129,7	58.156,0	46.983,4
Cow(5,804)	63.361,2	83.976,9	61.432,7	49.267,3

Table 5.1: Polygons processed per second for implemented different subdivision surface schemes.

approximately the same smoothness as the Loop subdivision scheme with a noticeable less number of triangles.

The reason for the comparatively high performance of the $\sqrt{3}$-subdivision and Loop schemes are that constructing triangles is faster than constructing quadrilaterals in our data structure. Besides, in the construction of the subdivided control mesh triangles generally have to be connected to three neighbors, while quadrilaterals have four neighbors. By the way, for a given number of vertices a triangular scheme will have twice as many polygons as a quadrilateral mesh. This makes triangular schemes slightly slower than the quadrilateral schemes in total computational time.

The number of the masks used also important from the point of performance. By the increase of the number of the masks work to be done to decide which mask to be use and the calculations for the masks gets complex and time consuming. There is a trade of between obtaining more smooth surfaces by using more masks and rules and computation time. The smaller masks can explain the superiority of the Doo-Sabin over Catmull-Clark scheme.

It should be emphasized that when we started our implementation, our aim was correctness rather than speed. A fully optimized implementation would yield substantially higher polygon frequencies.

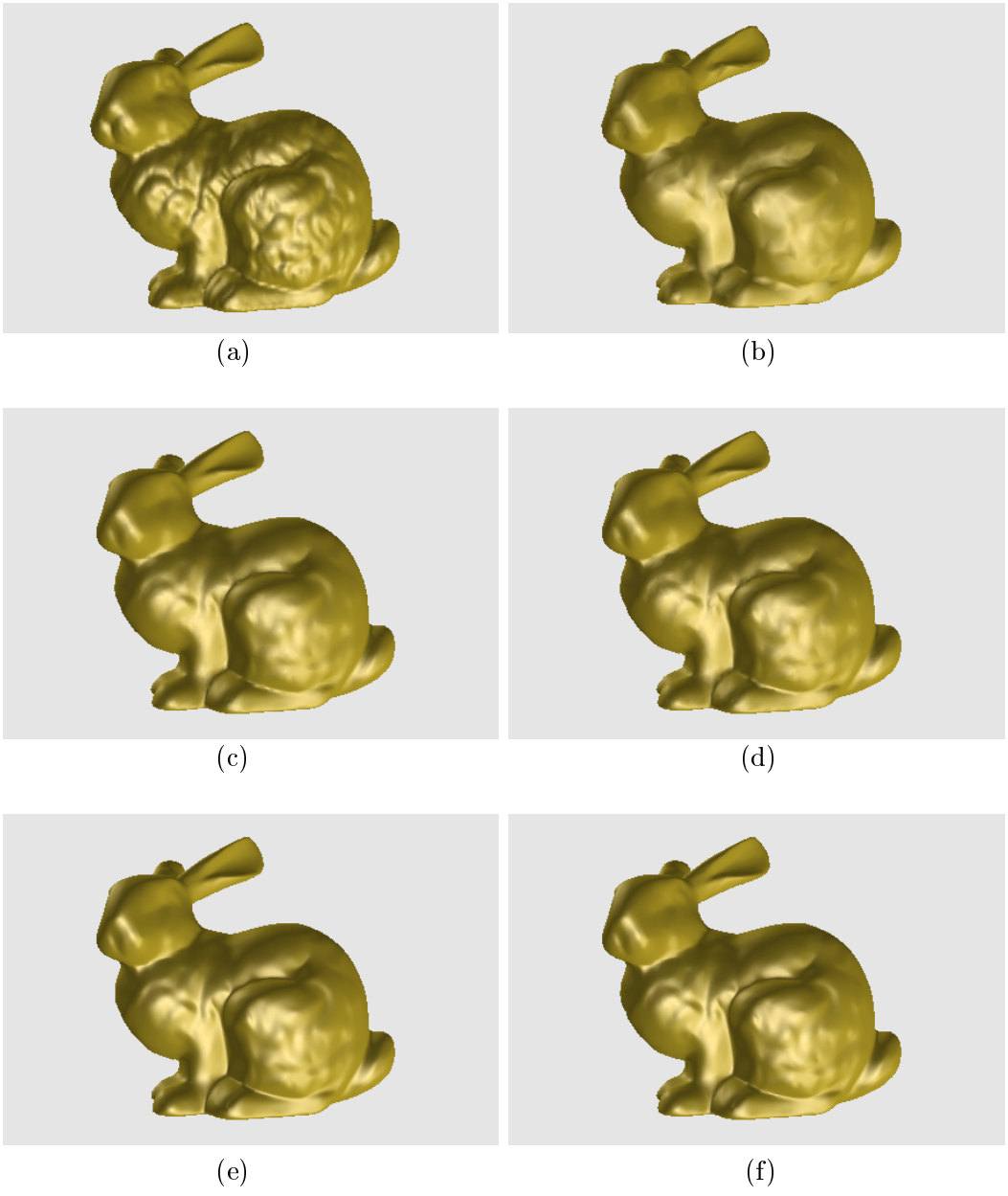


Figure 5.5: Applying the Loop and the $\sqrt{3}$-subdivision to a simplified bunny mesh. (a) original bunny(69,451) (b) decimated bunny(5,000), (c) after one Loop subdivision(20,000), (d) after one $\sqrt{3}$-subdivision (15,000), (e) after two Loop subdivision(80,000), and (f) after two $\sqrt{3}$-subdivision (45,000).

5.3 Comparing Sharp Features Produced by Triangular Schemes

Usually, it is necessary to have more control over the surface than it is offered by the uniform schemes. The most useful feature is to create creases and corners. One approach to modeling such features is to make the control mesh denser around the features. From the point of implementation this solution is not satisfactory, since it increases the amount of memory needed to store the control mesh and the time spent on subdividing the mesh. Another solution to this problem is using tagged meshes and introducing additional subdivision rules. By the way, introducing new rules makes the implementation more complicated.

The extended Loop scheme and extended $\sqrt{3}$-subdivision scheme has a very intuitive way of introducing creases and similar features, however they are less flexible, for example they can not produce semi-sharp features. This could be remedied by introducing additional subdivision rules and extending the tagging for the data structure. Obviously, introducing more rules makes the implementation more complicated and less efficient. Pixar Animation Studios use a variant of the Catmull-Clark with rules for creases [4]. They produced semi-sharp creases, first applying the sharp crease rule a number of times and then applying an interpolation of the sharp and smooth rule.

