Game Developer's Conference 2001 Course Notes

Using Subdivision Surfaces

Organizers: Denis Zorin, New York University
Peter Schröder, Caltech

Lecturers

Denis Zorin
Media Research Laboratory
719 Broadway,rm. 1201
New York University
New York, NY 10012
net: dzorin@mrl.nyu.edu

Peter Schröder
Caltech Multi-Res Modeling Group
Computer Science Department 256-80
California Institute of Technology
Pasadena, CA 91125
net: ps@cs.caltech.edu

Stephen Junkins
IAL Graphics and 3D Technologies
Intel, Suite 206
1011 SW Emkay Drive
Bend, OR 97702
net: stephen.junkins@intel.com

Henry Moreton
NVIDIA
3535 Monroe Street
Santa Clara, CA 95051
net: moreton@nvidia.com

Schedule

Morning Session: Introductory Material The morning section will focus on the foundations of subdivision, starting with subdivision curves and moving on to surfaces. We will review and compare a number of different schemes and discuss the relation between subdivision and splines. The emphasis will be on properties of subdivision most relevant for applications.

Foundations I: Basic Ideas
Peter Schröder and Denis Zorin

Foundations II: Subdivision Schemes for Surfaces
Denis Zorin

Afternoon Session: Applications and Algorithms The afternoon session will focus on applications of subdivision and the algorithmic issues practitioners need to address to build efficient, well behaving systems for modeling and animation with subdivision surfaces.

Multiresolution Subdivision Surfaces
Denis Zorin

Implementing Subdivision: Data Structures and Performance Issues
Stephen Junkins

Displaced Subdivision Surfaces
Henry Moreton

Compression and Progressive Transmission of Subdivision Surfaces with Details
Peter Schröder

Lecturers' Biographies

Denis Zorin is an assistant professor at the Courant Institute of Mathematical Sciences, New York University. He received a BS degree from the Moscow Institute of Physics and Technology, a MS degree in Mathematics from Ohio State University and a PhD in Computer Science from the California Institute of Technology. In 1997-98, he was a research associate at the Computer Science Department of Stanford University. His research interests include multiresolution modeling, the theory of subdivision, and applications of subdivision surfaces in Computer Graphics and CAGD. He co-organized the Siggraph course on subdivision surfaces for the past three years.

Peter Schröder is an associate professor of computer science and applied mathematics at Caltech, Pasadena, where he directs the Multi-Res Modeling Group. He received a Master's degree from the MIT Media Lab and a PhD from Princeton University. For the past 8 years his work has concentrated on exploiting wavelets and multiresolution techniques to build efficient representations and algorithms for many fundamental computer graphics problems. His current research focuses on subdivision as a basic paradigm for geometric modeling and rapid manipulation of large, complex geometric models. The results of his work have been published in venues ranging from Siggraph to special journal issues on wavelets and WIRED magazine, and he is a frequent consultant to industry. Most recently his work was honored when he was named a Packard Foundation Fellow. He has been the co-organizer of the Siggraph course on subdivision surfaces for the past three years.

Stephen Junkins is a senior software architect in the Graphics and 3D Technologies group in the Intel Architecture Labs. He has spent the past year as a senior contributor to the Shockwave 3D joint effort between Macromedia and Intel. Previously, he was the lead architect of the Intel 3D Software Toolkit, releases 1 and 2. His specific algorithm contributions include the Toolkit's Multi-Resolution Mesh (MRM) Technology currently being used by Valve Software, Pandemic Studios, and many other immersive 3D game developers. He also contributed the real time Subdivision Surface implementation. In previous lives, he has worked for the Los Alamos National Labs's Advanced Computing Laboratory, Siemens Corporation's Medical Imaging Group, Software Architects Inc., and Clemson University's Virtual Reality Center.

Henry Moreton joined NVIDIA in the fall of 1998 as a member of the architecture group. From 1984 to 1998 he worked at Silicon Graphics. In 1992 he received a Ph.D. from the University of California,

Berkeley. He has published in the areas of curve and surface modeling, rendering, texture mapping, video and image compression, and unmanned submarine control. He has patents issued and pending in the areas of optics, video compression, system and CPU architecture, and curve and surface modeling and rendering. Previous occupations include jackhammer operator and shotgun salesman. Other interests include skiing, squash and excavation.

1 Introduction **11**

2 Foundations I: Basic Ideas **15**
2.1 The Idea of Subdivision 16
2.2 Review of Splines 20
2.2.1 Piecewise Polynomial Curves 20
2.2.2 Definition of B-Splines 21
2.2.3 Refinability of B-splines 23
2.2.4 Refinement for Spline Curves 25
2.2.5 Subdivision for Spline Curves 27
2.3 Subdivision as Repeated Refinement 28
2.3.1 Discrete Convolution 28
2.3.2 Convergence of Subdivision 30
2.3.3 Summary 33
2.4 Analysis of Subdivision 33
2.4.1 Invariant Neighborhoods 34
2.4.2 Eigen Analysis 38
2.4.3 Convergence of Subdivision 40
2.4.4 Invariance under Affine Transformations 40
2.4.5 Geometric Behavior of Repeated Subdivision 41
2.4.6 Size of the Invariant Neighborhood 43
2.4.7 Summary 43

3 Subdivision Surfaces **45**
3.1 Subdivision Surfaces: an Example 46
3.2 Natural Parameterization of Subdivision Surfaces 48
3.3 Subdivision Matrix 51
3.4 Smoothness of Surfaces 54
3.4.1 C^1-continuity and Tangent Plane Continuity 54
3.5 Analysis of Subdivision Surfaces 55
3.5.1 C^1-continuity of Subdivision away from Extraordinary Vertices 56
3.5.2 Smoothness Near Extraordinary Vertices 58
3.5.3 Characteristic Map 59
3.6 Piecewise-smooth surfaces and subdivision 61

4 Subdivision Zoo **63**
4.1 Overview of Subdivision Schemes . . . 63
4.1.1 Notation and Terminology . . . 66
4.2 Loop Scheme . . . 67
4.3 Modified Butterfly Scheme . . . 70
4.4 Catmull-Clark Scheme . . . 74
4.5 Kobbelt Scheme . . . 74
4.6 Doo-Sabin and Midedge Schemes . . . 77
4.7 Uniform Approach to Quadrilateral Subdivision . . . 79
4.8 Comparison of Schemes . . . 82
4.8.1 Comparison of Dual Quadrilateral Schemes . . . 86
4.9 Tilings . . . 86
4.10 Limitations of Stationary Subdivision . . . 89

5 Implementing Subdivision and Multiresolution Surfaces **103**
5.1 Data Structures for Subdivision . . . 103
5.1.1 Representing Arbitrary Meshes . . . 103
5.1.2 Hierarchical Meshes: Arrays vs. Trees . . . 105
5.1.3 Implementations . . . 107

6 Implementing Subdivision: Slides

7 Displaced Subdivision Surfaces: Slides

Chapter 1

Introduction

More than twenty years ago the publication of the papers by Catmull and Clark [3] and Doo and Sabin [4] marked the beginning of subdivision for surface modeling. Now we can regularly see subdivision used in movie production (e.g., Geri's Game, A Bug's Life, and Toy Story 2), appear as a first class citizen in commercial modelers, and as a core technology in game engines.

The basic ideas behind subdivision are very old indeed and can be traced as far back as the late 40s and early 50s when G. de Rham used "corner cutting" to describe smooth curves. It was only recently though that subdivision surfaces have found their way into wide application in computer graphics and computer assisted geometric design (CAGD). One reason for this development is the importance of multiresolution techniques to address the challenges of ever larger and more complex geometry: subdivision is intricately linked to multiresolution and traditional mathematical tools such as wavelets.

Constructing surfaces through subdivision elegantly addresses many issues that computer graphics practitioners are confronted with

- **Arbitrary Topology:** Subdivision generalizes classical spline patch approaches to arbitrary topology. This implies that there is no need for trim curves or awkward constraint management between patches.

- **Scalability:** Because of its recursive structure, subdivision naturally accommodates level-of-detail rendering and adaptive approximation with error bounds. The result are algorithms which can make the best of limited hardware resources, such as those found on low end display terminals.

- **Uniformity of Representation:** Much of traditional modeling uses either polygonal meshes or spline patches. Subdivision spans the spectrum between these two extremes. Surfaces can behave

as if they are made of patches, or they can be treated as if consisting of many small polygons.

- **Numerical Stability:** The meshes produced by subdivision have many of the nice properties finite element solvers require. As a result subdivision representations are also highly suitable for many numerical simulation tasks which are of importance in engineering and computer animation settings.

- **Code Simplicity:** Last but not least the basic ideas behind subdivision are simple to implement and execute very efficiently. While some of the deeper mathematical analyses can get quite involved this is of little concern for the final implementation and runtime performance.

In this course and its accompanying notes we hope to convince you, the reader, that in fact the above claims are true!

The main focus of our notes will be on covering the basic principles behind subdivision; how subdivision rules are constructed; to indicate how their analysis is approached; and, most importantly, to address some of the practical issues in turning these ideas and techniques into real applications. We begin with some examples in the curve, i.e., 1D setting. This simplifies the exposition considerably, but still allows us to introduce all the basic ideas which are equally applicable in the surface setting. Proceeding to the surface setting we cover a variety of different subdivision schemes and their properties.

With these basics in place we proceed to the second, applications oriented part, covering algorithms and implementations addressing

- **Implementing Subdivision and Multiresolution Surfaces:** Subdivision can model smooth surfaces, but in many applications one is interested in surfaces which carry details at many levels of resolution. Multiresolution mesh editing extends subdivision by including detail offsets at every level of subdivision, unifying patch based editing with the flexibility of high resolution polyhedral meshes. In this part, we will focus on general implementation concerns common for subdivision and multiresolution surfaces based on subdivision.

- **Data Structures and Performance Issues:** In this section we will focus on the nitty-gritty of datastructures and implementation issues for control meshes and the subdivision hierarchy. Topics include quadtree datastructures, neighbor finding in quadtrees, array implementations, depth/breadth first subdivision, LOD controls, attribute subdivision, bones animation, vertex reuse, memory layout and caching.

- **Displaced Subdivision Surfaces:** In this section a new surface representation, the displaced subdivision surface is described. It represents a detailed surface model as a scalar-valued displace-

ment over a smooth domain surface. The representation defines both the domain surface and the displacement function using subdivision, allowing for simple and efficient evaluation of surface properties. We present a simple, automatic scheme for converting detailed geometric models into such a representation. The challenge in this conversion process is to find a simple subdivision surface that still faithfully expresses the detailed model as its offset. The displaced subdivision surfaces offer a number of benefits, including geometry compression, editing, animation, scalability, and adaptive rendering. In particular, the encoding of fine detail as a scalar function makes the representation extremely compact.

- **Compression and Progressive Transmission of Subdivision Surfaces with Details:** Once one allows for true multiresolution surfaces through the addition of details to a subdivision surface, highly detailed and complex geometries can be readily built. Because of their special structure such surfaces are ideally suited for very powerful compression algorithms. Using subdivision as a prediction operator and wavelets to encode the details followed by a zerotree algorithm results in exceedingly small archives which can be progressively decompressed at the receiver end.

Beyond these Notes

One of the reasons that subdivision is enjoying so much interest right now is that it is very easy to implement and very efficient. In fact it is used in many computer graphics courses at universities as a homework exercise. The mathematical theory behind it is very beautiful, but also very subtle and at times technical. We are not treating the mathematical details in these notes, since the notes are primarily intended for the computer graphics practitioners. However, for those interested in the theory there are many pointers to the literature.

These notes as well as other materials such as presentation slides, applets and code are available on the web at `http://www.mrl.nyu.edu/publications/gdc-tutorial2001` and all readers are encouraged to explore the online resources.

Chapter 2

Foundations I: Basic Ideas

Peter Schröder, Caltech

In this chapter we focus on the 1D case to introduce all the basic ideas and concepts before going on to the 2D setting. Examples will be used throughout to motivate these ideas and concepts. We begin initially with an example from interpolating subdivision, before talking about splines and their subdivision generalizations.

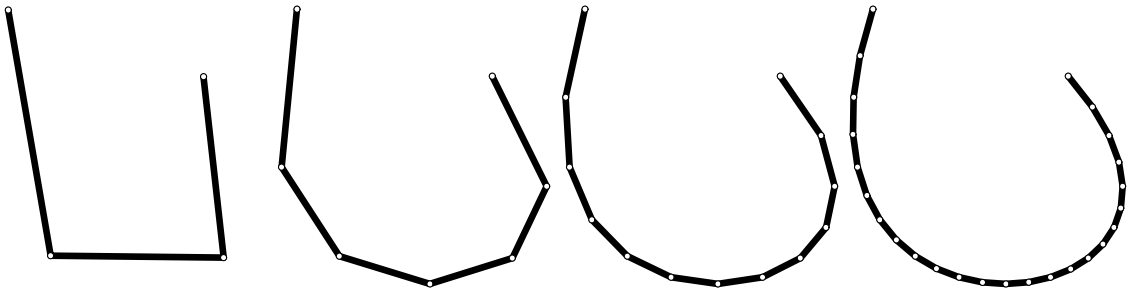


Figure 2.1: *Example of subdivision for curves in the plane. On the left 4 points connected with straight line segments. To the right of it a refined version: 3 new points have been inserted "inbetween" the old points and again a piecewise linear curve connecting them is drawn. After two more steps of subdivision the curve starts to become rather smooth.*

2.1 The Idea of Subdivision

We can summarize the basic idea of subdivision as follows:

> Subdivision defines a smooth curve or surface as the limit of a sequence of successive refinements.

Of course this is a rather loose description with many details as yet undetermined, but it captures the essence.

Figure 2.1 shows an example in the case of a curve connecting some number of initial points in the plane. On the left we begin with 4 points connected through straight line segments. Next to it is a refined version. This time we have the original 4 points and additionally 3 more points "inbetween" the old points. Repeating the process we get a smoother looking piecewise linear curve. Repeating once more the curve starts to look quite nice already. It is easy to see that after a few more steps of this procedure the resulting curve would be as well resolved as one could hope when using finite resolution such as that offered by a computer monitor or a laser printer.

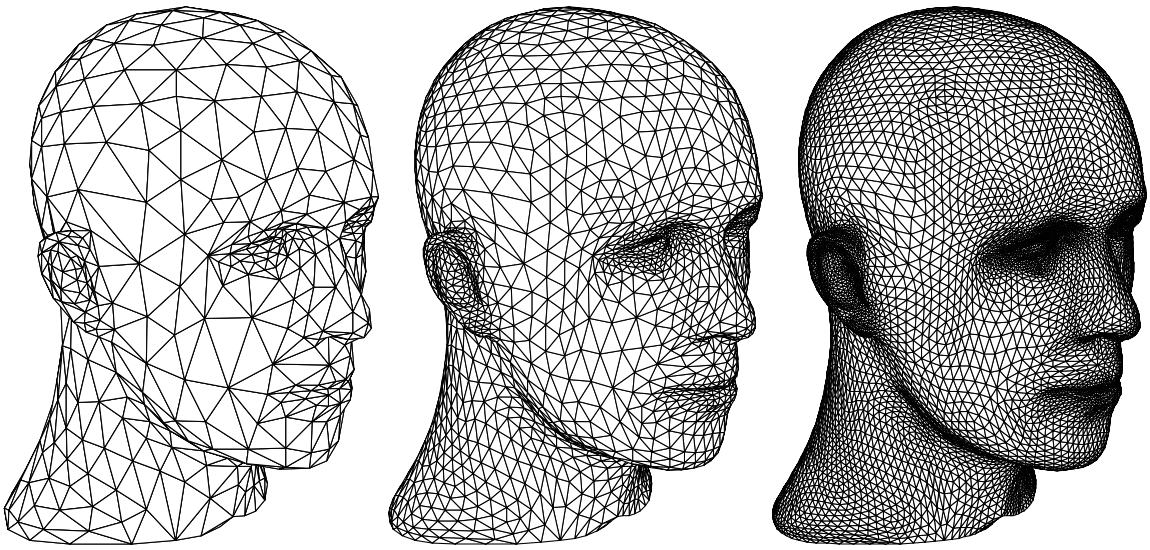


Figure 2.2: *Example of subdivision for a surface, showing 3 successive levels of refinement. On the left an initial triangle mesh approximating the surface. Each triangle is split into 4 according to a particular subdivision rule (middle). On the right the mesh is subdivided in this fashion once again.*

An example of subdivision for surfaces is shown in Figure 2.2. In this case each triangle in the original mesh on the left is split into 4 new triangles quadrupling the number of triangles in the mesh. Applying the same subdivision rule once again gives the mesh on the right.

Both of these examples show what is known as interpolating subdivision. The original points remain undisturbed while new points are inserted. We will see below that splines, which are generally not interpolating, can also be generated through subdivision. Albeit in that case new points are inserted *and* old points are moved in each step of subdivision.

How were the new points determined? One could imagine many ways to decide where the new points should go. Clearly, the shape and smoothness of the resulting curve or surface depends on the chosen rule. Here we list a number of properties that we might look for in such rules:

- **Efficiency:** the location of new points should be computed with a small number of floating point operations;
- **Compact support:** the region over which a point influences the shape of the final curve or surface should be finite and small;
- **Local definition:** the rules used to determine where new points go should not depend on "far away" places;
- **Affine invariance:** if the original set of points is transformed, e.g., translated, scaled, or rotated, the resulting shape should undergo the same transformation;
- **Simplicity:** determining the rules themselves should preferably be an offline process and there should only be a small number of rules;
- **Continuity:** what kind of properties can we prove about the resulting curves and surfaces, for example, are they differentiable?

For example, the rule used to construct the curve in Figure 2.1 computed new points by taking a weighted average of nearby old points: two to the left and two to the right with weights $1/16\,(-1,9,9,-1)$ respectively (we are ignoring the boundaries for the moment). It is very efficient since it only involves 4 multiplies and 3 adds (per coordinate); has compact support since only 2 neighbors on either side are involved; its definition is local since the weights do not depend on anything in the arrangement of the points; the rule is affinely invariant since the weights used sum to 1; it is very simple since only 1 rule is used (there is one more rule if one wants to account for the boundaries); finally the limit curves one gets by repeating this process ad infinitum are C^1.

Before delving into the details of how these rules are derived we quickly compare subdivision to other possible modeling approaches for smooth surfaces: traditional splines, implicit surfaces, and variational surfaces.

1. **Efficiency:** Computational cost is an important aspect of a modeling method. Subdivision is easy to implement and is computationally efficient. Only a small number of neighboring old points are used in the computation of the new points. This is similar to knot insertion methods found in spline modeling, and in fact many subdivision methods are simple generalizations of knot insertion. On the other hand implicit surfaces, for example, are much more costly. An algorithm such as marching cubes is required to generate the polygonal approximation needed for rendering. Variational surfaces can be even worse: a global optimization problem has to be solved each time the surface is changed.

2. **Arbitrary topology:** It is desirable to build surfaces of arbitrary topology. This is a great strength of implicit modeling methods. They can even deal with *changing* topology during a modeling session. Classic spline approaches on the other hand have great difficulty with control meshes of arbitrary topology. Here, "arbitrary topology" captures two properties. First, the topological genus, i.e., the number of handles, of the mesh and associated surface can be arbitrary. Second, the structure of the graph formed by the edges and vertices of the mesh can be arbitrary; specifically, each vertex may be of arbitrary degree.

   These last two aspects are related: if we insist on all vertices having degree 4 for quadrilateral control meshes, or having degree 6 for triangle control meshes, the Euler characteristic for a planar graph tells us that such meshes can only be constructed if the overall topology of the shape is that of the infinite plane, the infinite cylinder, or the torus. Any other shape, for example a sphere, cannot be built from a quadrilateral (triangle) control mesh having vertices of degree 4 (6).

   When rectangular spline patches are used in arbitrary control meshes, enforcing higher order continuity at extraordinary vertices becomes difficult and considerably increases the complexity of the representation (see Figure 2.3 for an example of points not having valence 4). Implicit surfaces can be of arbitrary topological genus, but the genus, precise location, and connectivity of a surface are typically difficult to control. Variational surfaces can handle arbitrary topology better than any other representation, but the computational cost can be high. Subdivision can handle arbitrary topology quite well without losing efficiency; this is one of its key advantages. Historically subdivision arose when researchers were looking for ways to address the arbitrary topology modeling challenge for splines.