The results of using tagging by the extended $\sqrt{3}$-subdivision scheme and extended Loop scheme can be seen in Figure 5.6. The surfaces on the first and third rows are produced by extended $\sqrt{3}$-subdivision scheme while the surfaces on the second and fourth row are produced by extended Loop subdivision scheme. All the surfaces are generated from the same initial mesh given in Figure 5.6(a) as wireframe and Figure 5.6(b) as smooth shaded surface. Figure 5.6(c) and (d) shows the surfaces generated without using any knot spacings by $\sqrt{3}$-subdivision and Loop scheme respectively. The surfaces in Figure 5.6(e) and (f) are generated by marking the outer and inner side edges of the front face as crease. Notice the front face surface that is only smoothed along the sides. $\sqrt{3}$-subdivision shrinks more than Loop scheme while producing creases. The surfaces in Figure 5.6(g) and (h) are generated by marking the inner side vertices of the front face as corner vertices, while marking the outer side vertices of the front face as corner vertices produces the surfaces in Figure 5.6(i) and (j). Notice the sharpness of the surfaces along the creases between corner points. Figure 5.6(g) and (h) show

that, extended $\sqrt{3}$-subdivision scheme produces more realistic surfaces than that of extended Loop scheme, in the case of marking the inner side vertices as corner. In extended Loop subdivision the corner points goes deep into the surface. The surfaces in Figure 5.6(k) and (l) are generated by marking the bottom face outer side vertices as corner vertices.

The surfaces in Figure 5.6 shows that by using the developed $\sqrt{3}$-subdivision and extended Loop subdivision we can control the behavior of the surface by using the specified taggings. This gives us the flexibility of obtaining different surfaces from the same coarse mesh and produce more realistic looking surfaces.

For the $\sqrt{3}$-subdivision and Loop subdivision we have implemented calculation of exact normals of the limit surfaces. Figure 5.7 illustrates the differences between using interpolated normals and exact normals. The same mesh rendered using both interpolating and exact normals. The right ones are rendered by using exact normals and the left ones with interpolating normals. The sharpness of the creases and corner points can not be distinguished clearly in the surfaces obtained by using interpolated normals.

The top row, Figure 5.7(a) and (b), hill figure, contains two creases. One is on the front hill from the bottom near corner to the top of the hill and the other one is between the hills. The ending point of the crease at the top of the hill is a dart point and the smoothness can be distinguished in both surfaces. High polygon frequency decreases the effect of using exact normals while rendering but a careful eye would catch the sharpness difference between the hills. On the middle row, Figure 5.7(c) and (d), a surface containing corner and crease rendered and the difference is very clear. The bottom row, Figure 5.7(e) and (f), also shows a surface with the sides of the inner hole are creases and seems smooth with the interpolated normals. With exact normals the sharpness of the creases is emphasized and can be distinguished clearly.

Additionally, the exact normals make the approximation to the surface look better, so it may be possible to save some subdivision steps.