3. **Surface features:** Often it is desirable to control the shape and size of features, such as creases, grooves, or sharp edges. Variational surfaces provide the most flexibility and exact control for creating features. Implicit surfaces, on the other hand, are very difficult to control, since all modeling

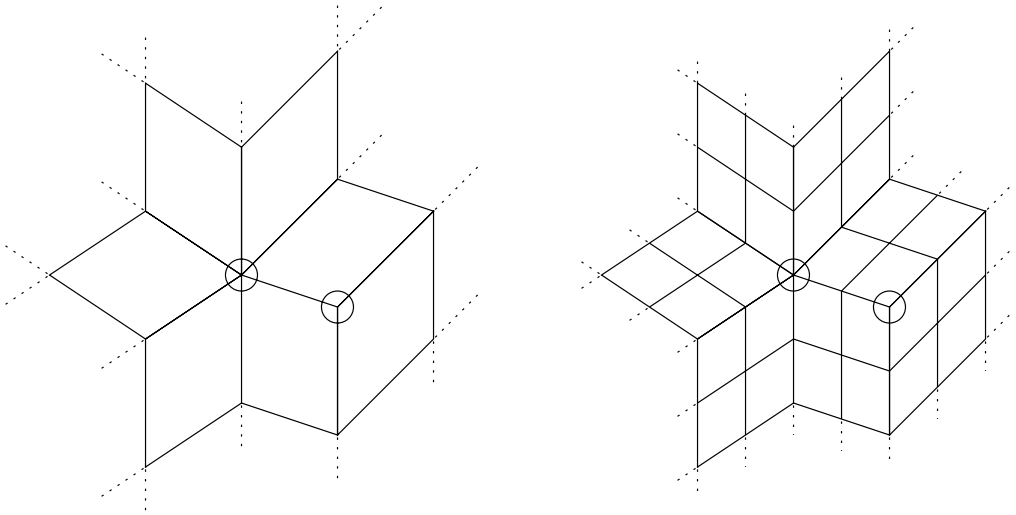


Figure 2.3: *A mesh with two extraordinary vertices, one with valence 6 the other with valence 3. In the case of quadrilateral patches the standard valence is 4. Special efforts are required to guarantee high order of continuity between spline patches meeting at the extraordinary points; subdivision handles such situations in a natural way.*

is performed indirectly and there is much potential for undesirable interactions between different parts of the surface. Spline surfaces allow very precise control, but it is computationally expensive and awkward to incorporate features, in particular if one wants to do so in arbitrary locations. Subdivision allows more flexible controls than is possible with splines. In addition to choosing locations of control points, one can manipulate the coefficients of subdivision to achieve effects such as sharp creases or control the behavior of the boundary curves.

4. **Complex geometry:** For interactive applications, efficiency is of paramount importance. Because subdivision is based on repeated refinement it is very straightforward to incorporate ideas such as level-of-detail rendering and compression for the internet. During interactive editing locally adaptive subdivision can generate just enough refinement based on geometric criteria, for example. For applications that only require the visualization of fixed geometry, other representations, such as progressive meshes, are likely to be more suitable.

Since most subdivision techniques used today are based upon and generalize splines we begin with a quick review of some basic facts of splines which we will need to understand the connection between splines and subdivision.

2.2 Review of Splines

2.2.1 Piecewise Polynomial Curves

Splines are piecewise polynomial curves of some chosen degree. In the case of cubic splines, for example, each polynomial segment of the curve can be written as

$$\begin{aligned} x(t) &= a_3^i t^3 + a_2^i t^2 + a_1^i t + a_0^i \\ y(t) &= b_3^i t^3 + b_2^i t^2 + b_1^i t + b_0^i, \end{aligned}$$

where $(\mathbf{a}, \mathbf{b})$ are constant coefficients which control the shape of the curve over the associated segment. This representation uses monomials (t^3, t^2, t^1, t^0), which are restricted to the given segment, as basis functions.

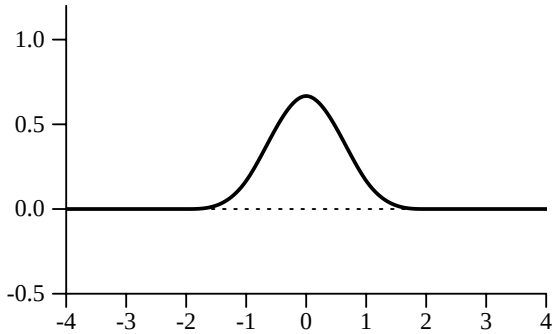


Figure 2.4: *Graph of the cubic B-spline. It is zero for the independent parameter outside the interval* $[-2, 2]$.

Typically one wants the curve to have some order of continuity along its entire length. In the case of cubic splines one would typically want C^2 continuity. This places constraints on the coefficients $(\mathbf{a}, \mathbf{b})$ of neighboring curve segments. Manipulating the shape of the desired curves through these coefficients, while maintaining the constraints, is very awkward and difficult. Instead of using monomials as the basic building blocks, we can write the spline curve as a linear combination of shifted *B-splines*, each with a coefficient known as a *control point*

$$\begin{aligned} x(t) &= \sum x_i B(t-i) \\ y(t) &= \sum y_i B(t-i). \end{aligned}$$

The new basis function $B(t)$ is chosen in such a way that the resulting curves are always continuous and that the influence of a control point is local. One way to ensure higher order continuity is to use basis

functions which are differentiable of the appropriate order. Since polynomials themselves are infinitely smooth, we only have to make sure that derivatives match at the points where two polynomial segments meet. The higher the degree of the polynomial, the more derivatives we are able to match. We also want the influence of a control point to be maximal over a region of the curve which is close to the control point. Its influence should decrease as we move away along the curve and disappear entirely at some distance. Finally, we want the basis functions to be piecewise polynomial so that we can represent any piecewise polynomial curve of a given degree with the associated basis functions. B-splines are constructed to exactly satisfy these requirements (for a cubic B-spline see Figure 2.4) and in a moment we will show how they can be constructed.

The advantage of using this representation rather than the earlier one of monomials, is that the continuity conditions at the segment boundaries are already "hardwired" into the basis functions. No matter how we move the control points, the spline curve will always maintain its continuity, for example, C^2 in the case of cubic B-splines.[1] Furthermore, moving a control point has the greatest effect on the part of the curve near that control point, and no effect whatsoever beyond a certain range. These features make B-splines a much more appropriate tool for modeling piecewise polynomial curves.

Note: When we talk about curves, it is important to distinguish the curve itself and the graphs of the coordinate functions of the curve, which can also be thought of as curves. For example, a curve can be described by equations $x(t) = \sin(t)$, $y(t) = \cos(t)$. The curve itself is a circle, but the coordinate functions are sinusoids. For the moment, we are going to concentrate on representing the coordinate functions.

2.2.2 Definition of B-Splines

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.

We start with the simplest case: piecewise constant coordinate functions. Any piecewise constant function can be written as

$$x(t) = \sum x_i B_0^i(t),$$

[1] The differentiability of the basis functions guarantees the differentiability of the coordinate functions of the curve. However, it does not guarantee the geometric smoothness of the curve. We will return to this distinction in our discussion of subdivision surfaces.

where $B_0(t)$ is the box function defined as

$$\begin{aligned} B_0(t) &= 1 \quad \text{if} \quad 0 \leq t < 1 \\ &= 0 \quad \text{otherwise,} \end{aligned}$$

and the functions $B_0^i(t) = B_0(t-i)$ are translates of $B_0(t)$. Furthermore, let us represent the continuous convolution of two functions $f(t)$and $g(t)$ with

$$(f \otimes g)(t) = \int f(s)g(t-s)ds.$$

A B-spline basis function of degree n can be obtained by convolving the basis function of degree $n-1$ with the box $B_0(t)$.[2] For example, the B-spline of degree 1 is defined as the convolution of $B_0(t)$ with itself

$$B_1(t) = \int B_0(s)B_0(t-s)ds.$$

Graphically (see Figure 2.5), this convolution can be evaluated by sliding one box function along the coordinate axis from minus to plus infinity while keeping the second box fixed. The value of the convolution for a given position of the moving box is the area under the product of the boxes, which is just the length of the interval where both boxes are non-zero. At first the two boxes do not have common support. Once the moving box reaches 0, there is a growing overlap between the supports of the graphs. The value of the convolution grows with t until $t = 1$. Then the overlap starts decreasing, and the value of the convolution decreases down to zero at $t = 2$. The function $B_1(t)$ is the linear hat function as shown in Figure 2.5.

We can compute the B-spline of degree 2 convolving $B_1(t)$ with the box $B_0(t)$ again

$$B_2(t) = \int B_1(s)B_0(t-s)ds.$$

In this case, the resulting curve consists of three quadratic segments defined on intervals $(0,1)$, $(1,2)$ and $(2,3)$. In general, by convolving l times, we can get a B-spline of degree l

$$B_l(t) = \int B_{l-1}(s)B_0(t-s)ds.$$

Defining B-splines in this way a number of important properties immediately follow. The first concerns the continuity of splines

[2]The *degree* of a polynomial is the highest order exponent which occurs, while the *order* counts the number of coefficients and is 1 larger. For example, a cubic curve is of degree 3 and order 4.

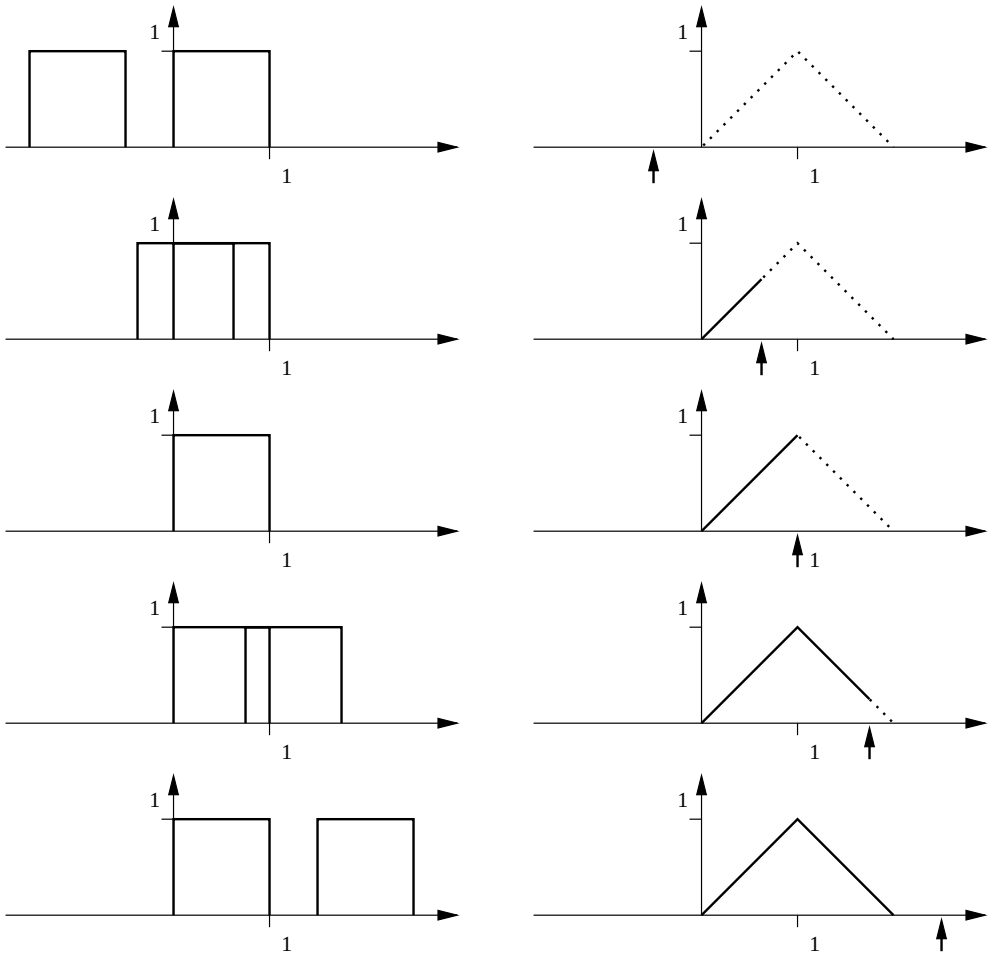


Figure 2.5: *The definition of degree 1 B-Spline $B_1(t)$ (right side) through convolution of $B_0(t)$ with itself (left side).*

Theorem 1 *If $f(t)$ is C^k-continuous, then $(B_0 \otimes f)(t)$ is C^{k+1}-continuous.*

This is a direct consequence of convolution with a box function. From this it follows that the B-spline of degree n is C^{n-1} continuous because the B-spline of degree 1 is C^0-continuous.

2.2.3 Refinability of B-splines

Another remarkable property of B-splines is that they obey a *refinement equation*. This is the key observation to connect splines and subdivision. The refinement equation for B-splines of degree l is

given by

$$B_l(t) = \frac{1}{2^l}\sum_{k=0}^{l+1}\binom{l+1}{k}B_l(2t-k). \tag{2.1}$$

In other words, the B-spline of degree l can be written as a linear combination of *translated* (k) and *dilated* ($2t$) copies of itself. For a function to be refinable in this way is a rather special property. As an example of the above equation at work consider the hat function shown in Figure 2.5. It is easy to see that it can be written as a linear combination of dilated hat functions with weights $(1/2, 1, 1/2)$ respectively.

The property of refinability is the key to subdivision and so we will take a moment to prove it. We start by observing that the box function, i.e., the B-spline of degree 0 can be written in terms of dilates and translates of itself

$$B_0(t) = B_0(2t) + B_0(2t-1), \tag{2.2}$$

which is easily checked by direct inspection. Recall that we defined the B-spline of degree l as

$$B_l(t) = \bigotimes_{i=0}^{l} B_0(t) = \bigotimes_{i=0}^{l} (B_0(2t) + B_0(2t-1)) \tag{2.3}$$

This expression can be "multiplied" out by using the following properties of convolution for functions $f(t)$, $g(t)$, and $h(t)$

$$\begin{array}{rcll} f(t)\otimes(g(t)+h(t)) & = & f(t)\otimes g(t) + f(t)\otimes h(t) & \text{linearity} \\ f(t-i)\otimes g(t-k) & = & m(t-i-k) & \text{time shift} \\ f(2t)\otimes g(2t) & = & \frac{1}{2}m(2t) & \text{time scaling} \end{array}$$

where $m(t) = f(t)\otimes g(t)$. These properties are easy to check by substituting the definition of convolution and amount to simple change of variables in the integration.

For example, in the case of B_1 we get

$$\begin{aligned} B_1(t) &= B_0(t)\otimes B_0(t) \\ &= (B_0(2t)+B_0(2t-1))\otimes(B_0(2t)+B_0(2t-1)) \\ &= B_0(2t)\otimes B_0(2t) + B_0(2t)\otimes B_0(2t-1) + B_0(2t-1)\otimes B_0(2t) + B_0(2t-1)\otimes B_0(2t-1) \\ &= \frac{1}{2}B_1(2t) + \frac{1}{2}B_1(2t-1) + \frac{1}{2}B_1(2t-1) + \frac{1}{2}B_1(2t-1-1) \\ &= \frac{1}{2}(B_1(2t) + 2B_1(2t-1) + B_1(2t-2)) \\ &= \frac{1}{2^1}\sum_{k=0}^{2}\binom{2}{k}B_1(2t-k). \end{aligned}$$

The general statement for B-splines of degree l now follows from the binomial theorem

$$(x+y)^{l+1} = \sum_{k=0}^{l+1} \binom{l+1}{k} x^{l+1-k} y^k,$$

with $B_0(2t)$ in place of x and $B_0(2t-1)$ in place of y.

2.2.4 Refinement for Spline Curves

With this machinery in hand let's revisit spline curves. Let

$$\gamma(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \sum_i p_i B_l^i(t)$$

be such a spline curve of degree l with control points $(x_i, y_i)^T = p_i \in \mathbf{R}^2$. Since we don't want to worry about boundaries for now we leave the index set i unspecified. We will also drop the subscript l since the degree, whatever it might be, is fixed for all our examples. Due to the definition of $B^i(t) = B(t-i)$ each control point exerts influence over a small part of the curve with parameter values $t \in [i, i+l]$.

Now consider $\mathbf{p}$, the vector of control points of a given curve:

$$\mathbf{p} = \begin{bmatrix} \vdots \\ p_{-2} \\ p_{-1} \\ p_0 \\ p_1 \\ p_2 \\ \vdots \end{bmatrix}$$

and the vector $\mathbf{B}(t)$, which has as its elements the translates of the function B as defined above

$$\mathbf{B}(t) = \begin{bmatrix} \dots & B(t+2) & B(t+1) & B(t) & B(t-1) & B(t-2) & \dots \end{bmatrix}.$$

In this notation we can denote our curve as $\mathbf{B}(t)\mathbf{p}$.

Using the refinement relation derived earlier, we can rewrite each of the elements of $\mathbf{B}$ in terms of its dilates

$$\mathbf{B}(2t) = \begin{bmatrix} \dots & B(2t+2) & B(2t+1) & B(2t) & B(2t-1) & B(2t-2) & \dots \end{bmatrix},$$

using a matrix S to encode the refinement equations

$$\mathbf{B}(t) = \mathbf{B}(2t)S.$$

The entries of S are given by Equation 2.1

$$S_{2i+k,i} = s_k = \frac{1}{2^l}\binom{l+1}{k}.$$

The only non-zero entries in each column are the weights of the refinement equation, while successive columns are copies of one another save for a shift down by two rows.

We can use this relation to rewrite $\gamma(t)$

$$\gamma(t) = \mathbf{B}(t)\mathbf{p} = \mathbf{B}(2t)S\mathbf{p}.$$

It is still the same curve, but described with respect to dilated B-splines, i.e., B-splines whose support is half as wide and which are spaced twice as dense. We performed a change from the old basis $\mathbf{B}(t)$ to the new basis $\mathbf{B}(2t)$ and concurrently changed the old control points $\mathbf{p}$ to the appropriate new control points $S\mathbf{p}$. This process can be repeated

$$\begin{aligned}
\gamma(t) &= \mathbf{B}(t)\mathbf{p}^0 \\
&= \mathbf{B}(2t)\mathbf{p}^1 &&= \mathbf{B}(2t)S\mathbf{p}^0 \\
&\quad\vdots \\
&= \mathbf{B}(2^j t)\mathbf{p}^j &&= \mathbf{B}(2^j t)S^j\mathbf{p}^0,
\end{aligned}$$

from which we can define the relationship between control points at different levels of subdivision

$$\mathbf{p}^{j+1} = S\mathbf{p}^j,$$

where S is our infinite subdivision matrix.

Looking more closely at one component, i, of our control points we see that

$$p_i^{j+1} = \sum_l S_{i,l}\, p_l^j.$$

To find out exactly which s_k is affecting which term, we can divide the above into odd and even entries. For the odd entries we have

$$p_{2i+1}^{j+1} = \sum_l S_{2i+1,l}\, p_l^j = \sum_l s_{2(i-l)+1}\, p_l^j$$

and for the even entries we have

$$p_{2i}^{j+1} = \sum_l S_{2i,l}\, p_l^j = \sum_l s_{2(i-l)}\, p_l^j.$$

From which we essentially get two different subdivision rules one for the new *even* control points of the curve and one for the new *odd* control points. As examples of the above, let us consider two concrete cases. For piecewise linear subdivision, the basis functions are hat functions. The odd coefficients are $\frac{1}{2}$ and $\frac{1}{2}$, and a lone 1 for the even point. For cubic splines the odd coefficients turn out to be $\frac{1}{2}$ and $\frac{1}{2}$, while the even coefficients are $\frac{1}{8}$, $\frac{6}{8}$, and $\frac{1}{8}$.

Another way to look at the distinction between even and odd is to notice that odd points at level $j+1$ are newly inserted, while even points at level $j+1$ correspond directly to the old points from level j. In the case of linear splines the even points are in fact the *same* at level $j+1$ as they were at level j. Subdivision schemes that have this property will later be called *interpolating*, since points, once they have been computed, will never move again. In contrast to this consider cubic splines. In that case even points at level $j+1$ are local averages of points at level j so that $p_{2i}^{j+1} \neq p_i^j$. Schemes of this type will later be called *approximating*.