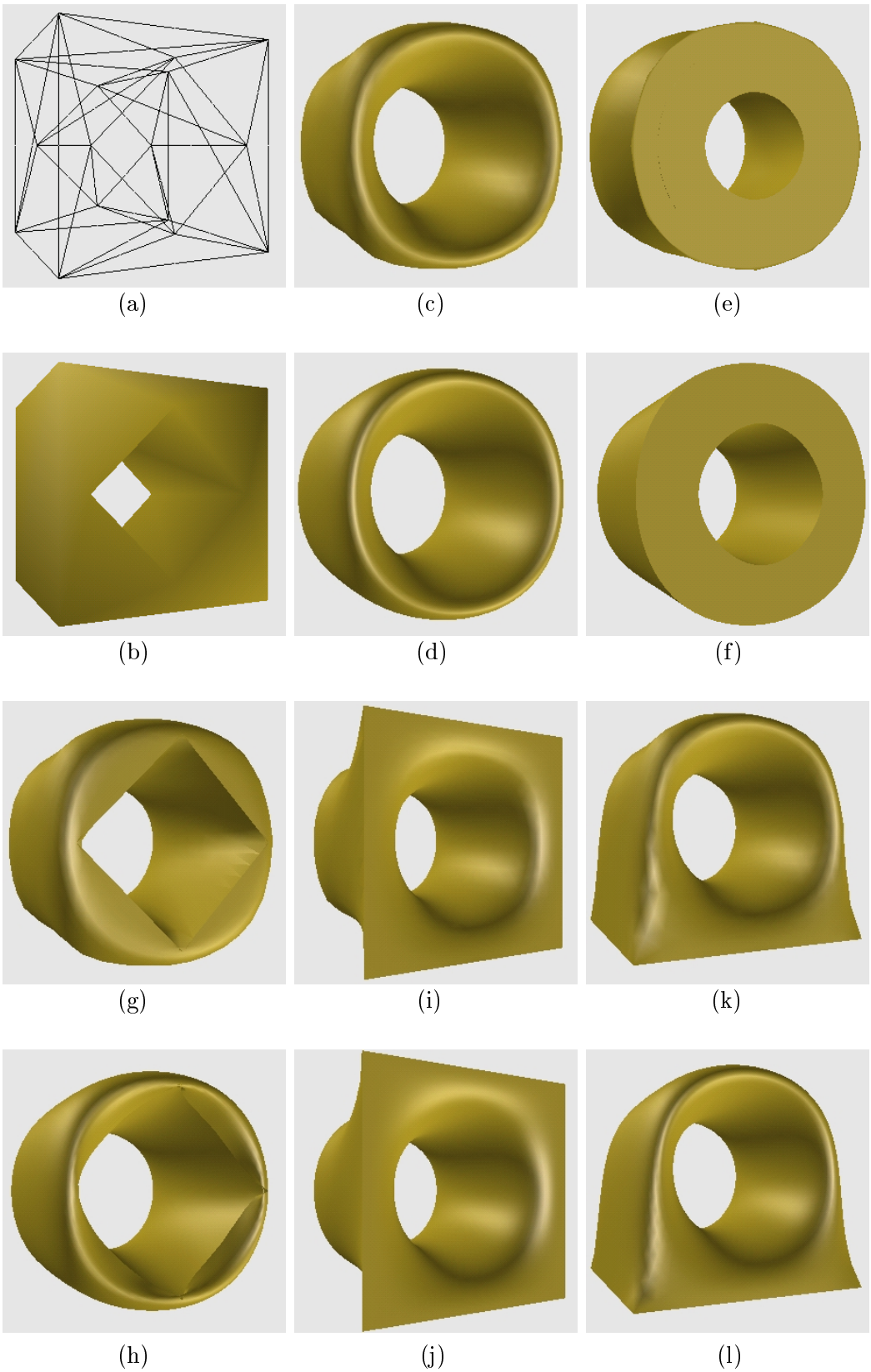
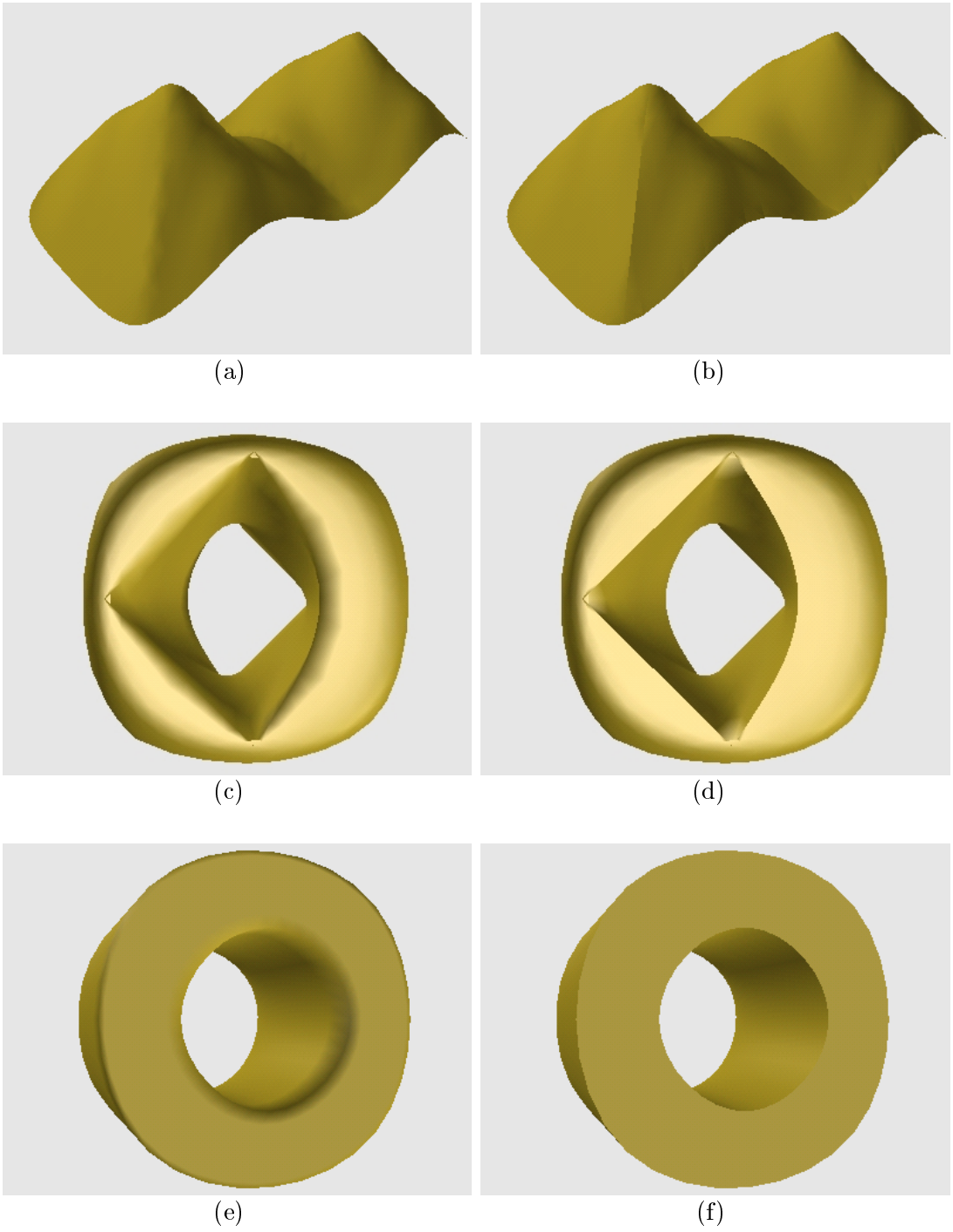


Figure 5.6: Example sharp features produced by using extended $\sqrt{3}$-subdivision scheme and extended Loop subdivision scheme. The surfaces on the first and third rows are produced by extended $\sqrt{3}$-subdivision scheme while the surfaces on the second and fourth row are produced by extended Loop subdivision scheme.



(a) (b)

(c) (d)

(e) (f)

Figure 5.7: Affect of using exact normals vs. interpolating normals. Surfaces on the left, (a), (c), (e) are rendered by using interpolated normals, and on the right, (b), (d), and (f) are rendered using exact normals.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Recently, subdivision surfaces has gained considerably interest in computer graphics research community. Mathematical concepts behind them are extended to produce limit surfaces required for many applications regardless of the topology. Support for the varying topology makes the subdivision surfaces very useful. The well-known spline surfaces have the major drawback of being limited by the topology of the surface.

We discuss the topological restrictions of B-spline based surface representations. Topological restrictions and computational difficulties of B-splines are mentioned and how subdivision surfaces eliminates these disadvantages are discussed. The subdivision schemes generalize spline surfaces to arbitrary topology by specifying additional masks for irregular vertices and faces in the control mesh. Then, four different subdivision schemes that we choose to implement, namely Doo-Sabin, Catmull-Clark, Loop and $\sqrt{3}$-subdivision, are briefly explained.

Next, we discuss our implementation of these four subdivision schemes. Since the corner-lath data structure handles various mesh topologies, we choose this data structure. It allows us to traverse a mesh in an intuitive and efficient manner based on the mesh topology. Then, we briefly explain the implementation of discussed subdivision schemes in a step by step manner.

Then, the implemented subdivision surface schemes are compared. Discussed subdivision schemes are compared by generating a variety of meshes using them and examining images of example surfaces rendered. We highlight certain properties of the schemes by examining a number of surfaces rendered using the implemented *Interface* program.

Briefly, the advantages of subdivision surfaces are: Subdivision surfaces are simple methods for describing complex surfaces. They are constructed easily through recursive splitting and averaging: splitting involves creating new faces by removing old ones and averaging involves taking a weighted average of neighboring vertices for the new vertices. They are easy to implement and efficient to execute. The well defined surface normals allows them to be rendered smoothly without the faceted look of low polygon count polygonal geometry. They can represent smooth surfaces with arbitrary topology (with holes or boundaries). Subdivision surfaces are defined algorithmically and they guaranties a smoothness degree. Subdivision naturally accommodates level-of-details control through adaptive subdivision because of its recursive nature. The difficulties faced with subdivision are intuitive specification, parameterization of the surfaces, and getting intersections of surfaces or attaching them together.