2.2.5 Subdivision for Spline Curves

In the previous section we saw that we can refine the control point sequence for a given spline by multiplying the control point vector $\mathbf{p}$ by the matrix S, which encodes the refinement equation for the B-spline used in the definition of the curve. What happens if we keep repeating this process over and over, generating ever denser sets of control points? It turns out the control point sequence converges to the actual spline curve. The speed of convergence is geometric, which is to say that the difference between the curve and its control points decreases by a constant factor on every subdivision step. Loosely speaking this means that the actual curve is hard to distinguish from the sequence of control points after only a few subdivision steps.

We can turn this last observation into an algorithm and the core of the subdivision paradigm. Instead of drawing the curve itself on the screen we draw the control polygon, i.e., the piecewise linear curve through the control points. Applying the subdivision matrix to the control points defines a sequence of piecewise linear curves which quickly converge to the spline curve itself.

In order to make these observations more precise we need to introduce a little more machinery in the next section.

2.3 Subdivision as Repeated Refinement

2.3.1 Discrete Convolution

The coefficients s_k of the B-spline refinement equation can also be derived from another perspective, namely discrete convolution. This approach mimics closely the definition of B-splines through continuous convolution. Using this machinery we can derive and check many useful properties of subdivision by looking at simple polynomials.

Recall that the generating function of a sequence a_k is defined as

$$A(z) = \sum_k a_k z^k,$$

where $A(z)$ is the z-transform of the sequence a_k. This representation is closely related to the discrete Fourier transform of a sequence by restricting the argument z to the unit circle, $z = \exp(i\theta)$. For the case of two coefficient sequences a_k and b_k their convolution is defined as

$$c_k = (a \otimes b)_k = \sum_n a_{k-n} b_n.$$

In terms of generating functions this can be stated succinctly as

$$C(z) = A(z)B(z),$$

which comes as no surprise since convolution in the time domain is multiplication in the Fourier domain.

The main advantage of generating functions, and the reason why we use them here, is that manipulations of sequences can be turned into simple operations on the generating functions. A very useful example of this is the next observation. Suppose we have two functions that each satisfy a refinement equation

$$\begin{aligned} f(t) &= \sum_k a_k f(2t-k) \\ g(t) &= \sum_k b_k g(2t-k). \end{aligned}$$

In that case the convolution $h = f \otimes g$ of f and g also satisfies a refinement equation

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

whose coefficients c_k are given by the convolution of the coefficients of the individual refinement equations

$$c_k = \frac{1}{2}\sum_i a_{k-i}\, b_i.$$

With this little observation we can quickly find the refinement equation, and thus the coefficients of the subdivision matrix S, by repeated multiplication of generating functions. Recall that the box function $B_0(t)$ satisfies the refinement equation $B_0(t) = B_0(2t) + B_0(2t-1)$. The generating function of this refinement equation is $A(z) = (1+z)$ since the only non-zero terms of the refinement equation are those belonging to indices 0 and 1. Now recall the definition of B-splines of degree l

$$B_l(t) = \bigotimes_{k=0}^{l} B_0(t),$$

from which we immediately get the associated generating function

$$S(z) = \frac{1}{2^l}(1+z)^{l+1}.$$

The values s_k used for the definition of the subdivision matrix are simply the coefficients of the various powers of z in the polynomial $S(z)$

$$S(z) = \frac{1}{2^l}\sum_{k=0}^{l+1}\binom{l+1}{k} z^k,$$

where we used the binomial theorem to expand $S(z)$. Note how this matches the definition of s_k in Equation 2.1.

Recall Theorem 1, which we used to argue that B-splines of degree n are C^{n-1} continuous. That same theorem can now be expressed in terms of generating functions as follows

Theorem 2 *If $S(z)$ defines a convergent subdivision scheme yielding a C^k-continuous limit function then $\frac{1}{2}(1+z)S(z)$ defines a convergent subdivision scheme with C^{k+1}-continuous limit functions.*

We will put this theorem to work in analyzing a given subdivision scheme by peeling off as many factors of $\frac{1}{2}(1+z)$ as possible, while still being able to prove that the remainder converges to a continuous limit function. With this trick in hand all we have left to do is establish criteria for the convergence of a subdivision scheme to a continuous function. Once we can verify such a condition for the subdivision scheme associated with B-spline control points we will be justified in drawing the piecewise linear approximations of control polygons as approximations for the spline curve itself. We now turn to this task.

2.3.2 Convergence of Subdivision

There are many ways to talk about the convergence of a sequence of functions to a limit. One can use different norms and different notions of convergence. For our purposes the simplest form will suffice, uniform convergence.

We say that a sequence of functions f_i defined on some interval $[a,b] \subset \mathbf{R}$ *converges uniformly* to a limit function f if for all $\varepsilon > 0$ there exists an $n_0 > 0$ such that for all $n > n_0$

$$\max_{t \in [a,b]} |f(t) - f_n(t)| < \varepsilon.$$

Or in words, as of a certain index (n_0) all functions in the sequence "live" within an ε sized tube around the limit function f. This form of convergence is sufficient for our purposes and it has the nice property that if a sequence of continuous functions converges uniformly to some limit function f, that limit function is itself continuous.

For later use we introduce some norm symbols

$$\begin{aligned} \|f(t)\| &= \sup_t |f(t)| \\ \|\mathbf{p}\| &= \sup_i |p_i| \\ \|S\| &= \sup_i \sum_k |S_{ik}|, \end{aligned}$$

which are compatible in the sense that, for example, $\|S\mathbf{p}\| \leq \|S\|\|\mathbf{p}\|$.

The sequence of functions we want to analyze now are the control polygons as we refine them with the subdivision rule S. Recall that the control polygon is the piecewise linear curve through the control points $\mathbf{p}^j$ at level j. Independent of the subdivision rule S we can use the linear B-splines to define the piecewise linear curve through the control points as $P^j(t) = \mathbf{B_1}(2^j t)\mathbf{p}^j$.

One way to show that a given subdivision scheme S converges to a continuous limit function is to prove that (1) the limit

$$P^\infty(t) = \lim_{j \to \infty} P^j(t)$$

exists for all t and (2) that the sequence $P^j(t)$ converges uniformly. In order to show this property we need to make the assumption that all rows of the matrix S sum to 1, i.e., the odd and even coefficients of the refinement relation separately sum to 1. This is a reasonable requirement since it is needed to ensure the affine invariance of the subdivision process, as we will later see. In matrix notation this means $S\mathbf{1} = \mathbf{1}$, or in other words, the vector of all 1's is an eigenvector of the subdivision matrix with eigenvalue 1. In

terms of generating functions this means $S(-1) = 0$, which is easily verified for the generating functions we have seen so far.

Recall that the definition of continuity in the function setting is based on differences. We say $f(t)$ is continuous at t_0 if for any $\varepsilon > 0$ there exists a $\delta > 0$ so that $|f(t_0) - f(t)| < \varepsilon$ as long as $|t_0 - t| < \delta$. The corresponding tool in the subdivision setting is the difference between two adjacent control points $p_{i+1}^j - p_i^j = (\Delta \mathbf{p}^j)_i$. We will show that if the differences between neighboring control points shrink fast enough, the limit curve will exist and be continuous:

Lemma 3 *If $\|\Delta \mathbf{p}^j\| < c\gamma^j$ for some constant $c > 0$ and a shrinkage factor $0 < \gamma < 1$ for all $j > j_0 \geq 0$ then $P^j(t)$ converges to a continuous limit function $P^\infty(t)$.*

Proof: Let S be the subdivision rule at hand, $\mathbf{p}^1 = S\mathbf{p}^0$ and S_1 be the subdivision rule for B-splines of degree 1. Notice that the rows of $S - S_1$ sum to 0

$$(S - S_1)\mathbf{1} = S\mathbf{1} - S_1\mathbf{1} = \mathbf{1} - \mathbf{1} = \mathbf{0}.$$

This implies that there exists a matrix D such that $S - S_1 = D\Delta$, where Δ computes the difference of adjacent elements $(\Delta)_{ii} = -1$, $(\Delta)_{i,i+1} = 1$, and zero otherwise. The entries of D are given as $D_{ij} = -\sum_{k=i}^{j} (S - S_1)_{ik}$. Now consider the difference between two successive piecewise linear approximations of the control points

$$\begin{aligned}
\|P^{j+1}(t) - P^j(t)\| &= \|\mathbf{B_1}(2^{j+1}t)\mathbf{p}^{j+1} - \mathbf{B_1}(2^j t)\mathbf{p}^j\| \\
&= \|\mathbf{B_1}(2^{j+1}t)S\mathbf{p}^j - \mathbf{B_1}(2^{j+1}t)S_1\mathbf{p}^j\| \\
&= \|\mathbf{B_1}(2^{j+1}t)(S - S_1)\mathbf{p}^j\| \\
&\leq \|\mathbf{B_1}(2^{j+1}t)\|\|D\Delta\mathbf{p}^j\| \\
&\leq \|D\|\|\Delta\mathbf{p}^j\| \\
&\leq \|D\|c\gamma^j.
\end{aligned}$$

This implies that the telescoping sum $P^0(t) + \sum_{k=0}^{j} (P^{k+1} - P^k)(t)$ converges to a well defined limit function since the norms of each summand are bounded by a constant times a geometric term γ^j. Let $P^\infty(t)$ as $j \to \infty$, then

$$\|P^\infty(t) - P^j(t)\| < \frac{\|D\|c}{1 - \gamma}\gamma^j,$$

since the latter is the tail of a geometric series. This implies uniform convergence and thus continuity of $P^\infty(t)$ as claimed.

How do we check such a condition for a given subdivision scheme? Suppose we had a derived subdivision scheme D for the differences themselves

$$\Delta \mathbf{p}^{j+1} = D\Delta \mathbf{p}^{j},$$

defined as the scheme that satisfies

$$\Delta S = D\Delta.$$

Or in words, we are looking for a *difference scheme D* such that taking differences after subdivision is the same as applying the difference scheme to the differences. Does D always exist? The answer is yes if S is affinely invariant, i.e., $S(-1) = 0$. This follows from the following argument. Multiplying S by Δ computes a matrix whose rows are differences of adjacent rows in S. Since odd and even numbered rows of S each sum to one, the rows of ΔS must each sum to zero. Now the existence of a matrix D such that $\Delta S = D\Delta$ follows as in the argument above.

Given this difference scheme D all we would have to show is that some power $m > 0$ of D has norm less than 1, $\|D^m\| = \gamma < 1$. In that case $\|\Delta \mathbf{p}^j\| < c(\gamma^{1/m})^j$. (We will see in a moment that the extra degree of freedom provided by the parameter m is needed in some cases.)

As an example, let us check this condition for cubic B-splines. Recall that $B_3(z) = \frac{1}{8}(1+z)^4$, i.e.,

$$\begin{aligned} p_{2i+1}^{j+1} &= \frac{1}{8}(4p_i^j + 4p_{i+1}^j) \\ p_{2i}^{j+1} &= \frac{1}{8}(p_{i-1}^j + 6p_i^j + p_{i+1}^j). \end{aligned}$$

Taking differences we have

$$\begin{aligned} (\Delta \mathbf{p}^{j+1})_{2i} &= p_{2i+1}^{j+1} - p_{2i}^{j+1} = \frac{1}{8}(-p_{i-1}^j - 2p_i^j + 3p_{i+1}^j) \\ &= \frac{1}{8}(3(p_{i+1}^j - p_i^j) + 1(p_i^j - p_{i-1}^j)) = \frac{1}{8}(3(\Delta \mathbf{p}^j)_i + 1(\Delta \mathbf{p}^j)_{i-1}), \end{aligned}$$

and similarly for the odd entries so that $D(z) = \frac{1}{8}(1+z)^3$, from which we conclude that $\|D\| = \frac{1}{2}$, and that the subdivision scheme for cubic B-splines converges uniformly to a continuous limit function, namely the B-spline itself.

Another example, which is not a spline, is the so called 4 point scheme [5]. It was used to create the curve in Figure 2.1, which is interpolating rather than approximating as is the case with splines. The generating function for the 4 point scheme is

$$S(z) = \frac{1}{16}(-z^{-3} + 4z^{-2} - z^{-1})(1+z)^4$$

Recall that each additional factor of $\frac{1}{2}(1+z)$ in the generating function increases the order of continuity of the subdivision scheme. If we want to show that the limit function of the 4 point scheme is differentiable we need to show that $\frac{1}{8}(-z^{-3}+4z^{-2}-z^{-1})(1+z)^3$ converges to a continuous limit function. This in turn requires that $D(z)=\frac{1}{8}(-z^{-3}+4z^{-2}-z^{-1})(1+z)^2$ satisfy a norm estimate as before. The rows of D have non-zero entries of $(\frac{1}{4},\frac{1}{4})$, and $(\frac{-1}{8},\frac{6}{8},\frac{-1}{8})$ respectively. Thus $\|D\|=1$, which is not strong enough. However, with a little bit more work one can show that $\|D^2\|=\frac{3}{4}$, so that indeed the 4 point scheme is C^1.

In general, the difficult part is to find a set of coefficients for which subdivision converges. There is no general method to achieve this. Once a convergent subdivision scheme is found, one can always obtain a desired order of continuity by convolving with the box function.

2.3.3 Summary

So far we have considered subdivision only in the context of splines where the subdivision rule, i.e., the coefficients used to compute a refined set of control points, was fixed and everywhere the same. There is no pressing reason for this to be so. We can create a variety of different curves by manipulating the coefficients of the subdivision matrix. This could be done globally or locally. I.e., we could change the coefficients within a subdivision level and/or between subdivision levels. In this regard, splines are just a special case of the more general class of curves, subdivision curves. For example, at the beginning of this chapter we briefly outlined an interpolating subdivision method, while spline based subdivision is approximating rather than interpolating.

Why would one want to draw a spline curve by means of subdivision? In fact there is no sufficiently strong reason for using subdivision in one dimension and none of the commercial line drawing packages do so, but the argument becomes much more compelling in higher dimensions as we will see in later chapters.

In the next section we use the subdivision matrix to study the behavior of the resulting curve at a point or in the neighborhood of a point. We will see that it is quite easy, for example, to evaluate the curve exactly at a point, or to compute a tangent vector, simply from a deeper understanding of the subdivision matrix.

2.4 Analysis of Subdivision

In the previous section we have shown that uniform spline curves can be thought of as a special case of subdivision curves. So far, we have seen only examples for which we use a fixed set of coefficients to

compute the control points everywhere. The coefficients define the appearance of the curve, for example, whether it is differentiable or has sharp corners. Consequently it is possible to control the appearance of the curve by modifying the subdivision coefficients locally. So far we have not seen a compelling reason to do so in the 1D setting. However, in the surface setting it will be essential to change the subdivision rule locally around extraordinary vertices to ensure maximal order of continuity. But before studying this question we once again look at the curve setting first since the treatment is considerably easier to follow in that setting.

To study properties such as differentiability of the curve (or surface) we need to understand which of the control points influences the neighborhood of the point of interest. This notion is captured by the concept of invariant neighborhoods to which we turn now.

2.4.1 Invariant Neighborhoods

Suppose we want to study the limit curve of a given subdivision scheme in the vicinity of a particular control point.[3] To determine *local* properties of a subdivision curve, we do not need the whole infinite vector of control points or the infinite matrix describing subdivision of the entire curve. Differentiability, for example, is a local property of a curve. To study it we need consider only an arbitrarily small piece of the curve around the origin. This leads to the question of which control points influence the curve in the neighborhood of the origin?

As a first example consider cubic B-spline subdivision. There is one cubic segment to the left of the origin with parameter values $t \in [-1,0]$ and one segment to the right with parameter range $t \in [0,1]$. Figure 2.6 illustrates that we need 5 control points at the coarsest level to reach any point of the limit curve which is associated with a parameter value between -1 and 1, no matter how close it is to the origin. We say that the *invariant neighborhood* has size 5. This size depends on the number of non-zero entries in each row of the subdivision matrix, which is 2 for odd points and 3 for even points. The latter implies that we need one extra control point to the left of -1 and one to the right of 1.

Another way to see this argument is to consider the basis functions associated with a given subdivision scheme. Once those are found we can find all basis functions overlapping a region of interest and their control points will give us the control set for that region. How do we find these basis functions in the setting when we don't necessarily produce B-splines through subdivision? The argument is straightforward

[3] Here and in the following we assume that the point of interest is the origin. This can always be achieved through renumbering of the control points.

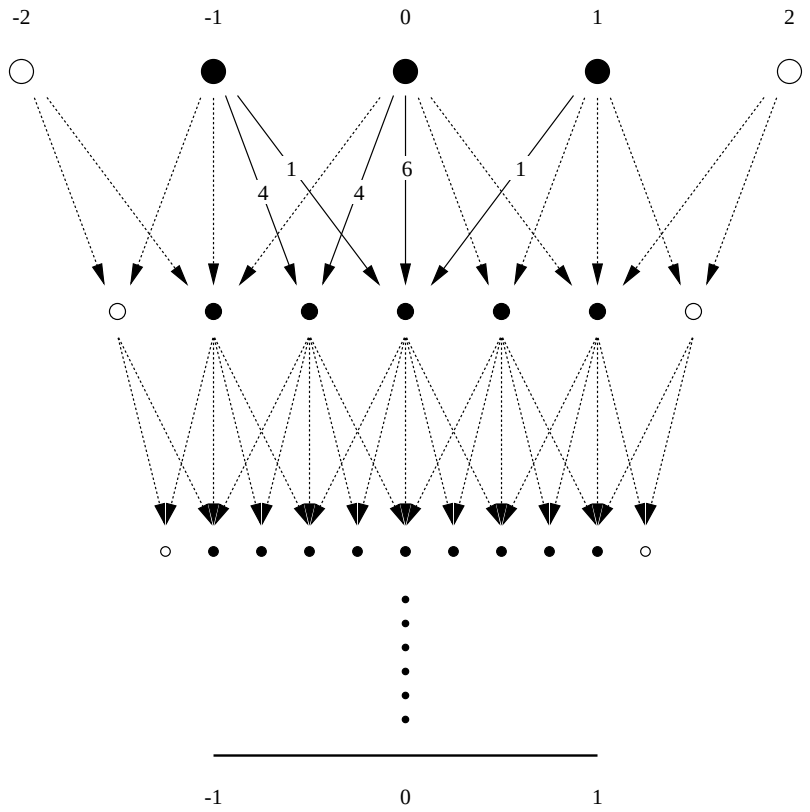


Figure 2.6: *In the case of cubic B-spline subdivision the invariant neighborhood is of size 5. It takes 5 control points at the coarsest level to determine the behavior of the subdivision limit curve over the two segments adjacent to the origin. At each level we need one more control point on the outside of the interval* $t \in [-1,1]$ *in order to continue on to the next subdivision level. 3 initial control points for example would not be enough.*

and also applies to surfaces. Recall that the subdivision operator is linear, i.e.,

$$\begin{aligned} P^j(t) &= \mathbf{B_1}(2^j t) S^j \mathbf{p}^0 \\ &= \mathbf{B_1}(2^j t) S^j \left(\sum_i p_i^0 (\mathbf{e_i})^0 \right) \\ &= \sum_i p_i^0 \mathbf{B_1}(2^j t) S^j (\mathbf{e_i})^0 \\ &= \sum_i p_i^0 \varphi_i^j(t) \end{aligned}$$

In this expression $\mathbf{e_i}^0$ stands for the vector consisting of all 0s except a single 1 in position i. In other

words the final curve is always a linear combination with weights p_i^0 of *fundamental solutions*

$$\lim_{j\to\infty} \varphi_i^j(t) = \varphi_i(t).$$

If we used the same subdivision weights throughout the domain it is easy to see that $\varphi_i(t) = \varphi(t-i)$, i.e., there is a single function $\varphi(t)$ such that all curves produced through subdivision from some initial sequence of points $\mathbf{p}^0$ are linear combinations of translates of $\varphi(t)$. This function is called the fundamental solution of the subdivision scheme. Questions such as differentiability of the limit curve can now be studied by examining this one function

$$\varphi(t) = \lim_{j\to\infty} S^j(\mathbf{e_0})^0.$$

For example, we can read off from the support of this function how far the influence of a control point will be felt. Similarly, the shape of this function tells us something about how the curve (or surface) will change when we pull on a control point. Note that in the surface case the rules we apply will depend on the valence of the vertex in question. In that case we won't get only a single fundamental solution, but a different one for each valence. More on this later.