As a result of our experiments, we can say, it is not possible to appoint the best subdivision scheme. For a given application the choice of the scheme must depend on the type of the control mesh, and requirement for smoothness and performance. For applications requiring higher speed but low surface quality, the Doo-Sabin scheme can be a good choice. In some applications Catmull-Clark and Loop schemes can be candidates with their C^2 continuity on regular meshes. The slower increase of the mesh complexity and the suitability for adaptive refinement makes $\sqrt{3}$-subdivision scheme a good candidate for practical and industrial applications.

In the computer aided geometric design community, the popularity of the subdivision surfaces increases rapidly to make subdivision surfaces one of the modeling primitives.

6.2 Future Work

Although, many researches and studies done to find out the properties and behaviour of subdivision surface schemes, still there are many questions and properties to explore. One of the search topic can be the modification of the smoothing rules to lead to more smooth surfaces especially for the newly proposed schemes. Another search topic can be finding out a suitable data structure that could handle the large and constantly changing data sets involved in subdivision, keeping the storage, traverse and data access time minimum.

Detecting and processing sharp features of surfaces offers new flexibilities in modeling. Automatic detection of the topological type of the surface, and the presence and location of sharp features for the arbitrary meshes is still an open question and can be studied in detail. In addition to our extensions to $\sqrt{3}$-subdivision scheme, new subdivision rules can be developed to model wider variety of sharp features such as, conical or cusp points or more then two darts ending at a vertex, etc.

Obtaining semi-sharp features, while modeling an object with subdivision, is also an open question to explore for all subdivision schemes. Some methods proposed use interpolation of sharp and non-sharp state of surfaces. Finding out efficient and practical ways to produce semi-sharp and sharp features for subdivision schemes would probably make these modeling tools one of the modeling primitives.

Bibliography

[1] Ball, A.A. and Story, D.J.T., Conditions for tangent plane continuity over recursively generated B-spline surfaces. *ACM Transactions on Graphics* , Vol. 7, No. 2, pp. 83-102, 1998.

[2] Biermann, H., Levin, A. and Zorin, D., Piecewise smooth subdivision surfaces with normal control. *ACM Computer Graphics (SIGGRAPH'2000 Proceedings)* pp. 113-120, 2000.

[3] Catmull, E. and Clark, J., Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, Vol. 10, No. 6, pp. 350-355, 1978.

[4] Derose, T., Kass, M., and Troung, T., Subdivision surfaces in character animation. *ACM Computer Graphics (SIGGRAPH'98 Proceedings)*, pp. 85-94, 1998.

[5] Doo, D. and Sabin, M., Behaviour of recursive subdivision surfaces near extraordinary points. *Computer Aided Design*, Vol. 10, No. 6, pp. 356-360, 1978.

[6] Garland, M. and Heckbert, P.S., Surface simplification using quadric error metrics. *ACM Computer Graphics (SIGGRAPH'97 Proceedings)*, pp. 44-52, 1997.

[7] Halstead, M., Kass, M., and Derose, T., Efficient fair interpolation using Catmull-Clark surfaces. *ACM Computer Graphics (SIGGRAPH'93 Proceedings)*, pp. 35-44, 1993.

[8] Havemann, S., Interactive rendering of Catmull/Clark surfaces with crease edges. *The Visual Computer*, Vol. 18, pp. 286-298, 2002.

[9] Hoppe, H., Derose, T., Ducchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., and Stuetzle, W., Piecewise smooth subdivision surface reconstruction. *ACM Computer Graphics (SIGGRAPH'94 Proceedings)*, pp. 295-302, 1994.

[10] Kobbelt, L., $\sqrt{3}$-Subdivision. *ACM Computer Graphics (SIGGRAPH'2000 Proceedings)*, pp. 103-112, 2000.

[11] Legakis, J., MacKraken, R., and Joy, K.I., Data structures for unstructured meshes. Technical Report, CSE-97-6, Department of Computer Science, University of California, Davis, 1997, (available at `http://www.cs.umd.edu/class/fall2000/cmsc725/mesh-paper.pdf`).

[12] Loop, C., Smooth subdivision surfaces based on triangles. Master Thesis, *Department of Mathematics, University of Utah*, 1987.

[13] Loop, C., Bounded curvature triangle mesh subdivision with convex hull property *The Visual Computer*, Vol. 18, pp. 316-325, 2002.

[14] Peters, J. and Reif, U., Analysis of algorithms generalizing B-spline subdivision. *SIAM Journal on Numerical Analysis*, Vol. 35, No. 2, pp. 728-748, 1998.

[15] Schweitzer, J., Analysis and application of subdivision surfaces. Ph.D. Thesis, *Department of Computer Sciences and Engineering, University of Washington*, 1996.

[16] Sederberg, T. W., Zheng, J., Sewel, D., and Sabin, M., Non-uniform recursive subdivision surfaces. *ACM Computer Graphics (SIGGRAPH'98 Proceedings)*, pp. 387-394, 1998.

[17] Stam, J., Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. *ACM Computer Graphics (SIGGRAPH'98 Proceedings)*, pp. 395-404, 1998.

[18] Stam, J., On subdivision schemes generalizing uniform B-spline surfaces of arbitrary degree *Computer Aided geometric Design*, Vol. 18, pp. 383-396, 2002.

[19] Warren, J. and Weimer, H., *Subdivision methods for geometric design: A constructive approach. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling*, October 2001.

[20] Zorin, D., Stationary subdivision and multiresolution surface representation. Ph.D. Thesis, *California Institute of Technology*, 1997.

[21] Zorin, D., Schroder, P. Subdivision for modeling and animation. *ACM Computer Graphics SIGGRAPH Course Notes*, 2000.

[22] Zorin, D., Evaluation of piecewise smooth subdivision surfaces *The Visual Computer*, Vol. 18, pp. 299-315, 2002.

Appendix A

B-spline Curves and Surfaces

Splines are piecewise polynomial curves of some chosen degree. In this chapter, we define the uniform B-spline basis functions and use them to construct piecewise polynomial curves. There are many ways to derive B-splines. Here, we choose repeated convolution since we can see from it directly how splines can be generated through subdivision.

A.1 Continuous and Discrete Convolution

We begin by defining convolution and present some properties of convolutions we will need in the following sections.