With this we can revisit the argument for the size of the invariant neighborhood. The basis functions of cubic B-spline subdivision have support width of 4 intervals. If we are interested in a small open neighborhood of the origin we notice that 5 basis functions will overlap that small neighborhood. The fact that the central 5 control points control the behavior of the limit curve at the origin holds independent of the level. With the central 5 control points at level j we can compute the central 5 control points at level $j+1$. This implies that in order to study the behavior of the curve at the origin all we have to analyze is a small 5×5 subblock of the subdivision matrix

$$\begin{pmatrix} p_{-2}^{j+1} \\ p_{-1}^{j+1} \\ p_0^{j+1} \\ p_1^{j+1} \\ p_2^{j+1} \end{pmatrix} = \frac{1}{8} \begin{pmatrix} 1 & 6 & 1 & 0 & 0 \\ 0 & 4 & 4 & 0 & 0 \\ 0 & 1 & 6 & 1 & 0 \\ 0 & 0 & 4 & 4 & 0 \\ 0 & 0 & 1 & 6 & 1 \end{pmatrix} \begin{pmatrix} p_{-2}^{j} \\ p_{-1}^{j} \\ p_0^{j} \\ p_1^{j} \\ p_2^{j} \end{pmatrix}.$$

The 4 point subdivision scheme provides another example. This time we do not have recourse to splines to argue the properties of the limit curve. In this case each basis function has a support ranging over 6 intervals. An easy way to see this is to start with the sequence $\mathbf{e_0}^0$, i.e., a single 1 at the origin surrounded by zeros. Repeatedly applying subdivision we can see that no points outside the original $[-3,3]$ interval will become non-zero. Consequently for the invariant neighborhood of the origin we

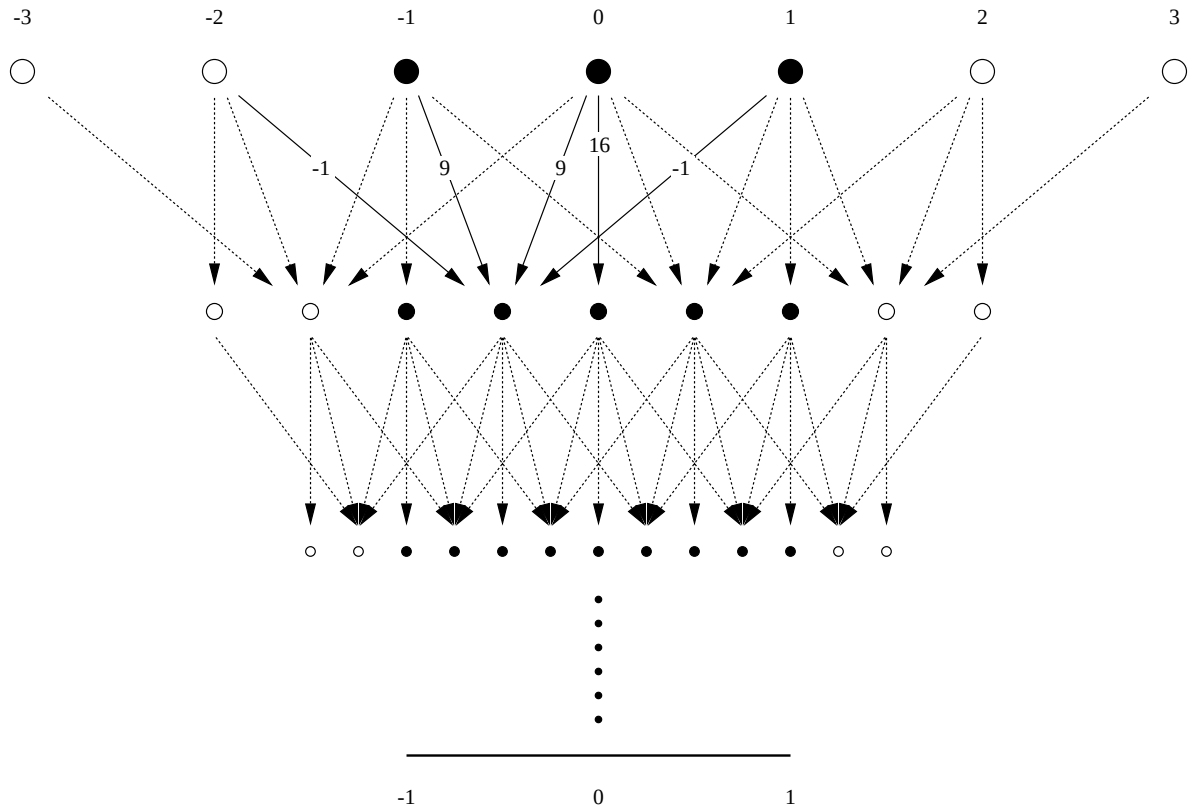


Figure 2.7: *In the case of the 4 point subdivision rule the invariant neighborhood is of size 7. It takes 7 control points at the coarsest level to determine the behavior of the subdivision limit curve over the two segments adjacent to the origin. One extra point at p_2^j is needed to compute p_1^{j+1}. The other is needed to compute p_3^{j+1}, which requires p_3^j. Two extra points on the left and right result in a total of 7 in the invariant neighborhood.*

need to consider 3 basis functions to the left, the center function, and 3 basis functions to the right. The 4 point scheme has an invariant neighborhood of 7 (see Figure 2.7). In this case the local subdivision

matrix is given by

$$\begin{pmatrix} p_{-3}^{j+1} \\ p_{-2}^{j+1} \\ p_{-1}^{j+1} \\ p_{0}^{j+1} \\ p_{1}^{j+1} \\ p_{2}^{j+1} \\ p_{3}^{j+1} \end{pmatrix} = \frac{1}{16} \begin{pmatrix} -1 & 9 & 9 & -1 & 0 & 0 & 0 \\ 0 & 0 & 16 & 0 & 0 & 0 & 0 \\ 0 & -1 & 9 & 9 & -1 & 0 & 0 \\ 0 & 0 & 0 & 16 & 0 & 0 & 0 \\ 0 & 0 & -1 & 9 & 9 & -1 & 0 \\ 0 & 0 & 0 & 0 & 16 & 0 & 0 \\ 0 & 0 & 0 & -1 & 9 & 9 & -1 \end{pmatrix} \begin{pmatrix} p_{-3}^{j+1} \\ p_{-2}^{j+1} \\ p_{-1}^{j+1} \\ p_{0}^{j+1} \\ p_{1}^{j+1} \\ p_{2}^{j+1} \\ p_{3}^{j+1} \end{pmatrix}$$

Since the local subdivision matrix controls the behavior of the curve in a neighborhood of the origin, it comes as no surprise that many properties of curves generated by subdivision can be inferred from the properties of the local subdivision matrix. In particular, differentiability properties of the curve are related to the eigen structure of the local subdivision matrix to which we now turn. From now on the symbol S will denote the *local* subdivision matrix.

2.4.2 Eigen Analysis

Recall from linear algebra that an *eigenvector* $\mathbf{x}$ of the matrix M is a non-zero vector such that $M\mathbf{x} = \lambda\mathbf{x}$, where λ is a scalar. We say that λ is the *eigenvalue* corresponding to the right eigenvector $\mathbf{x}$.

Assume the local subdivision matrix S has size $n \times n$ and has real eigenvectors $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$, which form a basis, with corresponding real eigenvalues $\lambda_0 \geq \lambda_1 \geq \ldots \geq \lambda_{n-1}$. For example, in the case of cubic splines $n = 5$ and

$$\begin{aligned} (\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4) &= (1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}) \\ (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) &= \begin{pmatrix} 1 & -1 & 1 & 1 & 0 \\ 1 & -\frac{1}{2} & \frac{2}{11} & 0 & 0 \\ 1 & 0 & -\frac{1}{11} & 0 & 0 \\ 1 & \frac{1}{2} & \frac{2}{11} & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}. \end{aligned}$$

Given these eigenvectors we have

$$\begin{aligned} S(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{x}_4) &= (\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{x}_4)\begin{pmatrix} \lambda_0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & 0 & \lambda_4 \end{pmatrix} \\ SX &= XD \\ X^{-1}SX &= D. \end{aligned}$$

The rows $\tilde{\mathbf{x}}_i$ of X^{-1} are called left eigenvectors since they satisfy $\tilde{\mathbf{x}}_i S = \lambda_i \tilde{\mathbf{x}}_i$, which can be seen by multiplying the last equality with X^{-1} on the right.

Note: not all subdivision schemes have only real eigenvalues or a complete set of eigenvectors. For example, the 4 point scheme has eigenvalues

$$(\lambda_0,\lambda_1,\lambda_2,\lambda_3,\lambda_4,\lambda_5,\lambda_6) = (1,\frac{1}{2},\frac{1}{4},\frac{1}{4},\frac{1}{8},-\frac{1}{16},-\frac{1}{16}),$$

but it does not have a complete set of eigenvectors. These degeneracies are the cause of much technical difficulty in the theory of subdivision. To keep our exposition simple and communicate the essential ideas we will ignore these cases and assume from now on that we have a complete set of eigenvectors.

In this setting we can write any vector $\mathbf{p}$ of length n as a linear combination of eigenvectors:

$$\mathbf{p} = \sum_{i=0}^{n-1} a_i \mathbf{x}_i,$$

where the a_i are given by the inner products $a_i = \tilde{\mathbf{x}}_i \cdot p$. This decomposition works also when the entries of $\mathbf{p}$ are *n 2-D points* (or 3-D points in the case of surfaces) rather than single numbers. In this case each "coefficient" a_i is a 2-D (3-D) point. The eigenvectors $\mathbf{x}_0,\ldots,\mathbf{x}_{n-1}$ are simply vectors of n real numbers.

In the basis of eigenvectors we can easily compute the result of application of the subdivision matrix to a vector of control points, that is, the control points on the next level

$$\begin{aligned} S\mathbf{p}^0 &= S\sum_{i=0}^{n-1} a_i \mathbf{x}_i \\ &= \sum_{i=0}^{n-1} a_i S\mathbf{x}_i \quad \text{by linearity of } S \\ &= \sum_{i=0}^{n-1} a_i \lambda_i \mathbf{x}_i \end{aligned}$$

Applying S j times, we obtain

$$\mathbf{p}^j = S^j \mathbf{p}^0 = \sum_{i=0}^{n-1} a_i \lambda_i^j \mathbf{x}_i.$$

2.4.3 Convergence of Subdivision

If $\lambda_0 > 1$, then $S^j \mathbf{x}^0$ would grow without bound as j increased and subdivision would not be convergent. Hence, we can see that in order for the sequence $S^j \mathbf{p}^0$ to converge at all, it is necessary that all eigenvalues are at most 1. It is also possible to show that only a single eigenvalue may have magnitude 1 [33].

A simple consequence of this analysis is that we can compute the limit position directly in the eigen basis

$$P^\infty(0) = \lim_{j\to\infty} S^j \mathbf{p}^0 = \lim_{j\to\infty} \sum_{i=0}^{n-1} a_i \lambda_i^j \mathbf{x}_i = a_0,$$

since all eigen components $|\lambda_i| < 1$ decay to zero. For example, in the case of cubic B-spline subdivision we can compute the limit position of p_i^j as $a_0 = \tilde{\mathbf{x}}_0 \cdot \mathbf{p}^j$, which amounts to

$$p_i^\infty = a_0 = \frac{1}{6}(p_{i-1}^j + 4p_i^j + p_{i+1}^j).$$

Note that this expression is completely independent of the level j at which it is computed.

2.4.4 Invariance under Affine Transformations

If we moved all the control points simultaneously by the same amount, we would expect the curve defined by these control points to move in the same way as a rigid object. In other words, the curve should be *invariant under distance-preserving transformations, such as translation and rotation*. It follows from linearity of subdivision that if subdivision is invariant with respect to distance-preserving transformations, it also should be invariant under any affine transformations. The family of affine transformations in addition to distance-preserving transformations, contains shears.

Let $\mathbf{1}$ be an n-vector of 1's and $a \in \mathbf{R}^2$ a displacement in the plane (see Figure 2.8) Then $\mathbf{1} \cdot a$ represents a displacement of our seven points by a vector a. Applying subdivision to the transformed points, we get

$$\begin{aligned} S(\mathbf{p}^j + \mathbf{1} \cdot a) &= S\mathbf{p}^j + S(\mathbf{1} \cdot a) \quad \text{by linearity of } S \\ &= \mathbf{p}^{j+1} + S(\mathbf{1} \cdot a). \end{aligned}$$

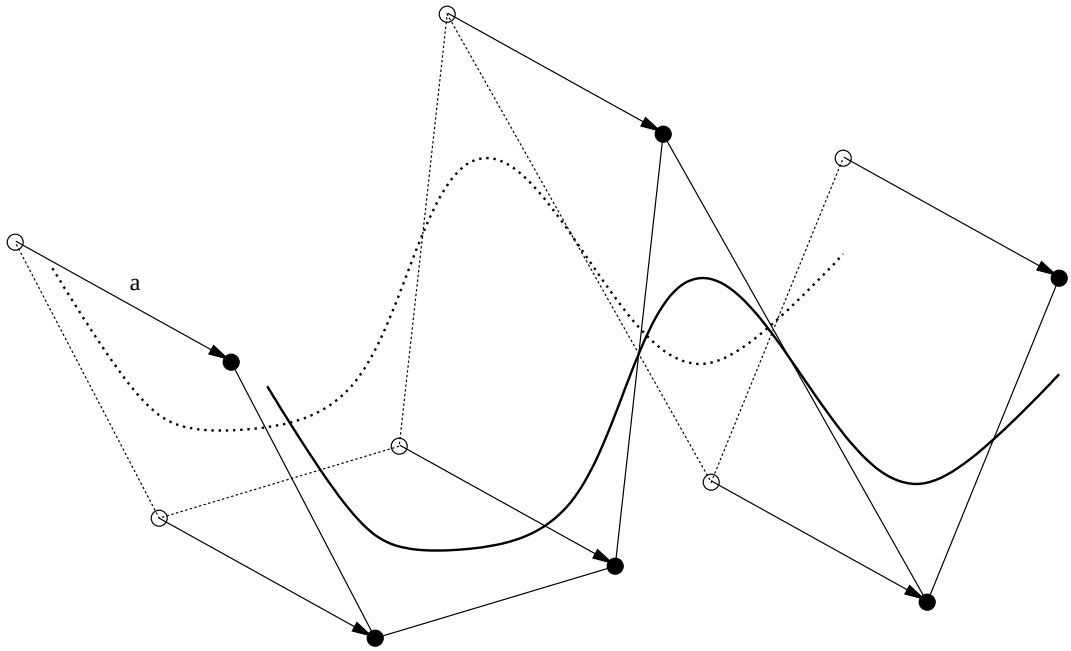


Figure 2.8: *Invariance under translation.*

From this we see that for translational invariance we need

$$S(\mathbf{1} \cdot a) = \mathbf{1} \cdot a$$

Therefore, **1** should be the eigenvector of S with eigenvalue $\lambda_0 = 1$.

Recall that when proving convergence of subdivision we assumed that **1** is an eigenvector with eigenvalue 1. We now see that this assumption is satisfied by any reasonable subdivision scheme. It would be rather unnatural if the shape of the curve changed as we translate control points.

2.4.5 Geometric Behavior of Repeated Subdivision

If we assume that λ_0 is 1, and all other eigenvalues are less than 1, we can choose our coordinate system in such a way that a_0 is the origin in $\mathbf{R}^2$. In that case we have

$$\mathbf{p}^j = \sum_{i=1}^{n-1} a_i \lambda_i^j \mathbf{x}_i$$

Dividing both sides by λ_1^j, we obtain

$$\frac{1}{\lambda_1^j}\mathbf{p}^j = a_1\mathbf{x}_1 + \sum_{i=2}^{n-1} a_i \left(\frac{\lambda_i}{\lambda_1}\right)^j \mathbf{x}_i.$$

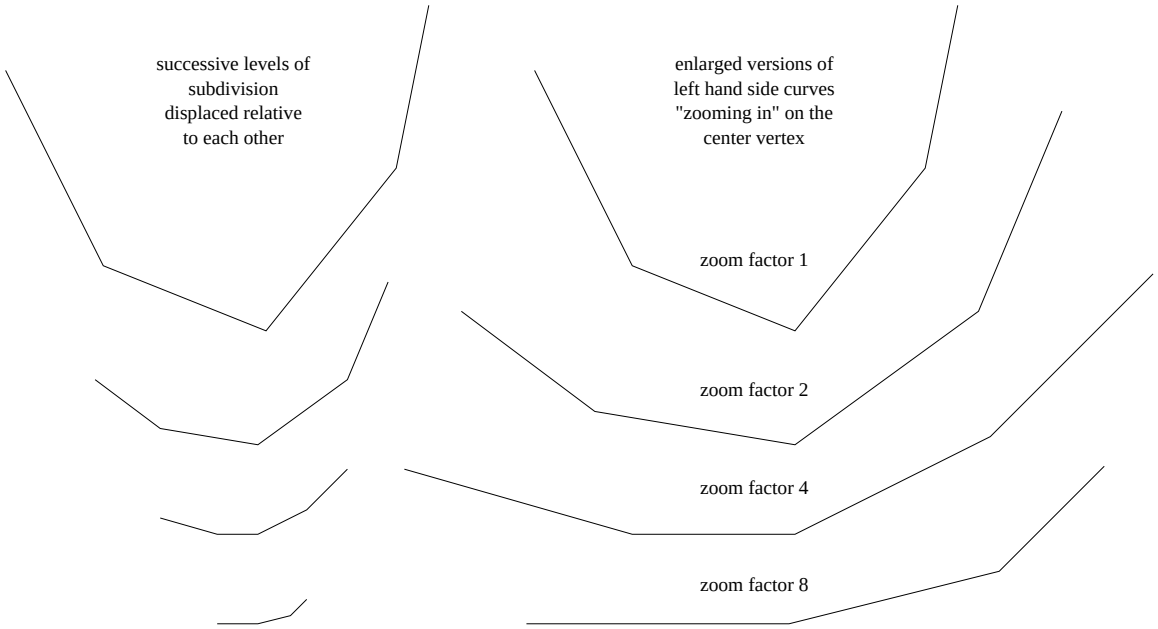


Figure 2.9: *Repeatedly applying the subdivision matrix to our set of n control points results in the control points converging to a configuration aligned with the tangent vector. The various subdivision levels have been offset vertically for clarity.*

If we assume that $|\lambda_2|, \ldots, |\lambda_{n-1}| < |\lambda_1|$, the sum on the right approaches zero as $j \to \infty$. In other words the term corresponding to λ_1 will "dominate" the behavior of the vector of control points. In the limit, we get a set of n points arranged along the vector a_1. Geometrically, this is a vector tangent to our curve at the center point (see Figure 2.9).

Just as in the case of computing the limit point of cubic B-spline subdivision by computing a_0 we can compute the tangent vector at p_i^j by computing $a_1 = \tilde{\mathbf{x}}_1 \cdot \mathbf{p}^j$

$$t_i^\infty = a_1 = p_{i+1}^j - p_{i-1}^j.$$

If there were two equal eigenvalues, say $\lambda_1 = \lambda_2$, as j increases, the points in the limit configuration will be linear combinations of two vectors a_1 and a_2, and in general would not be on the same line. This indicates that there will be no tangent vector at the central point. This leads us to the following condition, that, under some additional assumptions, is necessary for the existence of a tangent

All eigenvalues of S except $\lambda_0 = 1$ *should be less than* λ_1.

2.4.6 Size of the Invariant Neighborhood

We have argued above that the size of the invariant neighborhood for cubic splines is 5 (7 for the 4pt scheme). This was motivated by the question of which basis functions overlap a finite sized, however small, neighborhood of the origin. Yet, when we computed the limit position as well as the tangent vector for the cubic spline subdivision we used left eigenvectors, whose non-zero entries did not extend beyond the immediate neighbors of the vertex at the origin. This turns out to be a general observation. While the larger invariant neighborhood is needed for *analysis*, we can actually get away with a smaller neighborhood if we are only interested in *computation* of point positions and tangents at those points corresponding to one of the original vertices. The value of the subdivision curve at the center point only depends on those basis functions which are non-zero at that point. In the case of cubic spline subdivision there are only 3 basis functions with this property. Similarly the first derivatives at the origin of the basis functions centered at -2 and +2 are zero as well. Hence the derivative only depends on the immediate neighbors as well. This must be so since the subdivision scheme is C^1. The basis functions have zero derivative at the edge of their support by C^1-continuity assumption, because outside of the support the derivative is identically zero.

For curves this distinction does not make too much of a difference in terms of computations, but in the case of surfaces life will be much easier if we can use a smaller invariant neighborhood for the computation of limit positions and tangents. For example, for Loop's scheme we will be able to use a 1-ring (only immediate neighbors) rather than a 2-ring. For the Butterfly scheme we will find that a 2-ring, rather than a 3-ring is sufficient to compute tangents.