Definition 1 : *The continuous convolution of functions f(t) and g(t), is defined as*

$$f(t) \otimes g(t) = \int f(s)g(t-s)ds \tag{A.1}$$

The functions $U_i(t) = U(t-i)$ are *translates* of $U(t)$. By construction, these functions are 1 between i and $i+1$ A degree zero B-spline is the sum of translates of $U(t)$.

$$f(t) = \sum_i p_i U_i(t) \tag{A.2}$$

This B-spline is *uniform* since the breaks between each pair of adjacent constant functions are evenly spaced.

The piecewise constant functions over half-integers, quarter-integers, etc, are *dilates* of $U(t)$.

$$U_i^j(t) = U(2^j t - i). \tag{A.3}$$

The following theorem shows how convolution effects dilates and translates.

Theorem 1 : *Let f(t), g(t), and h(t) be functions and let* $m(t) = f(t) \otimes g(t)$ *be the continuous convolution of f(t) and g(t), then*

$$f(t) \otimes (g(t) + h(t)) = f(t) \otimes g(t) + f(t) \otimes h(t) \qquad \text{linearity} \tag{A.4}$$

$$f(t - i) \otimes g(t - k) = m(t - i - k) \qquad \text{time shift} \tag{A.5}$$

$$f(2t) \otimes g(2t) = \frac{1}{2} m(2t) \qquad \text{time scaling} \tag{A.6}$$

Proof : Substituting the definition of convolution and using simple properties of integration proves these identities. Linearity;

$$\begin{aligned} f(t) \otimes (g(t) + h(t)) &= \int f(s) \left(g(t - s) + h(t - s)\right) ds \\ &= \int f(s) g(t - s) ds + \int f(s) h(t - s) ds \\ &= f(s) \otimes g(t) + f(s) \otimes h(t) \end{aligned}$$

time shifting;

$$\begin{aligned} g(t - i) \otimes h(t - j) &= \int g(s - i) h(t - j - s) ds = \int g(\overline{s}) h(t - i - j - \overline{s}) d\overline{s} \\ &= \int g(\overline{s}) h(\overline{t} - \overline{s}) d\overline{s} = g(\overline{t}) \otimes h(\overline{t}) \\ &= m(\overline{t}) = m(t - i - j), \end{aligned}$$

where we can change variables $\overline{s} = s - i$ and $\overline{t} = t - i - j$

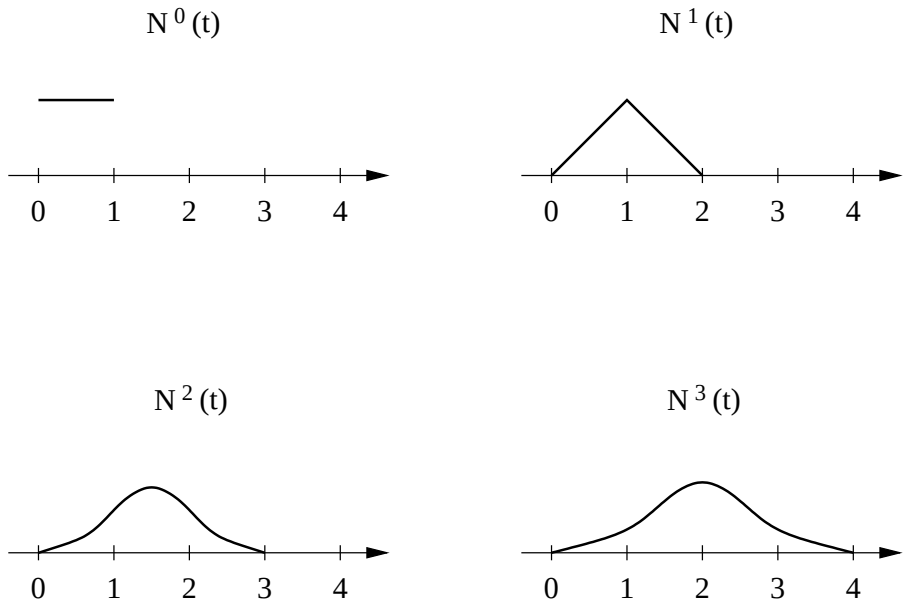


Figure A.1: Graph of a B-spline function of degree zero, one, two and three.

time scaling;

$$\begin{aligned} g(2t) \otimes h(2t) &= \int g(2s)h(2(t-s))ds = \frac{1}{2}\int g(\overline{s})h(2t-\overline{s})d\overline{s} \\ &= \frac{1}{2}g(2t) \otimes h(2t) = \frac{1}{2}m(2t), \end{aligned}$$

where $\overline{s} = 2s$. ∎

The next theorem shows that convolution raises the degree of differentiability.

Theorem 2 : *If f(t) is a C^k continuous function then $U(t) \otimes f(t)$ is a C^{k+1} continuous function.*

Proof : Using the definition of convolution and $U(t)$ a direct calculation gives

$$\begin{aligned} U(t) \otimes f(t) &= \int U(s)f(t-s)ds = \int_0^1 f(t-s)ds \\ &= -\int_t^{t-1} f(\overline{s})d\overline{s} = \int_{t-1}^{t} f(\overline{s})d\overline{s}, \end{aligned}$$

with $\overline{s} = t - s$. The theorem then follows from the fact that the differentiability of the last integral is one higher than that of $f(t)$. ∎

We now turn to the discrete convolution of sequences.

Definition 2 : *Let a and b sequences, then the discrete convolution of a and b is a sequence, c, with elements*

$$c_k = (a \otimes b)_k = \sum_{i+j=k} a_i b_j. \tag{A.7}$$

Certain properties of discrete convolution can conveniently be expressed in terms of *generating functions*. The generating function of a sequence, a, is the unique polynomial function, $A(t)$, given by

$$A(t) = \sum_k a_k t^k. \tag{A.8}$$

It is easy to verify that the generating function of the sequence c in definition 2 can be written as

$$C(t) = A(t)B(t), \tag{A.9}$$

where $A(t)$ and $B(t)$ are the generating functions for the sequences a and b, respectively.

When we write the function, $f(t)$, in terms of its dilates we get the equation of form

$$f(t) = \sum_i c_i f(2t - i), \tag{A.10}$$

These equations are called *subdivision equations* or *refinement equations*. The following theorem provides the link between subdivision equations and discrete convolution.

Theorem 3 : *Let $h(t) = f(t) \otimes g(t)$ be the continuous convolution of two functions, f(t) and g(t), that both satisfy subdivision equations*

$$f(t) = \sum_i a_i f(2t - i) \tag{A.11}$$

$$g(t) = \sum_j b_j g(2t - j). \tag{A.12}$$

Then, h(t) satisfies a similar subdivision equation

$$h(t) = \sum_k c_k h(2t - k),$$

where $c_k = \sum_{i+j=k} \frac{1}{2} a_i b_j = \frac{1}{2}(a \otimes b)_k$.