2.4.7 Summary

For our subdivision matrix S we desire the following characteristics

- 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.

Chapter 3

Subdivision Surfaces

Denis Zorin, New York University

In this chapter we review the basic principles of subdivision surfaces. These principles can be applied to a variety of subdivision schemes described in Chapter 4: Doo-Sabin, Catmull-Clark, Loop, Modified Butterfly, Kobbelt, Midedge.

Some of these schemes were around for a while: the 1978 papers of Doo and Sabin and Catmull and Clark were the first papers describing subdivision algorithms for surfaces. Other schemes are relatively new. Remarkably, during the period from 1978 until 1995 little progress was made in the area. In fact, until Reif's work [26] on C^1-continuity of subdivision most basic questions about the behavior of subdivision surfaces near extraordinary vertices were not answered. Since then there was a steady stream of new theoretical and practical results: classical subdivision schemes were analyzed [28, 17], new schemes were proposed [40, 10, 8, 18], and general theory was developed for C^1 and C^k-continuity of subdivision [26, 19, 35, 37]. Smoothness analysis was performed in some form for almost all known schemes, for all of them, definitive results were obtained during the last 2 years only.

One of the goals of this chapter is to provide an accessible introduction to the mathematics of subdivision surfaces (Sections 3.4 and 3.5). Building on the material of the first chapter, we concentrate on the few general concepts that we believe to be of primary importance: subdivision surfaces as parametric surfaces, C^1-continuity, eigen structure of subdivision matrices, characteristic maps.

The developments of recent years have convinced us of the importance of understanding the mathematical foundations of subdivision. A Computer Graphics professional who wishes to use subdivision, probably is not interested in the subtle points of a theoretical argument. However, understanding the

general concepts that are used to construct and analyze subdivision schemes allows one to choose the most appropriate subdivision algorithm or customize one for a specific application.

3.1 Subdivision Surfaces: an Example

One of the simplest subdivision schemes is the *Loop scheme*, invented by Charles Loop [15]. We will use this scheme as an example to introduce some basic features of subdivision for surfaces.

The Loop scheme is defined for triangular meshes. The general pattern of refinement, which we call *vertex insertion*, is shown in Figure 3.1.

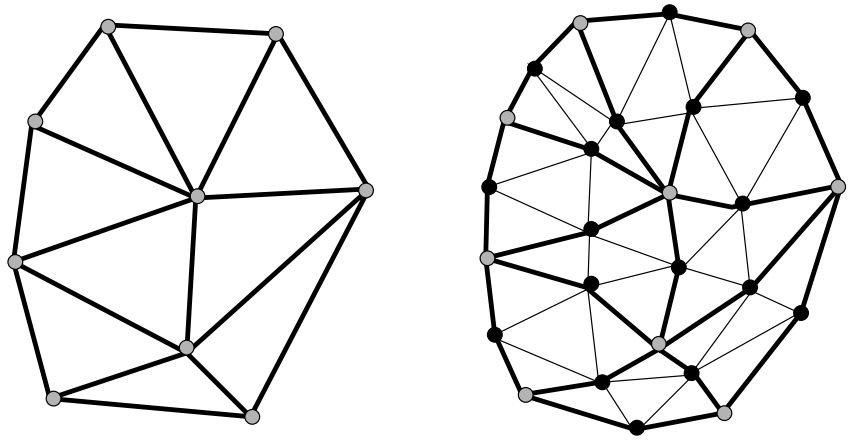


Figure 3.1: *Refinement of a triangular mesh. New vertices are shown as black dots. Each edge of the control mesh is split into two, and new vertices are reconnected to form 4 new triangles, replacing each triangle of the mesh.*

Like most (but not all) other subdivision schemes, this scheme is based on a spline basis function, called the three-directional quartic box spline. Unlike more conventional splines, such as the bicubic spline, the three-directional box spline is defined on the regular *triangular* grid; the generating polynomial for this spline is

$$S(z_1, z_2) = \frac{1}{16}(1+z_1)^2(1+z_2)^2(1+z_1z_2)^2.$$

Note that the generating polynomial for surfaces has two variables, while the generating polynomials for curves described in Chapter 2, had only one. This spline basis function is C^2-continuous. Subdivision rules for it are shown in Figure 3.2.

In one dimension, once a spline basis is chosen, all the coefficients of the subdivision rules that are

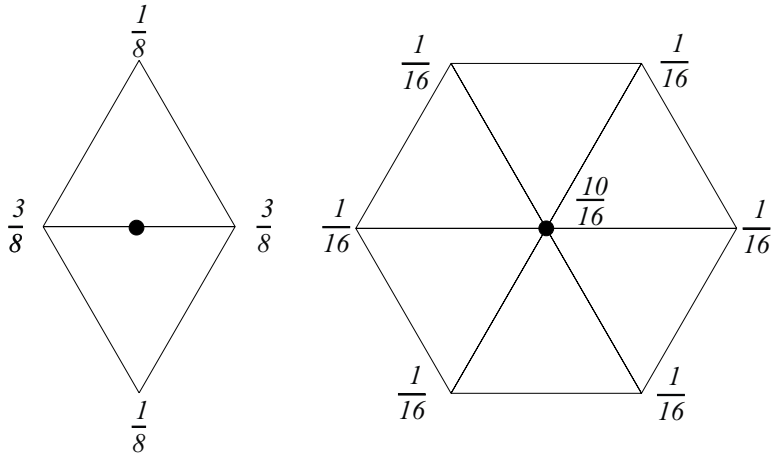


Figure 3.2: *Subdivision coefficients for a three directional box spline.*

needed to generate a curve are completely determined. The situation is radically different and more complex for surfaces. The structure of the control polygon for curves is always very simple: the vertices are arranged into a chain, and any two pieces of the chain of the same length always have identical structure. For two-dimensional meshes, the local structure of the mesh may vary: the number of edges connected to a vertex may be different from vertex to vertex. As a result the rules derived from the spline basis function may be applied only to parts of the mesh that are locally regular; that is, only to those vertices that have a valence of 6 (in the case of triangular schemes). In other cases, we have to design new rules for vertices with different valences. Such vertices are called *extraordinary*.

For the time being, we consider only meshes without a boundary. Note that the quartic box spline rule used to compute the control point inserted at an edge (Figure 3.2, left) can be applied anywhere. The only rule that needs modification is the rule used to compute new positions of control points inherited from the previous level.

Loop proposed to use coefficients shown in Figure 3.3. It turns out that this choice of coefficients guarantees that the limit surface of the scheme is "smooth."

Note that these new rules only influence local behavior of the surface near extraordinary vertices. All vertices inserted in the course of subdivision are always regular, i.e., have valence 6.

This example demonstrates the main challenge in the design of subdivision schemes for surfaces: one has to define additional rules for irregular parts of the mesh in such a way that the limit surfaces have desired properties, in particular, are smooth. In this chapter one of our main goals is to describe the conditions that guarantee that a subdivision scheme produces smooth surfaces. We start with defin-

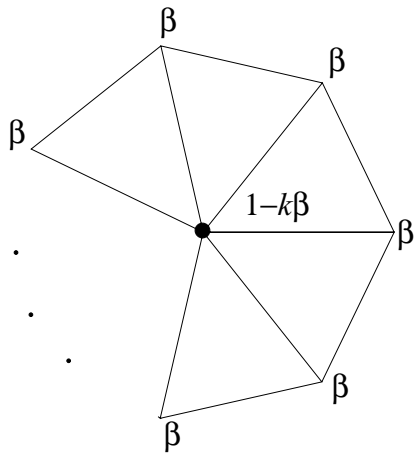


Figure 3.3: *Loop scheme: coefficients for extraordinary vertices. The choice of* β *is not unique; Loop [15] suggests* $\frac{1}{k}(5/8-(\frac{3}{8}+\frac{1}{4}\cos\frac{2\pi}{k})^2)$.

ing subdivision surfaces more rigorously (Section 3.2), and defining subdivision matrices (Section 3.3). Subdivision matrices have many applications, including computing limit positions of the points on the surface, normals, and explicit evaluation of the surface (Chapter 4). Next, we define more precisely what a smooth surface is (Section 3.4), introducing two concepts of geometric smoothness—*tangent plane continuity* and C^1*-continuity*. Then we explain how it is possible to understand local behavior of subdivision near extraordinary vertices using characteristic maps (Section 3.5). In Chapter 4 we discuss a variety of subdivision rules in a systematic way.

3.2 Natural Parameterization of Subdivision Surfaces

The subdivision process produces a sequence of polyhedra with increasing numbers of faces and vertices. Intuitively, the subdivision surface is the limit of this sequence. The problem is that we have to define what we mean by the limit more precisely. For this, and many other purposes, it is convenient to represent subdivision surfaces as functions defined on some parametric domain with values in $\mathbf{R}^3$. In the regular case, the plane or a part of the plane is the domain. However, for arbitrary control meshes, it might be impossible to parameterize the surface continuously over a planar domain.

Fortunately, there is a simple construction that allows one to use the *initial control mesh*, or more precisely, the corresponding polygonal complex, as the domain for the surface.

Parameterization over the initial control mesh. We start with the simplest case: suppose the initial control mesh is a simple polyhedron, i.e., it does not have self-intersections.

Suppose each time we apply the subdivision rules to compute the finer control mesh, we also apply midpoint subdivision to a copy of the initial control polyhedron (see Figure 3.4). This means that we leave the old vertices where they are, and insert new vertices splitting each edge in two. Note that each control point that we insert in the mesh using subdivision corresponds to a point in the midpoint-subdivided polyhedron. Another important fact is that midpoint subdivision does not alter the control polyhedron regarded as a set of points; and no new vertices inserted by midpoint subdivision can possibly coincide.

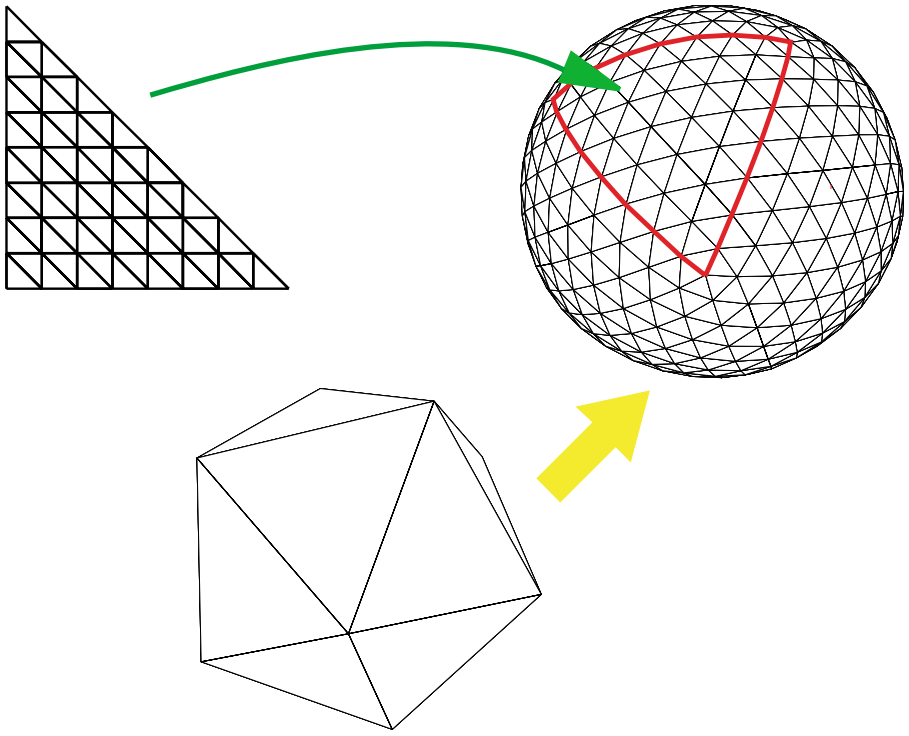


Figure 3.4: *Natural parameterization of the subdivision surface*

We will use the second copy of the control polyhedron as our domain. We denote it as K, when it is regarded as a polyhedron with identified vertices, edges and faces, and $|K|$ when it is regarded simply as a subset of $\mathbf{R}^3$.

Important remark on notation: we will refer to the points computed by subdivision as **control points**; the word **vertex** is reserved for the vertices of the polyhedron that serves as the domain and new vertices added to it by midpoint subdivision. We will use the letter v to denote vertices, and $p^j(v)$ to denote the control point corresponding to v after j subdivision steps.

As we repeatedly subdivide, we get a mapping from a denser and denser subset of the domain to the control points of a finer and finer control mesh. At each step, we linearly interpolate between control vertices, and regard the mesh generated by subdivision as a piecewise linear function on the domain K. Now we have the same situation that we had for curves: a sequence of piecewise linear functions defined on a common domain. If this sequence of functions converges uniformly, the limit is a map f from $|K|$ into $\mathbf{R}^3$. This is the limit surface of subdivision.

An important fact about the parameterization that we have just constructed is that for a regular mesh the domain can be taken to be the plane with a regular triangular grid. If in the regular case the subdivision scheme reduces to spline subdivision, our parameterization is precisely the standard (u,v) parameterization of the spline, which is guaranteed to be smooth.

To understand the general idea, this definition is sufficient, and a reader not interested in the subtle details can proceed to the next section and assume from now on that the initial mesh has no self-intersections.

General case. The crucial fact that we needed to parameterize the surface over its control polyhedron was the absence of self-intersections. Otherwise, it could happen that a vertex on the control polyhedron has more than one control point associated with it.

In general, we cannot rely on this assumption: quite often control meshes have self-intersections or coinciding control points. We can observe though that the positions of vertices of the control polyhedron are of no importance for our purposes: we can deform it in any way we want. In many cases, this is sufficient to eliminate the problem with self intersections; however, there are cases when the self-intersection cannot be removed by any deformation (example: Klein bottle, Figure 3.5). It is always possible to do that if we place our mesh in a higher-dimensional space; in fact, 4 dimensions are always enough.

This leads us to the following general choice of the domain: a polyhedron with no self-intersections, possibly in four-dimensional space. The polyhedron has to have the same structure as the initial control mesh of the surface, that is, there is a one-to-one correspondence between vertices, edges and faces of the domain and the initial control mesh. Note that now we are completely free to chose the control points of the initial mesh any way we like.

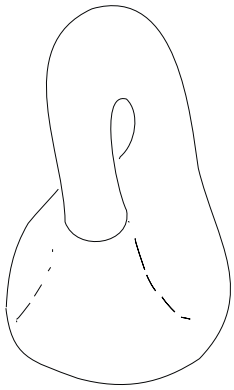


Figure 3.5: *The surface (Klein bottle) has an intersection that cannot be removed in 3D.*

3.3 Subdivision Matrix

An important tool both for understanding and using subdivision is the *subdivision matrix*, similar to the subdivision matrix for the curves introduced in Chapter 2. In this section we define the subdivision matrix and discuss how it can be used to compute tangent vectors and limit positions of points. Another application of subdivision matrices is explicit evaluation of subdivision surfaces described in Chapter 4.

Subdivision matrix. Similarly to the one-dimensional case, the subdivision matrix relates the control points in a fixed neighborhood of a vertex on two sequential subdivision levels. Unlike the one-dimensional case, there is not a single subdivision matrix for a given surface subdivision scheme: a separate matrix is defined for each valence.

For the Loop scheme control points for only two rings of vertices around an extraordinary vertex B define $f(U)$ completely. We will call the set of vertices in these two rings the *control set* of U.

Let p_0^j be the value at level j of the control point corresponding to B. Assign numbers to the vertices in the two rings (there are $3k$ vertices). Note that U^j and U^{j+1} are similar: one can establish a one-to-one correspondence between the vertices simply by shrinking U^j by a factor of 2. Enumerate the vertices in the rings; there are $3k$ vertices, plus the vertex in the center. Let p_i^j, $i = 1 \ldots 3k$ be the corresponding control points.

By definition of the control set, we can compute all values p_i^{j+1} from the values p_i^j. Because we only consider subdivision which computes finer levels by linear combination of points from the coarser level,

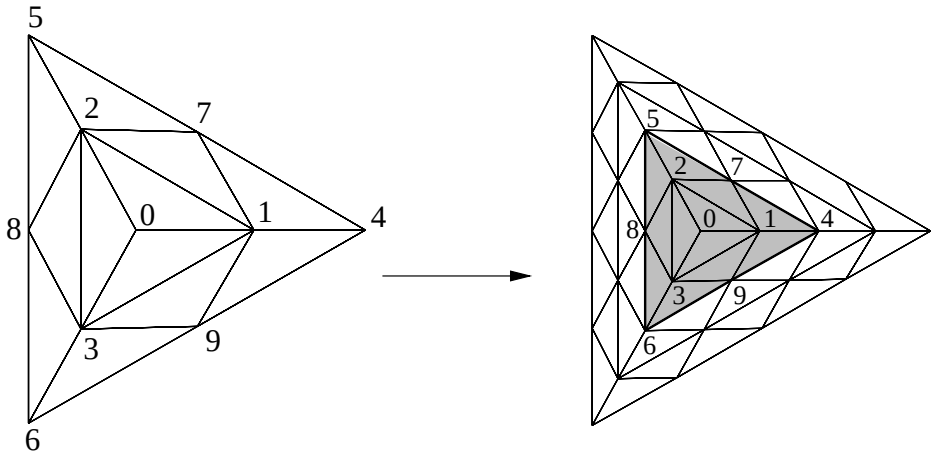


Figure 3.6: *The Loop subdivision scheme near a vertex of degree 3. Note that* $3 \times 3 + 1 = 10$ *points in two rings are required.*

the relation between the vectors of points $\mathbf{p}^{j+1}$ and $\mathbf{p}^{j}$ is given by a $(3k+1) \times (3k+1)$ matrix:

$$\begin{pmatrix} p_0^{j+1} \\ \vdots \\ p_{3k}^{j+1} \end{pmatrix} = S \begin{pmatrix} p_0^{j} \\ \vdots \\ p_{3k}^{j} \end{pmatrix}.$$

It is important to remember that each component of p^j is a point in the three-dimensional space. The matrix S is the subdivision matrix, which, in general, can change from level to level. We consider only schemes for which it is fixed. Such schemes are called *stationary*.

We can now rewrite each of the coordinate vectors in terms of the eigenvectors of the matrix S (compare to the use of eigen vectors in the 1D setting). Thus,

$$\mathbf{p}^0 = \sum_i \mathbf{a}_i x_i$$

and

$$\mathbf{p}^j = (\mathrm{S})^j \mathbf{p}^0 = \sum_i (\lambda_i)^j \mathbf{a}_i x_i$$

where the x_i are the eigenvectors of S, and the λ_i are the corresponding eigenvalues, arranged in non increasing order. As discussed for the one-dimensional case, λ_0 has to be 1 for all subdivision schemes, in order to guarantee invariance with respect to translations and rotations. Furthermore, all stable, converging subdivision schemes will have all the remaining λ_i less than 1.

Subdominant eigenvalues and eigenvectors. It is clear that as we subdivide, the behavior of $\mathbf{p}^j$, which determines the behavior of the surface in the immediate vicinity of our point of interest, will depend only on the eigenvectors corresponding to the largest eigenvalues of S.

To proceed with the derivation, we will assume for simplicity that $\lambda = \lambda_1 = \lambda_2 > \lambda_3$. We will call λ_1 and λ_2 *subdominant eigenvalues*. Furthermore, we let $\mathbf{a}_0 = 0$; this corresponds to choosing the origin of our coordinate system in the limit position of the vertex of interest (just as we did in the 1D setting). Then we can write

$$\frac{\mathbf{p}^j}{(\lambda)^j} = \mathbf{a}_1 x_1 + \mathbf{a}_2 x_2 + \mathbf{a}_3 \left(\frac{\lambda_3}{\lambda}\right)^j x_3 \dots \tag{3.1}$$

where the higher-order terms disappear in the limit.

This formula is very important, and deserves careful consideration. Recall that $\mathbf{p}^j$ is a vector of $3k+1$ 3D points, while x_i are vectors of $3k+1$ numbers. Hence the coefficients $\mathbf{a}_i$ in the decomposition above have to be 3D points.

This means that, up to a scaling by $(\lambda)^j$, the control set for $f(U)$ approaches a fixed configuration. This configuration is determined by x_1 and x_2, which depend only on the subdivision scheme, and on $\mathbf{a}_1$ and $\mathbf{a}_2$ which depend on the initial control mesh.

Each vertex in $\mathbf{p}^j$ for sufficiently large j is a linear combination of $\mathbf{a}_1$ and $\mathbf{a}_2$, up to a vanishing term. This indicates that $\mathbf{a}_1$ and $\mathbf{a}_2$ span the tangent plane. Also note that if we apply an affine transform A, taking $\mathbf{a}_1$ and $\mathbf{a}_2$ to coordinate vectors e_1 and e_2 in the plane, then, up to a vanishing term, the scaled configuration will be independent of the initial control mesh. The transformed configuration consists of 2D points with coordinates (x_{1i}, x_{2i}), $i = 0 \dots 3k$, which depend on the subdivision matrix.

Informally, this indicates that up to a vanishing term, all subdivision surfaces generated by a scheme differ near an extraordinary point only by an affine transform. In fact, this is not quite true: it may happen that a particular configuration (x_{1i}, x_{2i}), $i = 0 \dots 3k$ does not generate a surface patch, but, say, a curve. In that case, the vanishing terms will have influence on the smoothness of the surface.

Tangents and limit positions. We have observed that similar to the one-dimensional case, the coefficients $\mathbf{a}_0$ $\mathbf{a}_1$ and $\mathbf{a}_2$ in the decomposition 3.1 are the limit position of the control point for the central vertex v_0, and two tangents respectively. To compute these coefficients, we need corresponding left eigenvectors:

$$\mathbf{a}_0 = (l_0 \cdot \mathbf{p}), \quad \mathbf{a}_1 = (l_1 \cdot \mathbf{p}), \quad \mathbf{a}_0 = (l_2 \cdot \mathbf{p})$$

Similarly to the one-dimensional case, the left eigenvectors can be computed using only a smaller submatrix of the full subdivision matrix. For example, for the Loop scheme we need to consider the $k+1 \times k+1$ matrix acting on the control points of 1-neighborhood of the central vertex, not on the points of the 2-neighborhood.

In the descriptions of subdivision schemes in the next section we describe these left eigenvectors whenever information is available.

3.4 Smoothness of Surfaces

Intuitively, we call a surface smooth, if, at a close distance, it becomes indistinguishable from a plane. Before discussing smoothness of subdivision surfaces in greater detail, we have to define more precisely what we mean by a surface, in a way that is convenient for analysis of subdivision.

The discussion in the section is somewhat informal; for a more rigorous treatment, see [26, 25, 35],

3.4.1 C^1-continuity and Tangent Plane Continuity

Recall that we have defined the subdivision surface as a function $f : |K| \to \mathbf{R}^3$ on a polyhedron. Now we can formalize our intuitive notion of smoothness, namely local similarity to a piece of the plane. A surface is smooth at a point x of its domain $|K|$, if for a sufficiently small neighborhood U_x of that point the image $f(U_x)$ can be smoothly deformed into a planar disk. More precisely,

Definition 1 *A surface $f : |K| \to \mathbf{R}^3$ is **C^1-continuous**, if for every point $x \in |K|$ there exists a regular parameterization $\pi : D \to f(U_x)$ of $f(U_x)$ over a unit disk D in the plane, where U_x is the neighborhood in $|K|$ of x. A **regular parameterization** π is one that is continuously differentiable, one-to-one, and has a Jacobi matrix of maximum rank.*

The condition that the Jacobi matrix of p has maximum rank is necessary to make sure that we have no degeneracies, i.e., that we really do have a surface, not a curve or point. If $p = (p_1, p_2, p_3)$ and the disc is parameterized by x_1 and x_2, the condition is that the matrix

$$\begin{pmatrix} \frac{\partial p_1}{\partial x_1} & \frac{\partial p_1}{\partial x_2} \\ \frac{\partial p_2}{\partial x_1} & \frac{\partial p_2}{\partial x_2} \\ \frac{\partial p_3}{\partial x_1} & \frac{\partial p_3}{\partial x_2} \end{pmatrix}$$

have maximal rank (2).

There is another, weaker, definition of smoothness, which is often useful. This definition captures the intuitive idea that the tangent plane to a surface changes continuously near a smooth point. Recall that a tangent plane is uniquely characterized by its normal. This leads us to the following definition:

Definition 2 *A surface $f : |K| \to \mathbf{R}^3$ is **tangent plane continuous** at $x \in |K|$ if and only if surface normals are defined in a neighborhood around x and there exists a limit of normals at x.*

This is a useful definition, since it is easier to prove surfaces are tangent plane continuous. Tangent plane continuity, however, is weaker than C^1-continuity.

As a simple example of a surface that is tangent plane continuous but not C^1-continuous, consider the shape in Figure 3.7. Points in the vicinity of the central point are "wrapped around twice." There exists a tangent plane at that point, but the surface does not "locally look like a plane." Formally speaking, there is no regular parameterization of the neighborhood of the central point, even though it has a well-defined tangent plane.

From the previous example, we see how the definition of tangent plane continuity must be strengthened to become C^1:

Lemma 4 *If a surface is tangent plane continuous at a point and the projection of the surface onto the tangent plane at that point is one-to-one, the surface is C^1.*

The proof can be found in [35].

3.5 Analysis of Subdivision Surfaces

In this section we discuss how to determine if a subdivision scheme produces smooth surfaces. Typically, it is known in advance that a scheme produces C^1-continuous (or better) surfaces in the regular setting. For local schemes this means that the surfaces generated on arbitrary meshes are C^1-continuous away from the extraordinary vertices. We start with a brief discussion of this fact, and then concentrate on analysis of the behavior of the schemes near extraordinary vertices. Our goal is to formulate and provide some motivation for Reif's sufficient condition for C^1-continuity of subdivision.

We assume a subdivision scheme defined on a triangular mesh, with certain restrictions on the structure of the subdivision matrix, defined in Section 3.5.2. Similar derivations can be performed without these assumptions, but they become significantly more complicated. We consider the simplest case so as not to obscure the main ideas of the analysis.

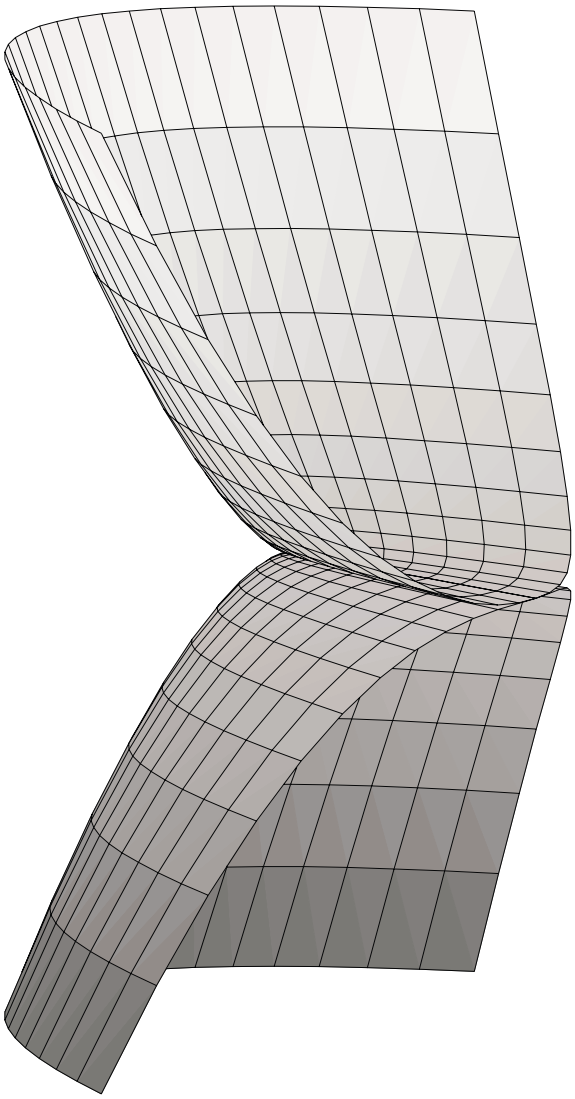


Figure 3.7: *Example of a surface that is tangent plane continuous but not C^1-continuous.*

3.5.1 C^1-continuity of Subdivision away from Extraordinary Vertices

Most subdivision schemes are constructed from regular schemes, which are known to produce at least C^1-continuous surfaces in the regular setting for almost any initial configuration of control points. If our subdivision rules are local, we can take advantage of this knowledge to show that the surfaces generated by the scheme are C^1-continuous for almost any choice of control points anywhere *away from extraor-*

dinary vertices. We call a subdivision scheme local, if only a finite number of control points is used to compute any new control point, and does not exceed a fixed number for all subdivision levels and all control points.

One can demonstrate, as we did for the curves, that for any triangle T of the domain the surface $f(T)$ is completely determined by only a finite number of control points corresponding to vertices around T. For example, for the Loop scheme, we need only control points for vertices that are adjacent to the triangle. (see Figure 3.8). This is true for triangles at any subdivision level.

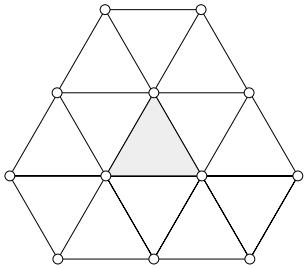


Figure 3.8: *Control set for a triangle for the three-directional box spline.*

To show this, fix a point x of the domain $|K|$ (not necessarily a vertex). For any level j, x is contained in a face of the domain; if x is a vertex, it is shared by several faces. Let $U^j(x)$ be the collection of faces on level j containing x, the *1-neighborhood* of x. The 1-neighborhood of a vertex can be identified with a k-gon in the plane, where k is the valence. We need j to be large enough so that all neighbors of triangles in $U^j(x)$ are free of extraordinary vertices. Unless x is an extraordinary vertex, this is easily achieved. $f(U^j(x))$ will be regular (see Figure 3.9).

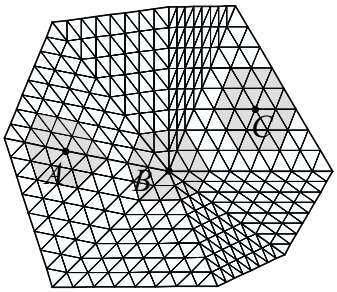


Figure 3.9: *2-neighborhoods (1-neighborhood of 1-neighborhood) of vertices A, C contain only regular vertices; this is not the case for B, which is an extraordinary vertex.*

This means that $f(U^j(x))$ is identical to a part of the surface corresponding to a regular mesh, and is therefore C^1-continuous for almost any choice of control points, because we have assumed that our

scheme generates C^1-continuous surfaces over regular meshes.[1]

3.5.2 Smoothness Near Extraordinary Vertices

Now that we know that surfaces generated by our scheme are (at least) C^1-continuous away from the extraordinary vertices, all we have to do is find a a smooth parameterization near each extraordinary vertex, or establish that no such parameterization exists.

Consider the extraordinary vertex B in Figure 3.9. After sufficient number of subdivision steps, we will get a 1-neighborhood U^j of B, such that all control points defining $f(U^j)$ are regular, except B itself. This demonstrates that it is sufficient to determine if the scheme generates C^1-continuous surfaces for a very specific type of domains K: triangulations of the plane which have a single extraordinary vertex in their center, surrounded by regular vertices. We can assume all triangles of these triangulations to be identical (see Figure 3.10) and call such triangulations k-regular.

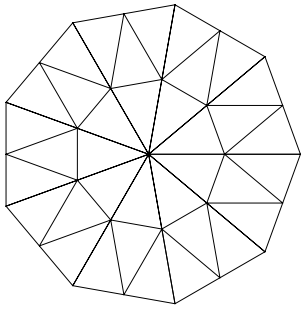


Figure 3.10: *k-regular triangulation for* $k = 9$.

At first, the task still seems to be very difficult: for any configuration of control vertices, we have to find a parameterization of $f(U^j)$. However, it turns out that the problem can be further simplified.

We outline the idea behind a *sufficient* condition for C^1-continuity proposed by Reif [26]. This criterion tells us when the scheme is guaranteed to produce C^1-continuous surfaces, but if it fails, it is still possible that the scheme might be C^1-continuous.

In addition to the subdivision matrix described in Section 3.3 , we need one more tool to formulate the criterion: the *characteristic map*. It turns out that rather than trying to consider all possible surfaces generated by subdivision, it is typically sufficient to look at a single map—the characteristic map.

[1] Our argument is informal, and there are certain unusual cases when it fails; see [35] for details.

3.5.3 Characteristic Map

Our observations made in Section 3.3 motivate the definition of the *characteristic map*. Recall that the control points near a vertex converge to a limit configuration independent, up to an affine transformation, from the control points of the original mesh. This limit configuration defines a map. Informally speaking, any subdivision surface generated by a scheme looks near an extraordinary vertex of valence k like the characteristic map of that scheme for valence k.

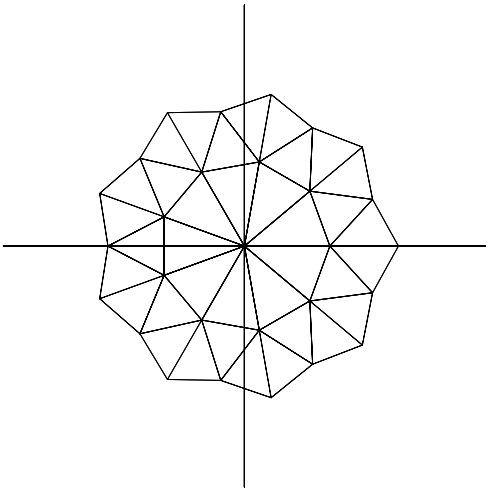


Figure 3.11: *Control set of the characteristic map for* $k = 9$.

Note that when we described subdivision as a function from the plane to $\mathbf{R}^3$, we may use control vertices not from $\mathbf{R}^3$, but from $\mathbf{R}^2$; clearly, subdivision rules can be applied in the plane rather then in space. Then in the limit we obtain a map from the plane into the plane. The characteristic map is a map of this type.

As we have seen, the configuration of control points near an extraordinary vertex approaches $\mathbf{a}_1 x_1 + \mathbf{a}_2 x_2$, up to a scaling transformation. This means that the part of the surface defined on the k-gon U^j as $j \to \infty$, and scaled by the factor $1/\lambda^j$, approaches the surface defined by the vector of control points $\mathbf{a}_1 x_1 + \mathbf{a}_2 x_2$. Let $f[\mathbf{p}] : U \to \mathbf{R}^3$ be the limit surface generated by subdivision on U from the control set p.

Definition 3 *The characteristic map of a subdivision scheme for a valence k is the map* $\Phi : U \to \mathbf{R}^2$ *generated by the vector of 2D control points* $e_1 x_1 + e_2 x_2$*:* $\Phi = f[e_1 x_1 + e_2 x_2]$*, where* e_1 *and* e_2 *are unit coordinate vectors, and* x_1 *and* x_2 *are subdominant eigenvectors.*

Regularity of the characteristic map. Inside each triangle of the k-gon U, the map is C^1: the argument of Section 3.5.1 can be used to show this. Moreover, the map has one-sided derivatives on the boundaries of the triangles, except at the extraordinary vertex, so we can define one-sided Jacobians on the boundaries of triangles too. We will say that the characteristic map is *regular* if its Jacobian is not zero anywhere on U excluding the extraordinary vertex but including the boundaries between triangles.

The regularity of the characteristic map has a geometric meaning: any subdivision surface can be written, up to a scale factor λ^j, as

$$f[\mathbf{p}^j](t) = A\Phi(t) + a(t)O\left((\lambda_3/\lambda)^j\right),$$

$t \in U^j$, $a(t)$ a bounded function $U^j \to \mathbf{R}^3$, and A is a linear transform taking the unit coordinate vectors in the plane to $\mathbf{a}_1$ and $\mathbf{a}_2$. Differentiating along the two coordinate directions t_1 and t_2 in the parametric domain U^j, and taking a cross product, after some calculations, we get the expression for the normal to the surface:

$$\mathbf{n}(t) = (\mathbf{a}_1 \times \mathbf{a}_2)J[\Phi(t)] + O\left((\lambda_3/\lambda)^{2j}\right)\tilde{a}(t)$$

where $J[\Phi]$ is the Jacobian, and $\tilde{a}(t)$ some bounded vector function on U^j.

The fact that the Jacobian does not vanish for Φ means that the normal is guaranteed to converge to $\mathbf{a}_1 \times \mathbf{a}_2$; therefore, the surface is tangent plane continuous.

Now we need to take only one more step. If, in addition to regularity, we assume that Φ is injective, we can invert it and parameterize any surface as $f(\Phi^{-1}(s))$, where $s \in \Phi(U)$. Intuitively, it is clear that up to a vanishing term this map is just an affine map, and is differentiable. We omit a rigorous proof here. For a complete treatment see [26]; for more recent developments, see [35] and [37].

We arrive at the following condition, which is the basis of smoothness analysis of all subdivision schemes considered in these notes.

Reif's sufficient condition for smoothness. Suppose the eigenvectors of a subdivision matrix form a basis, the largest three eigenvalues are real and satisfy

$$\lambda_0 = 1 > \lambda_1 = \lambda_2 > |\lambda_3|$$

If the characteristic map is regular, then almost all surfaces generated by subdivision are tangent plane continuous; if the characteristic map is also injective, then almost all surfaces generated by subdivision are C^1-continuous.

Note: Reif's original condition is somewhat different, because he defines the characteristic map on an annular region, rather than on a k-gon. This is necessary for applications, but makes it somewhat more difficult to understand.

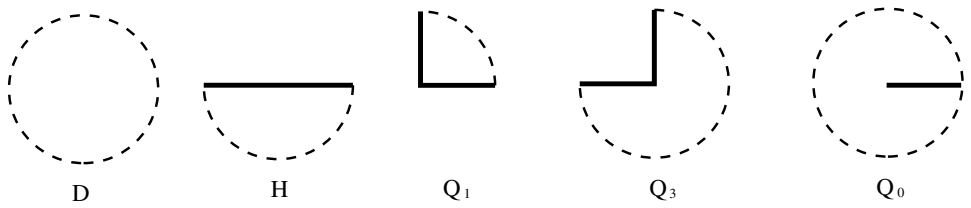


Figure 3.12: *The charts for a surface with piecewise smooth boundary.*

In Chapter 4, we will discuss the most popular stationary subdivision schemes, all of which have been proved to be C^1-continuous at extraordinary vertices. These proofs are far from trivial: checking the conditions of Reif's criterion is quite difficult, especially checking for injectivity. In most cases calculations are done in symbolic form and use closed-form expressions for the limit surfaces of subdivision [28, 8, 17, 18]. In [36] an interval-based approach is described, which does not rely on closed-form expressions for limit surfaces, and can be applied, for example, to interpolating schemes.

3.6 Piecewise-smooth surfaces and subdivision

Piecewise smooth surfaces. So far, we have assumed that we consider only closed smooth surfaces. However, in reality we typically need to model more general classes of surfaces: surfaces with boundaries, which may have corners, creases, cusps and other features. One of the significant advantages of subdivision is that it is possible to introduce features into surfaces using simple modifications of rules. Here we briefly describe a class of surfaces (*piecewise smooth surfaces*) which appears to be adequate for many applications. This is the class of surfaces that includes, for example, quadrilateral free-form patches, and other common modeling primitives. At the same time, we have excluded from consideration surfaces with various other types of singularities. To generate surfaces from this class, in addition to vertex and edge rules such as the Loop rules (Section 3.1), we need to define several other types of rules.

To define piecewise smooth surfaces, we start with smooth surfaces that have a piecewise-smooth boundary. For simplicity, assume that our surfaces do not have self-intersections. Recall that for closed C^1-continuous surface M in $\mathbf{R}^3$ each point has a neighborhood that can be smoothly deformed into an open planar disk D.

A surface with a smooth boundary is defined in a similar way, but the neighborhoods of points on the boundary can be smoothly deformed into a half-disk H, with closed boundary. To define a surface with piecewise smooth boundaries, we introduce two additional types of local charts: concave and convex corner charts, Q_3 and Q_1 (Figure 3.12). Thus, a C^1-continuous surface with piecewise smooth boundary locally looks like one of the domains D, H, Q_1 and Q_3.

Piecewise-smooth surfaces are the surfaces that can be constructed out of surfaces with piecewise smooth boundaries joined together.