Proof : By the linearity of continuous convolution (Theorem 1)

$$\begin{aligned} h(t) &= f(t) \otimes g(t) = \left(\sum_i a_i f(2t-i) \right) \otimes \left(\sum_j b_j g(2t-j) \right) \\ &= \sum_i \sum_j (a_i b_j f(2t-i) \otimes g(2t-j)). \end{aligned}$$

From Theorem 1 we also have $f(2t-i) \otimes g(2t-j) = \frac{1}{2} h(2t-i-j)$. Inserting this in the sum we get

$$\begin{aligned} h(t) &= \sum_i \sum_j \left(\frac{1}{2} a_i b_j h(2t-i-j) \right) = \sum_k \sum_{i+j=k} \frac{1}{2} a_i b_j h(2t-k) \\ &= \sum_k c_k h(2t-k), \end{aligned}$$

where $c_k = \sum_{i+j=k} \frac{1}{2} a_i b_j$. ∎

Comparing the definition of the generating function with the elements in the sequence c in Theorem 3 we see that the generating function of c is $C(t) = \frac{1}{2} A(t) B(t)$.

A.2 B-spline Functions

Given a *uniform* knot vector, i.e. a vector

$$T^0 = \left[\cdots, t^0_{-2}, t^0_{-1}, t^0_0, t^0_1, t^0_2, \cdots \right],$$

where the knots t_i are equally spaced and $t_i < t_{i+1}$, we define the characteristic function $U(t)$ is

$$U(t) = \begin{cases} 1 & \text{if } t^0_0 \le t < t^0_1 \\ 0 & \text{otherwise} \end{cases}$$

Subdivision of $f(t)$ involves expressing $f(t)$ in terms of finer and finer dilates of $U(t)$.

$$f(t) = \sum_i p^j_i U^j_i(t). \tag{A.13}$$

If $U^j(t)$ denotes the row vector whose i^th entry is $U_i^j(t)$, then in vector form

$$f(t) = U^j(t)p^j. \tag{A.14}$$

This process is possible due to the fact that $U(t)$ can be expressed in terms of its dilates. The functions

$$U(2t) = 1 \quad \text{for} \quad 0 \leq t < \frac{1}{2} \qquad \text{and}$$

$$U(2t-1) = 1 \quad \text{for} \quad \frac{1}{2} \leq t < 1$$

so;

$$U(t) = U(2t) + U(2t-1).$$

This relation is a subdivision formula for $U(t)$. More generally,

$$U_i^j(t) = U_{2i}^{j+1}(t) + U_{2i+1}^{j+1}(t). \tag{A.15}$$

In terms of matrices,

$$U^j(t) = U^{j+1}(t)S, \tag{A.16}$$

where S is the matrix whose entries $(2i, i)$ and $(2i+1, i)$ are 1 and 0 otherwise. A finite portion of S (rows -4 to 3 and columns -4 to 3) is,

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

S is the *subdivision matrix* associated with this process. If the initial set of coefficients p^0 are just p then substituting into Equation A.14 yield the relation

$$p^{j+1} = p^j S. \tag{A.17}$$

Application of the subdivision matrix S to p^i produces a new set of coefficients p^{j+1}.

Higher degree B-splines can be defined in a variety of ways. Perhaps the simplest definition is through convolution.

A B-spline basis function of degree $n, N_n(t)$ satisfies

$$N_n(t) = \bigotimes_{i=0}^{n} U(t) \tag{A.18}$$

If $n = 0$, then $N_0(t) = U(t)$, if $n = 1$, then $N_1(t) = U(t) \otimes U(t)$, and so on.

A few important properties of these functions are

- $N(t)$ is piecewise polynomial function of degree n.
- The support of $N(t)$ lies between 0 and $n + 1$.
- $N(t)$ is a C^{n-1} function (Theorem 2).
- The sum of the translates of $N(t)$ is the function 1.
- $N(t)$ is non-negative everywhere.

For example, in the case of $N_1(t)$ we get

$$\begin{aligned}
N_1(t) &= N_0(t) \otimes N_0(t) \\
&= (N_0(2t) + N_0(2t-1)) \otimes (N_0(2t) + N_0(2t-1)) \\
&= N_0(2t) \otimes N_0(2t) + N_0(2t) \otimes N_0(2t-1) + \\
&\quad N_0(2t-1) \otimes N_0(2t) + N_0(2t-1) \otimes N_0(2t-1) \\
&= \frac{1}{2}N_1(2t) + \frac{1}{2}N_1(2t-1) + \frac{1}{2}N_1(2t-1) + \frac{1}{2}N_1(2t-1-1) \\
&= \frac{1}{2}(N_1(2t) + 2N_1(2t-1) + N_1(2t-2)) \\
&= \frac{1}{2^1}\sum_{k=0}^{2} \begin{pmatrix} 2 \\ k \end{pmatrix} N_1(2t-k)
\end{aligned}$$

Replacing several factors of $U(2t)$ by $U(2t-1)$ yields various translates of $N(2t)$. Therefore, there must exist constant s_k such that

$$N(t) = \sum s_k N(2t-k) \tag{A.19}$$

For dilates and translates of $N(t)$, the subdivision formula is

$$N_i^j(t) = \sum s_k N_{2i+k}^{j+1}(t). \tag{A.20}$$

In matrix notation, the basis functions are related by

$$N^j(t) = \sum N^{j+1}(t)S, \tag{A.21}$$

where S is the matrix whose $S_{2i+k,i}$the entry is s_k and zero otherwise. The column vectors of control points p^j and p^{j+1} are related by

$$p^{j+1} = Sp^i. \tag{A.22}$$

In Equation A.22 the subdivision process is expressed as the repeated application of a fixed subdivision matrix to a set of coefficients For B-splines, another view is possible in terms of discrete convolution. By Definition 2 multiplication of generating functions is equivalent to convolving their associated coefficient sequences. In terms of the definition of discrete convolution

$$C(z) = A(z)B(z). \tag{A.23}$$

Discrete convolution can be used to derive the subdivision formula for the continuous convolution of two basis functions. Using the Theorem 2, the subdivision formula for the B-spline basis function $N(t)$ of degree n can be derived. The subdivision formula for $U(t)$ is

$$U(t) = U(2t) + U(2t-1).$$

If the generating function for this subdivision formula is $1+z$. By Theorem 2, the generating function $N(z)$ associated with the subdivision formula of Equation A.19 is