If the resulting surface is not C^1-continuous at the common boundary of two pieces, this common boundary is a crease. We allow two adjacent smooth segments of a boundary to be joined, producing a crease ending in a *dart* (cf. [9]). For dart vertices an additional chart Q_0 is required; the surface near a dart can be deformed into this chart smoothly everywhere except at an open edge starting at the center of the disk.

Subdivision schemes for piecewise smooth surfaces. An important observation for constructing subdivision rules for the boundary is that the last two corner types are not equivalent, that is, there is no smooth *non-degenerate* map from Q_1 to Q_3. It follows from the theory of subdivision [35], that a single subdivision rule cannot produce both types of corners. In general, any complete set of subdivision rules should contain separate rules for all chart types. Most, if not all, known schemes provide rules for charts of type D and H (smooth boundary and interior vertices); rules for charts of type Q_1 and Q_0 (convex corners and darts) are typically easy to construct; however, Q_3 (concave corner) is more of a challenge, and no rules were known until recently.

In Chapter 4 we present descriptions of various rules for smooth (not piecewise smooth) surfaces with boundary. For extensions of the Loop and Catmull-Clark schemes including concave corner rules, see [2].

Interpolating boundaries. Quite often our goal is not just to generate a smooth surface of a given topological type approximating or interpolating an initial mesh with boundary, but to interpolate a given set of boundary or even an arbitrary set of curves. In this case, one can use a technique developed by A. Levin [12, 13, 14]. The advantage of this approach is that the interpolated curves need not be generated by subdivision; one can easily create blend subdivision surfaces with different types of parametric surfaces (for a example, NURBS).

Chapter 4

Subdivision Zoo

Denis Zorin, New York University

4.1 Overview of Subdivision Schemes

In this section we describe most known stationary subdivision schemes generating C^1-continuous surfaces on arbitrary meshes. Without doubt, our discussion is not exhaustive even as far as stationary schemes are concerned. There are even wholly different classes of subdivision schemes, most importantly variational schemes, that we do not discuss here.

At first glance, the variety of existing schemes might appear chaotic. However, there is a straightforward way to classify most of the schemes based on four criteria:

- the type of refinement rule (face split or vertex split);
- the type of generated mesh (triangular or quadrilateral);
- whether the scheme is approximating or interpolating;
- smoothness of the limit surfaces for regular meshes (C^1, C^2 etc.)

The following table shows this classification:

Face split		
	Triangular meshes	*Quad. meshes*
Approximating	Loop (C^2)	Catmull-Clark (C^2)
Interpolating	Mod. Butterfly (C^1)	Kobbelt (C^1)

Vertex split
Doo-Sabin, Midedge (C^1)
Biquartic (C^2)

Out of recently proposed schemes, $\sqrt{3}$ subdivision [11], and $4--8$ subdivision [31, 32] do not fit into this classification. In this survey, we focus on the better-known and established schemes, and this classification is sufficient for most purposes. It can be extended to include the new schemes, as discussed in Section 4.9.

The table shows that there is little replication in functionality: most schemes produce substantially different types of surfaces. Now we consider our classification criteria in greater detail.

First, we note that each subdivision scheme defined on meshes of arbitrary topology is based on a *regular subdivision scheme*, for example, one based on splines. Our classification is primarily a classification of regular subdivision schemes—once such a scheme is fixed, additional rules have to be specified only for extraordinary vertices or faces that cannot be part of a regular mesh.

Mesh type. Regular subdivision schemes act on regular control meshes, that is, vertices of the mesh correspond to regularly spaced points in the plane. However, the faces of the mesh can be formed in different ways. For a regular mesh, it is natural to use faces that are identical. If, in addition, we assume that the faces are regular polygons, it turns out that there are only three ways to choose the face polygons: we can use squares, equilateral triangles and regular hexagons. Meshes consisting of hexagons are not very common, and the first two types of tiling are the most convenient for practical purposes. These lead to two types of regular subdivision schemes: those defined for quadrilateral tilings, and those defined for triangular tilings.

Face split and vertex split. Once the tiling of the plane is fixed, we have to define how a refined tiling is related to the original tiling. There are two main approaches that are used to generate a refined tiling: one is *face split* and the other is *vertex split* (see Figure 4.1). The schemes using the first method are often called *primal*, and the schemes using the second method are called *dual*. In the first case, each face of a triangular or a quadrilateral mesh is split into four. Old vertices are retained, new vertices are inserted on the edges, and for quadrilaterals, an additional vertex is inserted for each face. In the second case, for each old vertex, several new vertices are created, one for each face adjacent to the vertex. A new face is created for each edge and old faces are retained; in addition, a new face is created for each vertex. For quadrilateral tilings, this results in tilings in which each vertex has valence 4. In the case of triangles vertex split (dual) schemes results in non-nesting hexagonal tilings. In this sense quadrilateral tilings are special: they support both primal and dual subdivision schemes easily (see also Chapter 5).

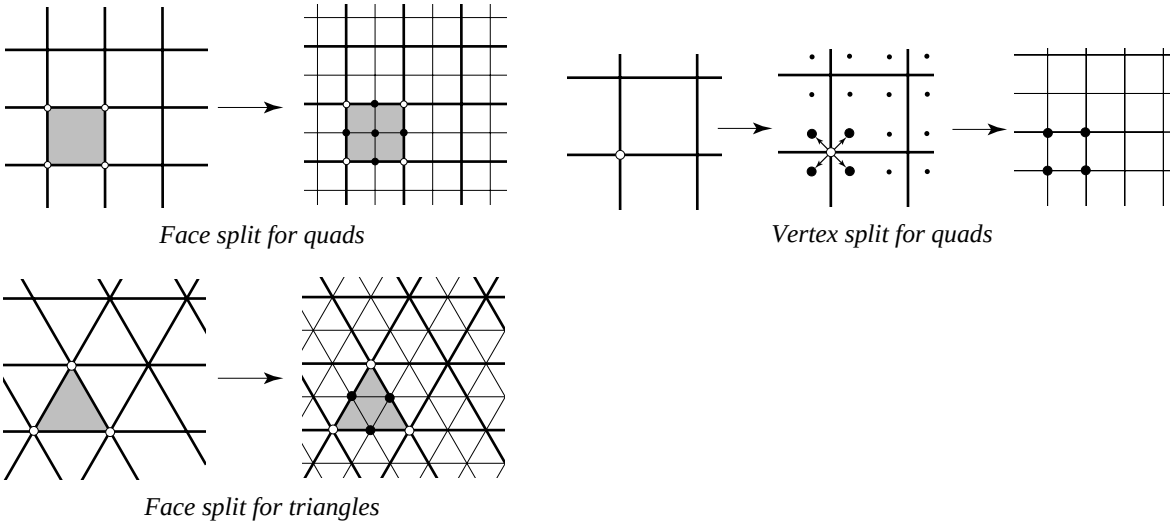


Figure 4.1: *Different refinement rules.*

Approximation vs. interpolation. Face-split schemes can be interpolating or approximating. Vertices of the coarser tiling are also vertices of the refined tiling. For each vertex a sequence of control points, corresponding to different subdivision levels, is defined. If all points in the sequence are the same, we say that the scheme is interpolating. Otherwise, we call it approximating. Interpolation is an attractive feature in more than one way. First, the original control points defining the surface are also points of the limit surface, which allows one to control it in a more intuitive manner. Second, many algorithms can be considerably simplified, and many calculations can be performed "in place." Unfortunately, the quality of these surfaces is not as high as the quality of surfaces produced by approximating schemes, and the schemes do not converge as fast to the limit surface as the approximating schemes.

4.1.1 Notation and Terminology

Here we summarize the notation that we use in subsequent sections. Some of it was already introduced earlier.

Regular and extraordinary vertices. We have already seen that subdivision schemes defined on triangular meshes create new vertices of valence 6 in the interior. On the boundary, the newly created vertices have valence 4. Similarly, on quadrilateral meshes both face-split and vertex-split schemes create only vertices of valence 4 in the interior, and 3 on the boundary. Hence, after several subdivision steps, most vertices in a mesh will have one of these valences (6 in the interior, 4 on the boundary for triangular meshes, 4 in the interior, 3 on the boundary for quadrilateral). The vertices with these valences are called *regular* and vertices of other valences *extraordinary*.

Notation for vertices near a fixed vertex. In Figure 4.2 we show the notation that we use for control points of quadrilateral and triangular subdivision schemes near a fixed vertex. Typically, we need it for extraordinary vertices. We also use it for regular vertices when describing calculations of limit positions and tangent vectors.

Odd and even vertices. For face-split (primal) schemes, the vertices of the coarser mesh are also vertices of the refined mesh. For any subdivision level, we call all new vertices that are created at that level, *odd vertices*. This term comes from the one-dimensional case, when vertices of the control polygons can be enumerated sequentially and on any level the newly inserted vertices are assigned odd numbers. The vertices inherited from the previous level are called *even*. (See also Chapter 2).

Face and edge vertices. For triangular schemes (Loop and Modified Butterfly), there is only one type of odd vertex. For quadrilateral schemes, some vertices are inserted when edges of the coarser mesh are split, other vertices are inserted for a face. These two types of odd vertices are called *edge* and *face* vertices respectively.

Boundaries and creases. Typically, special rules have to be specified on the boundary of a mesh. These rules are commonly chosen in such a way that the boundary curve of the limit surface does not depend on any interior control vertices, and is smooth or piecewise smooth (C^1 or C^2-continuous). The same rules can be used to introduce sharp features into C^1-surfaces: some interior edges can be *tagged* as crease edges, and boundary rules are applied for all vertices that are inserted on such edges.

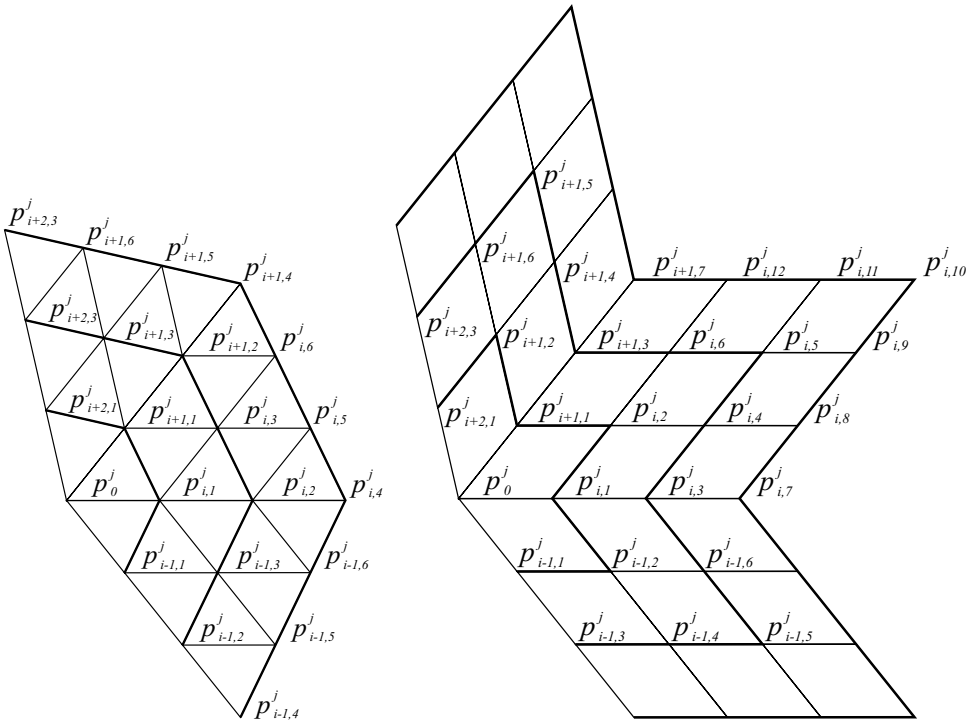


Figure 4.2: *Enumeration of vertices of a mesh near an extraordinary vertex; for a boundary vertex, the* $0-th$ *sector is adjacent to the boundary.*

Masks. We often specify a subdivision rule by providing its *mask*. The mask is a picture showing the control points used to compute a new control point, which we usually denote with a black dot. The numbers next to the vertices are the coefficients of the subdivision rule.

4.2 Loop Scheme

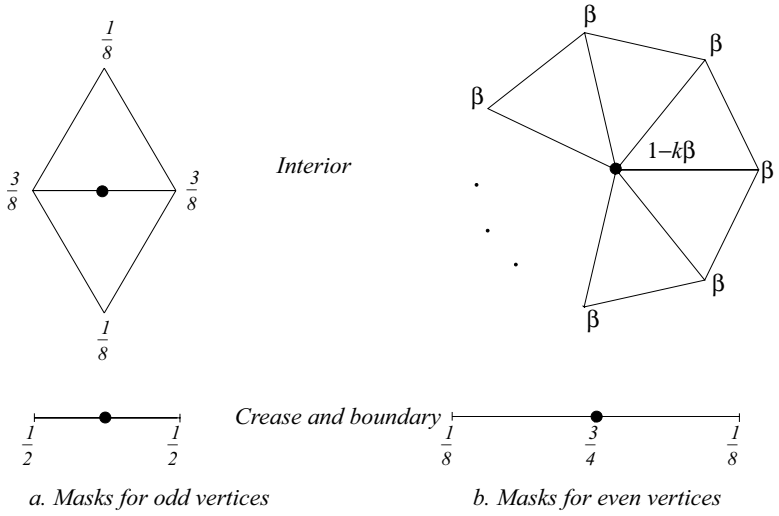
The Loop scheme is a simple approximating face-split scheme for triangular meshes proposed by Charles Loop [15]. C^1-continuity of this scheme for valences up to 100, including the boundary case, was proved by Schweitzer [28]. The proof for all valences can be found in [35].

The scheme is based on the *three-directional box spline*, which produces C^2-continuous surfaces over regular meshes. The Loop scheme produces surfaces that are C^2-continuous everywhere except at extraordinary vertices, where they are C^1-continuous. Hoppe, DeRose, Duchamp et al. [9] proposed a piecewise C^1-continuous extension of the Loop scheme, with special rules defined for edges; in [2], the

boundary rules are further improved, and new rules for concave corners and normal modification are proposed.

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.

Subdivision rules. The masks for the Loop scheme are shown in Figure 4.3. For boundaries and edges tagged as *crease* edges, special rules are used. These rules produce a cubic spline curve along the boundary/crease. The curve only depends on control points on the boundary/crease.



a. Masks for odd vertices *b. Masks for even vertices*

Figure 4.3: *Loop subdivision: in the picture above,* β *can be chosen to be either* $\frac{1}{k}\left(\frac{5}{8}-\left(\frac{3}{8}+\frac{1}{4}\cos\frac{2\pi}{k}\right)^2\right)$ *(original choice of Loop [15]), or, for* $k>3$, $\beta=\frac{3}{8k}$ *as proposed by Warren [33]. For* $k=3$, $\beta=\frac{3}{16}$ *can be used.*

In [9], the rules for extraordinary crease vertices and their neighbors on the crease were modified to produce tangent plane continuous surfaces on either side of the crease (or on one side of the boundary). In practice, this modification does not lead to a significant difference in the appearance of the surface. At the same time, as a result of this modification, the crease curve becomes dependent on the valences of vertices on the curve. This is a disadvantage in situations when two surfaces have to be joined together along a boundary. It appears that for display purposes it is safe to use the rules shown in Figure 4.3. Although the surface will not be formally C^1-continuous near vertices of valence greater than 7, the result will be visually indistinguishable from a C^1-surface obtained with modified rules, with the additional advantage of independence of the boundary from the interior.

If it is necessary to ensure C^1-continuity, a different modification can be used. Rather than modifying the rules for a crease, and making them dependent on the valence of vertices, we modify rules for interior odd vertices adjacent to an extraordinary vertex. For $k < 7$, no modification is necessary. For $k > 7$, it is sufficient to use the mask shown in Figure 4.4. Then the limit surface can be shown to be C^1-continuous at the boundary. A better, although slightly more complex modification can be found in [2]: instead of $\frac{1}{2}$ and $\frac{1}{4}$ we can use $\frac{1}{4} + \frac{1}{4}\cos\frac{\pi}{k-1}$ and $\frac{1}{2} - \frac{1}{4}\cos\frac{\pi}{k-1}$ respectively, where k is the valence of the boundary/crease vertex.

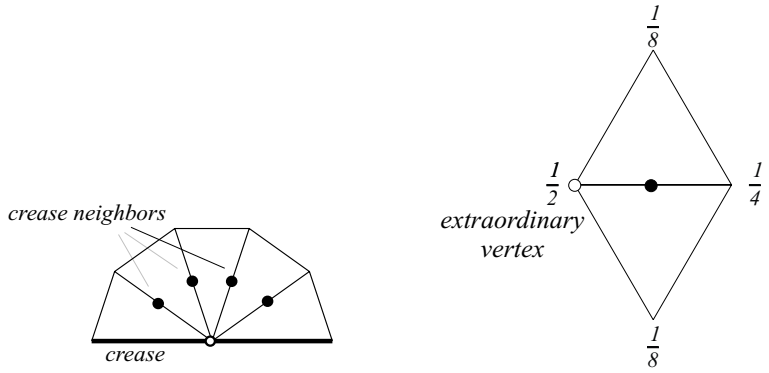


Figure 4.4: *Modified rule for odd vertices adjacent to a boundary/crease extraordinary vertex (Loop scheme).*

Tangent vectors. The rules for computing tangent vectors for the Loop scheme are especially simple. To compute a pair of tangent vectors at an interior vertex, use

$$\begin{aligned} t_1 &= \sum_{i=0}^{k-1} \cos\frac{2\pi i}{k} p_{i,1} \\ t_2 &= \sum_{i=0}^{k-1} \sin\frac{2\pi i}{k} p_{i,1}. \end{aligned} \tag{4.1}$$

These formulas can be applied to the control points at any subdivision level.

Quite often, the tangent vectors are used to compute a normal. The normal obtained as the cross product $t_1 \times t_2$ can be interpreted geometrically. This cross product can be written as a weighted sum of normals to all possible triangles formed by p_0, $p_{i,1}$, $p_{l,1}$, $i,l = 0 \ldots k-1$, $i \neq l$. The standard way of obtaining vertex normals for a mesh by averaging the normals of triangles adjacent to a vertex, can be regarded as a first approximation to the normals given by the formulas above. At the same time, it is worth observing that computing normals as $t_1 \times t_2$ is less expensive than averaging the normals of

triangles. The geometric nature of the normals obtained in this way suggests that they can be used to compute approximate normals for other schemes, even if the precise normals require more complicated expressions.

At a boundary vertex, the tangent along the curve is computed using $t_{along} = p_{0,1} - p_{k-1,1}$. The tangent across the boundary/crease is computed as follows [9]:

$$\begin{aligned} t_{across} &= p_{0,1} + p_{1,1} - 2p_0 \quad \text{for } k = 2 \\ t_{across} &= p_{2,1} - p_0 \quad \text{for } k = 3 \\ t_{across} &= \sin\theta\,(p_{0,1} + p_{k-1,1}) + (2\cos\theta - 2)\sum_{i=1}^{k-2} \sin i\theta\, p_{i,1} \quad \text{for } k \geq 4 \end{aligned} \tag{4.2}$$

where $\theta = \pi/(k-1)$. These formulas apply whenever the scheme is tangent plane continuous at the boundary; it does not matter which method was used to ensure tangent plane continuity.

Limit positions. Another set of simple formulas allows one to compute limit positions of control points for a fixed vertex, that is, the limit $\lim_{j\to\infty} p^j$ for a fixed vertex. For interior vertices, the mask for computing the limit value at an interior vertex is the same as the mask for computing the value on the next level, with β replaced by $\chi = \frac{1}{3/8\beta + k}$.

For boundary and crease vertices, the formula is always

$$p_0^\infty = \frac{1}{5}p_{0,1} + \frac{3}{5}p_0 + \frac{1}{5}p_{1,k-1}$$

This expression is similar to the rule for even boundary vertices, but with different coefficients. However, different formulas have to be used if the rules on the boundary are modified as in [9].

4.3 Modified Butterfly Scheme

The Butterfly scheme was first proposed by Dyn, Gregory and Levin in [6]. The original Butterfly scheme is defined on arbitrary triangular meshes. However, the limit surface is not C^1-continuous at extraordinary points of valence $k = 3$ and $k > 7$ [35], while it is C^1 on regular meshes.