$$N(z) = \frac{1}{2^n}(1+z)^{n+1}. \tag{A.24}$$

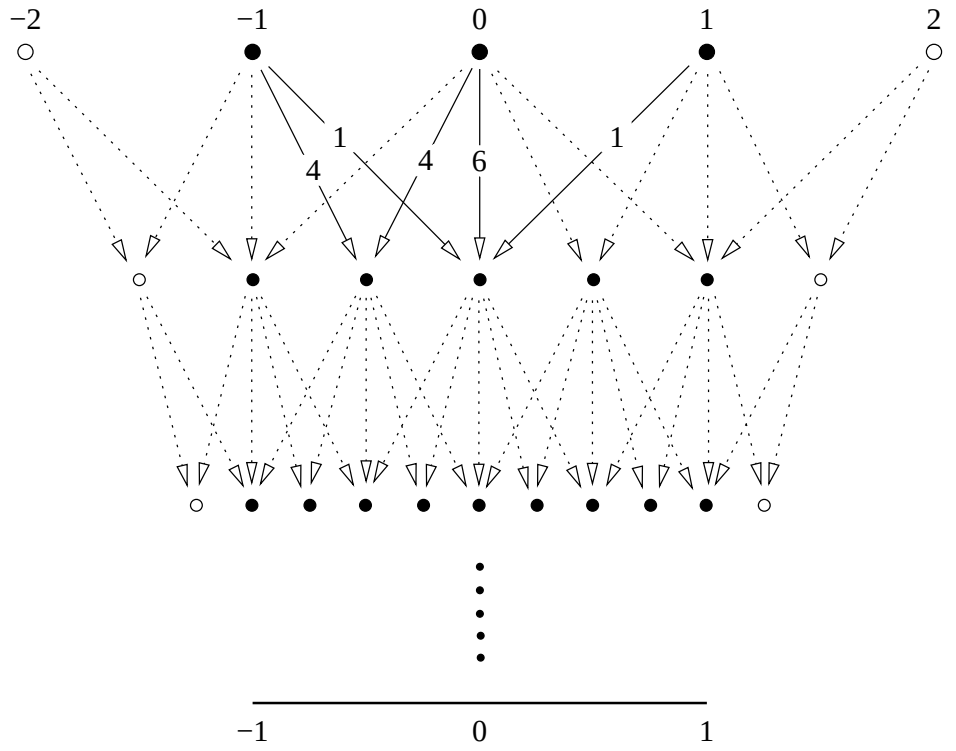


Figure A.2: Invariant neighborhood of cubic B-splines

A.3 Analysis of B-spline Curves

A.3.1 Invariant Neighborhood

One of the important question related to subdivision is which control point affect a given part of the limit curve or how curve changes when we change the position of a control point. One way to answer this question is to look at the support of the basis functions. . The knot vector and the degree determine the support of the B-spline functions. For the B-spline of degree k, the support is $k+1$ consecutive knot intervals.

Let's consider cubic B-spline subdivision. Suppose we want to find the control points needed to define the limit curve on the intervals next to the origin in the interval $[-1, 1]$. Figure A.2 shows that we need 5 control points at the coarsest level to reach any point of the limit curve. At each level we need one more control point on the outside of the interval $[-1, 1]$, in order to continue on the next subdivision level. So for the cubic

B-splines the size of invariant neighborhood is 5.

A.3.2 Eigen Analysis

An *eigenvector* x of the matrix M is a non-zero vector such that $Mx = \lambda x$, where λ is a scalar value. λ is called as the *eigenvalue* corresponding to the right eigenvector x.

Let's assume a subdivision matrix S with the size nxn and has the real eigenvectors $x_0, x_1, \cdots, x_{n-1}$, with corresponding real eigenvalues $\lambda_0, \lambda_1, \cdots, \lambda_{n-1}$. These eigenvectors forms a basis, where eigenvalues satisfies $\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{n-1}$. For example in the case of cubic B-splines where $n = 5$

$$(\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4) = (1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$$

$$(x_0, x_1, x_2, x_3, x_4) = \begin{pmatrix} 1 & -1 & 1 & 1 & 0 \\ 1 & \frac{-1}{2} & \frac{2}{11} & 0 & 0 \\ 1 & 0 & \frac{-1}{11} & 0 & 0 \\ 1 & \frac{1}{2} & \frac{2}{11} & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Given these eigenvectors we have

$$S(x_0, x_1, x_2, x_3, x_4) = (x_0, x_1, x_2, x_3, x_4) \begin{pmatrix} \lambda_0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & 0 & \lambda_4 \end{pmatrix}$$

$$SX = XD$$

$$X^{-1}SX = D.$$

The rows of x_i of X^1 are called left eigenvectors. Some subdivision schemes like 4 point, not have all complete set of real eigenvalues or eigenvectors and, this causes some technical difficulties. We will ignore these degeneracies and assume that we have a complete set of eigenvectors. A control point can be written in the basis formed by

the eigenvectors for some coefficients a_i, as

$$p = \sum_{i=0}^{n-1} a_i x_i$$

Then Equation A.22 can be written in terms of the eigenvectors

$$\begin{aligned} p^{j+1} = Sp^i &= S\sum_{i=0}^{n-1} a_i x_i \\ &= \sum_{i=0}^{n-1} a_i S x_i \qquad \text{by linearity of S} \\ &= \sum_{i=0}^{n-1} a_i \lambda_i x_i \end{aligned}$$

and applying S j-times yields

$$p^j = S^j p^0 = \sum_{i=0}^{n-1} a_i \lambda_i^j x_i$$

A.3.3 Convergence Analysis

A subdivision procedure S is convergent, if there exist a unique limit point p such that

$$\lim_{i \to \infty} x_i = p$$

The $S^j x_0$ in Equation A.25 guides us while examining the convergence. If $\lambda_0 > 1$ then the Equation A.25 would grow without bound as j increased and subdivision would not be convergent. Consequently, the Equation $S^j x_0$ converge at all, since all the eigenvalues must be smaller or equal to 1. Warren [19] show that only a single eigenvalue may have magnitude 1. A result of this analysis is that we can compute the limit position directly in the eigen basis as

$$p^\infty(0) \ = lim_{j\to\infty} S^j p^0 = lim_{j\to\infty} \sum_{i=0}^{n-1} a_i \lambda_i^j x_i = a_0$$

since all the eigenvalues $|\lambda_i| \leq 1$.

Consequently, for our subdivision matrix S the following characteristics desired,

- the eigenvectors should form a basis;
- the first eigenvalue λ_0 should be 1;
- the second eigenvalue λ_1 should be less than 1;
- all other eigenvalues should be less than λ_1.

Appendix B

Utilities

To model objects using subdivision surfaces we have to use some utilities to support input handling and the rendering of the modeled surfaces.

B.1 The Readers

Since we also need to access data and use it in a graphical platform to render the surfaces two different readers are implemented to access two different types of data formats. The data formats are chosen in the light of the structure of the meshes in hand and ease of implementation. The data formats usually describes the vertices and their associated coordinates, the faces and the connections between the faces.

B.1.1 SMF File Format Specifications

This format is designed for the description of 3D geometric models. It describes a single object; it does not describe viewing parameters, scene hierarchies, or anything else. An SMF file is a text file consisting of a series of lines. Each line is interpreted independently and in sequence. A line may have one of the following three forms:

(1) Entirely whitespace
These lines are completely ignored.

(2) A comment line, beginning with the character '#'
These lines are also ignored.

(3) A command line of the form: <op> <arg>
The first token on the line is interpreted as a command name. The remaining tokens are arguments to the command; their interpretation is command-dependent. Tokens are whitespace-separated character sequences.

An SMF file consist of two or three sections. The first section specifies the vertices in the control mesh. It is initiated with the character `v` at the beginning of every line. Each subsequent line should be of the form

```
v x_coord y_coord z_coord
```

The vertex number must be an integer, while the coordinates are real numbers. Within a given scope, vertices are implicitly numbered starting from 1 (NOT 0). The fields in a line are separated by spaces.

The second section specifies the faces in the control mesh. It is initiated by the character `f` at the beginning of every line. A face is defined by a line of the form

$$\texttt{f vnum}_1\ \texttt{vnum}_2\ \ldots\texttt{vnum}_n$$

where $\texttt{vnum}_i$ is the number of the vertex defined in the previous section. The vertices should be listed in a counter-clockwise order to give the correct orientation of the face.

The third, optional, section specifies the attribute values. For specifying the sharp features the lines are initiated with the string `Edge` at the beginning. Each line is of the form

$$\texttt{Edge fnum}_1\ \texttt{fnum}_2\ \{\texttt{sharp}\}$$

where $\texttt{fnum}_1$ and $\texttt{fnum}_2$ are both faces defined in the previous section of the file. The edge shared by these two vertices are marked as sharp.

A simple example of an SMF file is given in Figure B.1. All the reading process for the SMF file is contained in a class.

```
# Model for testing SMF reader
# SMF 1.0
# vertices 13
# faces 14

v   0.141979  -0.558788  0.929668
v   0.185852  -0.513599  0.829234
v   0.267333  -0.55719   0.822234
v   0.212088  -0.614101  0.948711
v   0.351817  -0.586426  0.907718
v   0.316137  -0.6334    0.977069
v   0.514369  -0.661343  0.929047
v   0.47456   -0.702343  1.020183
v   0.541247  -0.65971   1.038447
v   0.214357  -0.423649  0.837115
v   0.303229  -0.443924  0.832159
v   0.378885  -0.501011  0.915211
v   0.53383   -0.599934  0.934419

f   1  2  4
f   3  4  2
f   3  5  6
f   4  3  6
f   6  5  8
f   7  8  5
f   8  7  9
f   2 10 11
f   3  2 11
f   3 12  5
f  11 12  3
f   5 13  7
f  12 13  5
f   7 13  9

Edge 1 2 {sharp}
Edge 2 3 {sharp}
Edge 3 4 {sharp}
Edge 2 4 {sharp}
```

Figure B.1: A sample SMF file

B.1.2 MDL File Format Specifications

An MDL file consist of three of four sections each initiated with a text string. The first section specifies the vertices in the control mesh. It is initiated with the string `vertices`. Each subsequent line should be of the form

`vnum x_coord y_coord z_coord`

The vertex number `vnum`, must be an integer, while the coordinates are real numbers. The fields in a line are separated by spaces.

The second section specifies the faces in the control mesh. It is initiated by the string `faces`. A face is defined by a line of the form

$$\texttt{fnum vnum}_1\ \texttt{vnum}_2\ \ldots\texttt{vnum}_n$$

where `fnum` is a number identifying the face and $\texttt{vnum}_i$ is the number of the vertex defined in the previous section. The vertices should be listed in a counter-clockwise order to give the correct orientation of the face.

The third section specifies the connections between faces. It is initiated with the string `connections`. Each line is of the form

$$\texttt{fnum}_1\ \texttt{fnum}_2$$

where $\texttt{fnum}_1$ and $\texttt{fnum}_2$ are both faces defined in the previous section of the file. The two faces must share two vertices in the same orientation.

The fourth optional section specifies the knot spacings for edges in the control polygon. It is initiated by the string `knotspacings`. The knot spacings are given by a line

$$\texttt{fnum cwkspc}_1\ \texttt{ccwkspc}_1\ \texttt{cwkspc}_2\ \texttt{ccwkspc}_2\ \ldots\texttt{cwkspc}_n\ \texttt{ccwkspc}_n$$

giving the knot spacings for face number `fnum`. There are two knot spacings for each vertex one for the clockwise edge and the other for the counter-clockwise edge.

```
# Model for testing MDL reader
# Hexagon.mdl
# vertices 8
# faces 8

vertices
1  -2.0  0.0  0.0
2   0.0  0.0  0.0
3   2.0  0.0  0.0
4  -1.0 -2.0  1.0
5   1.0 -2.0  1.0
6  -1.0  2.0  1.0
7   1.0  2.0  1.0
8   0.0  4.0  0.0
9   0.0 -4.0  0.0

faces
1 1 4 2
2 4 5 2
3 5 3 2
4 3 7 2
5 7 6 2
6 6 1 2
7 4 9 5
8 7 8 6

connections
1 2
2 3
3 4
4 5
5 6
6 1
5 8
2 7

knotspacings
1  0.0  1.0  1.0  1.0  1.0  0.0
2  4.0  1.0  1.0  1.0  1.0  4.0
3  1.0  1.0  1.0  0.0  0.0  1.0
4  0.0  1.0  1.0  1.0  1.0  0.0
5  1.0  1.0  1.0  4.0  4.0  1.0
6  4.0  1.0  1.0  0.0  0.0  4.0
```

Figure B.2: A sample MDL file

A simple example of an MDL file is given in Figure B.2. All the reading process for the MDL file is contained in a class.

B.2 Rendering Process

We have used OpenGL to render the models on the screen. The user interface is developed by `glow 1.0.4` API takes control of the OpenGL routines. We initialize the OpenGL to control the graphics window, set up the proper projections, lighting and material properties. There are two routines for drawing surfaces: *wireframe* and *smooth-shaded*.