Unlike approximating schemes based on splines, this scheme does not produce piecewise polynomial surfaces in the limit. In [40] a modification of the Butterfly scheme was proposed, which guarantees that the scheme produces C^1-continuous surfaces for arbitrary meshes (for a proof see [35]). The scheme is known to be C^1 but not C^2 on regular meshes. The masks are shown in Figure 4.5. Figure 4.5b shows

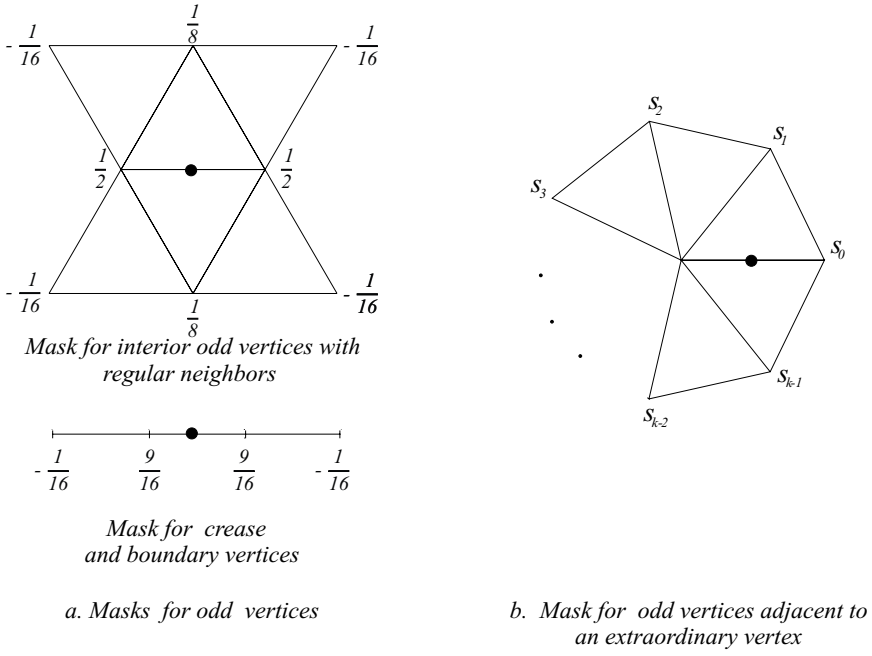


Figure 4.5: *Modified Butterfly subdivision. The coefficients s_i are $\frac{1}{k}\left(\frac{1}{4}+\cos\frac{2i\pi}{k}+\frac{1}{2}\cos\frac{4i\pi}{k}\right)$ for $k \geq 5$. For $k=3$, $s_0=\frac{5}{12}$, $s_{1,2}=-\frac{1}{12}$; for $k=4$, $s_0=\frac{3}{8}$, $s_2=-\frac{1}{8}$, $s_{1,3}=0$. The coefficient at the central vertex is $3/4$ in all cases.*

the weights for an odd vertex inserted between an extraordinary and a regular vertex. If both neighbors are extraordinary, this rule is applied to rings of vertices around each neighbor, and the result averaged.

The tangent vectors at extraordinary interior vertices can be computed using the same rules as for the Loop scheme. For regular vertices, the formulas are more complex: in this case, we have to use control points in a 2-neighborhood of a vertex. If the control points are arranged into a vector $p = [p_0, p_{0,1}, p_{1,1}, \ldots, p_{5,1}, p_{0,2}, p_{1,2}, p_{2,2}, \ldots p_{5,3}]$ of length 19, then the tangents are given by scalar products $(l_1 \cdot p)$ and $(l_2 \cdot p)$, where the vectors l_1 and l_2 are (Figure 4.6):

$$\begin{aligned} l_1 &= \left[0, 16, 8, -8, -16, -8, 8, -4, 0, 4, 4, 0, -4, 1, \frac{1}{2}, -\frac{1}{2}, -1, -\frac{1}{2}, \frac{1}{2}\right] \\ l_2 &= \sqrt{3}\left[0, 0, 8, 8, 0, -8, -8, -\frac{4}{3}, -\frac{8}{3}, -\frac{4}{3}, \frac{4}{3}, \frac{8}{3}, \frac{4}{3}, 0, \frac{1}{2}, \frac{1}{2}, 0, -\frac{1}{2}, -\frac{1}{2}\right] \end{aligned} \tag{4.3}$$

Because the scheme is interpolating, no formulas are needed to compute the limit positions: all control points are on the surface. On boundaries and creases the four-point subdivision scheme, also shown in Figure 4.5, is used [5]. To achieve C^1-continuity on the boundary, special coefficients have to be used for

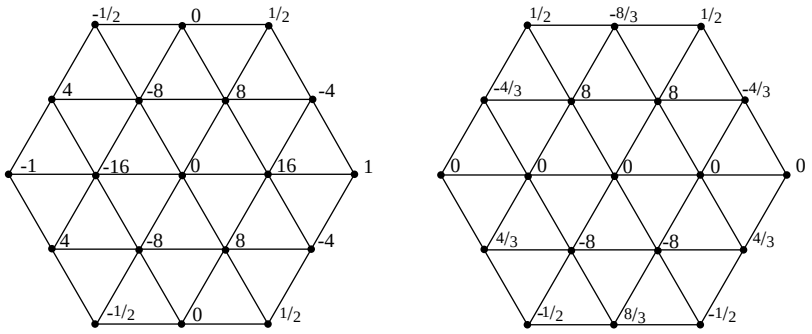


Figure 4.6: *Coefficients for computing tangents at extraordinary vertices (figure courtesy of Paul Heckbert).*

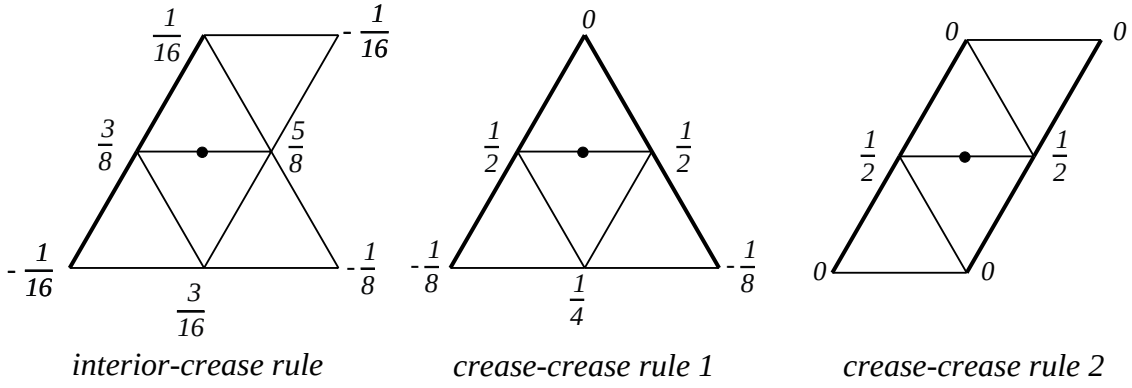
crease neighbors, similar to the case of the Loop scheme. One can also adopt a simpler solution: obtain missing vertices by reflection whenever the butterfly stencil is incomplete, and always use the standard Butterfly rule, when there is no adjacent interior extraordinary vertex. This approach however results in visible singularities. For completeness, we describe a set of rules that ensure C^1-continuity, as these rules were not previously published.

Boundary rules. The rules extending the Butterfly scheme to meshes with boundary are somewhat more complex, because the stencil of the Butterfly scheme is larger. A number of different cases have to be considered separately: first, there is a number of ways in which one can chop off triangles from the butterfly stencil; in addition, the neighbors of the vertex that we are trying to compute can be either regular or extraordinary.

A complete set of rules for a mesh with boundary (up to head-tail permutations), includes 7 types of rules: regular interior, extraordinary interior, regular interior-crease, regular crease-crease 1, regular crease-crease 2, crease, and extraordinary crease neighbor; see Figures 4.5, 4.7, and 4.8. To put it all into a system, the main cases can be classified by the types of head and tail vertices of the edge on which we add a new vertex.

Recall that an interior vertex is a regular if its valence is 6, and a crease vertex is regular if its valence is 4. The following table shows how the type of rule to be applied to compute a *non-crease* vertex is determined from the valence of the adjacent vertices and whether they are on a crease or not. As we have already mentioned, the 4-point rule is used to compute new crease vertices. The only case when additional information is necessary, is when both neighbors are regular crease vertices. In this case the decision is based on the number of crease edges of the adjacent triangles (Figure 4.7).

Head	Tail	Rule
regular interior	regular interior	standard rule
regular interior	regular crease	regular interior-crease
regular crease	regular crease	regular crease-crease 1 or 2
extraordinary interior	extraordinary interior	average two extraordinary rules
extraordinary interior	extraordinary crease	same
extraordinary crease	extraordinary crease	same
regular interior	extraordinary interior	interior extraordinary
regular interior	extraordinary crease	crease extraordinary
extraordinary interior	regular crease	interior extraordinary
regular crease	extraordinary crease	crease extraordinary



interior-crease rule *crease-crease rule 1* *crease-crease rule 2*

Figure 4.7: *Regular Modified Butterfly boundary/crease rules.*

The extraordinary crease rule (Figure 4.8) uses coefficients c_{ij}, $j = 0 \ldots k$, to compute the vertex number i in the ring, when counted from the boundary. Let $\theta_k = \pi/k$. The following formulas define c_{ij} :

$$c_0 = 1 - \frac{1}{k}\left(\frac{\sin\theta_k \sin i\theta_k}{1-\cos\theta_k}\right)$$
$$c_{i0} = -c_{ik} = \frac{\cos i\theta_k}{4} - \frac{1}{4k}\left(\frac{\sin 2\theta_k \sin 2\theta_k i}{\cos\theta_k - \cos 2\theta_k}\right)$$
$$c_{ij} = \frac{1}{k}\left(\sin i\theta_k \sin j\theta_k + \frac{1}{2}\sin 2i\theta_k \sin 2j\theta_k\right)$$

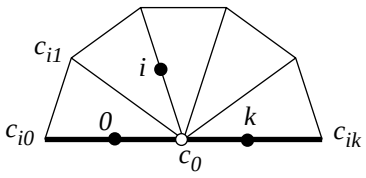


Figure 4.8: *Modified Butterfly rules for neighbors of a crease/boundary extraordinary vertex.*

4.4 Catmull-Clark Scheme

The Catmull-Clark scheme was described in [3]. It is based on the tensor product bicubic spline. The masks are shown in Figure 4.9. The scheme produces surfaces that are C^2 everywhere except at extraordinary vertices, where they are C^1. The tangent plane continuity of the scheme was analyzed by Ball and Storry [1], and C^1-continuity by Peters and Reif [17]. The values of α and β can be chosen from a wide range (see Figure 4.11). On the boundary, using the coefficients for the cubic spline produces acceptable results, however, the resulting surface formally is not C^1-continuous. A modification similar to the one performed in the case of Loop subdivision makes the scheme C^1-continuous (Figure 4.10). Again, a better, although a bit more complicated choice of coefficients is $\frac{3}{8}+\frac{1}{4}\cos\frac{2\pi}{k-1}$ instead of $\frac{5}{8}$ and $\frac{3}{8}-\frac{1}{4}\cos\frac{2\pi}{k-1}$ instead of $\frac{1}{8}$. See [38] for further details about the behavior on the boundary.

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]:

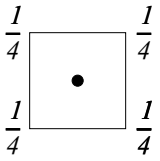
- a face control point for an n-gon is computed as the average of the corners of the polygon;
- an edge control point as the average of the endpoints of the edge and newly computed face control points of adjacent faces;
- the formula for even control points can be chosen in different ways; the original formula is

$$p_0^{j+1} = \frac{k-2}{k} p_0^j + \frac{1}{k^2}\sum_{i=0}^{k-1} p_{i,1}^j + \frac{1}{k^2}\sum_{i=0}^{k-1} p_{i,2}^{j+1}$$

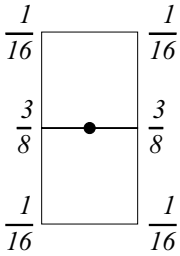
  using the notation of Figure 4.2. Note that face control points on level $j+1$ are used.

4.5 Kobbelt Scheme

This interpolating scheme was described by Kobbelt in [10]. For regular meshes, it reduces to the tensor product of the four point scheme. C^1-continuity of this scheme for interior vertices for all valences is proven in [36].

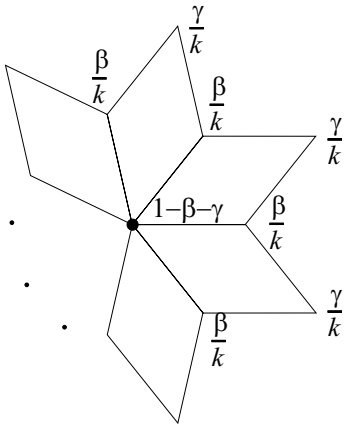


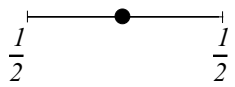
Mask for a face vertex



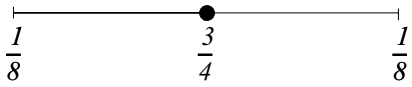
Mask for an edge vertex

Interior





Crease and boundary



Mask for a boundary odd vertex

a. Masks for odd vertices *b. Mask for even vertices*

Figure 4.9: *Catmull-Clark subdivision. Catmull and Clark [3] suggest the following coefficients for rules at extraordinary vertices:* $\beta = \frac{3}{2k}$ *and* $\gamma = \frac{1}{4k}$

Crucial for the construction of this scheme is the observation (valid for any tensor-product scheme) that the face control points can be computed in two steps: first, all edge control points are computed. Next, face vertices are computed using the *edge rule* applied to a sequence of edge control points on the same level. As shown in Figure 4.12, there are two ways to compute a face vertex in this way. In the regular case, the result is the same. Assuming this method of computing all face control points, only one rule of the regular scheme is modified: the edge odd control points adjacent to an extraordinary vertex

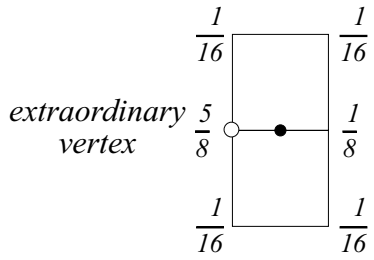


Figure 4.10: *Modified rule for odd vertices adjacent to a boundary extraordinary vertex (Catmull-Clark scheme).*

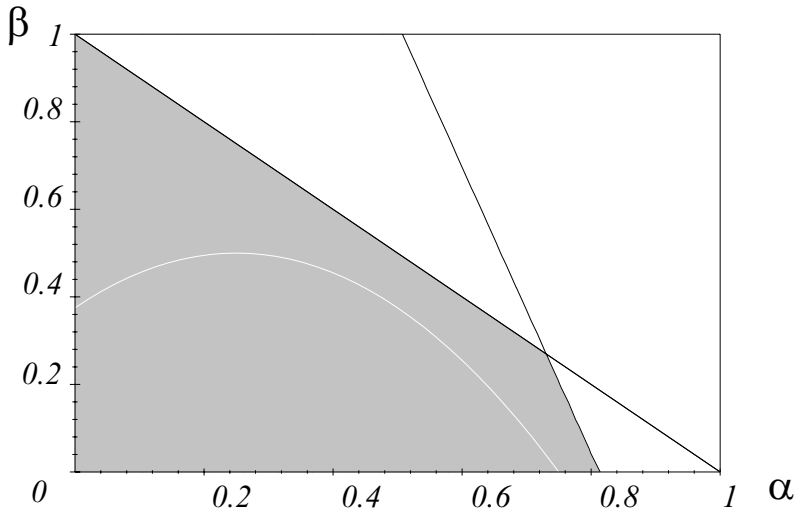


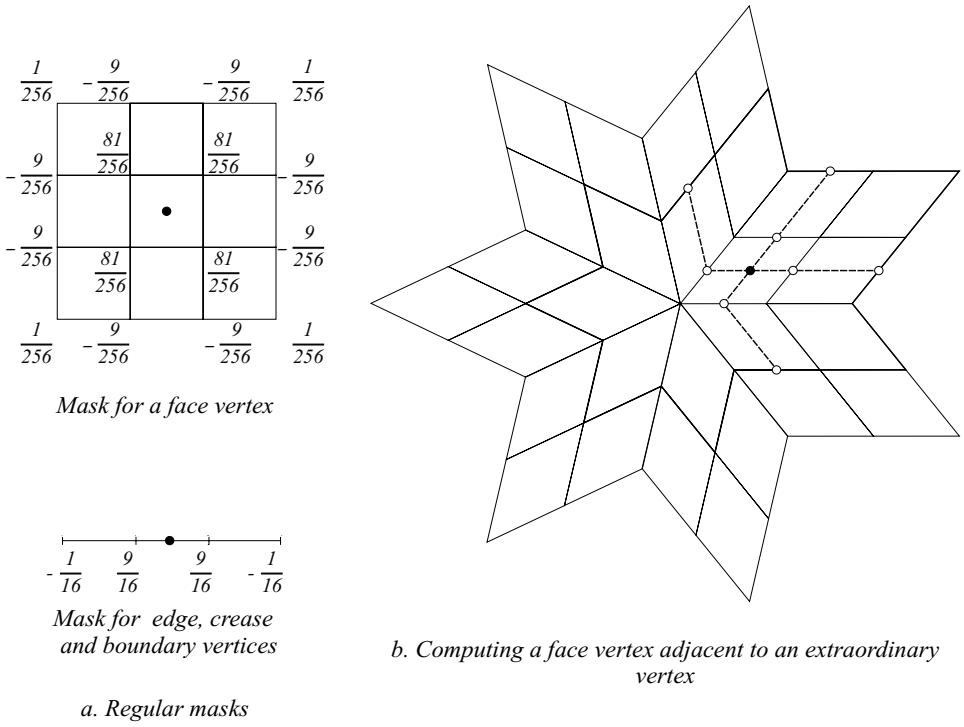
Figure 4.11: *Ranges for coefficients* α *and* β *of the Catmull-Clark scheme;* $\alpha = 1-\gamma-\beta$ *is the coefficient of the central vertex.*

are computed differently. Specifically,

$$p_{i,1}^{j+1} = (\frac{1}{2}-w)p_0^j + (\frac{1}{2}-w)p_{i,1}^j + wp_i^j + wp_{i,3}^j$$

$$v_i^j = \frac{4}{k}\sum_{i=0}^{k-1} p_{i,1}^j - (p_{i-1,1}^j + p_{i,1}^j + p_{i+1,1}^j) - \frac{w}{1/2-w}(p_{i-2,2}^j + p_{i-1,2}^j + p_{i,2}^j + p_{i+1,2}^j) + \frac{4w}{(1/2-w)k}\sum_{i=0}^{k-1} p_{i,2}^j \tag{4.4}$$

where $w = -1/16$ (also, see Figure 4.2 for notation). On the boundaries and creases, the four point subdivision rule is used.



Mask for a face vertex

Mask for edge, crease and boundary vertices

a. Regular masks

b. Computing a face vertex adjacent to an extraordinary vertex

Figure 4.12: *Kobbelt subdivision.*

Unlike other schemes, eigenvectors of the subdivision matrix cannot be computed explicitly; hence, there are no precise expressions for tangents. In any case, the effective support of this scheme is too large for such formulas to be of practical use: typically, it is sufficient to subdivide several times and then use, for example, the formulas for the Loop scheme (see discussion in the section on the Loop scheme).

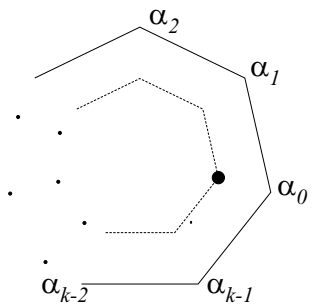
For more details on this scheme, see [10].

4.6 Doo-Sabin and Midedge Schemes

Doo-Sabin subdivision is quite simple conceptually: there is no distinction between odd and even vertices, and a single mask is sufficient to define the scheme. A special rule is required only for the boundaries, where the limit curve is a quadratic spline. It was observed by Doo that this can also be achieved by replicating the boundary edge, i.e., creating a quadrilateral with two coinciding pairs of vertices. Nasri [16] describes other ways of defining rules for boundaries. The rules for the Doo-Sabin scheme are shown in Figure 4.13. C^1-continuity for schemes similar to the Doo-Sabin schemes was analyzed by

Peters and Reif [17].

An even simpler scheme was proposed by Habib and Warren [8] and by Peters and Reif [18]: this scheme uses even smaller stencils than the Doo-Sabin scheme; for regular vertices, only three control points are used (Figure 4.14).



Mask for interior vertices

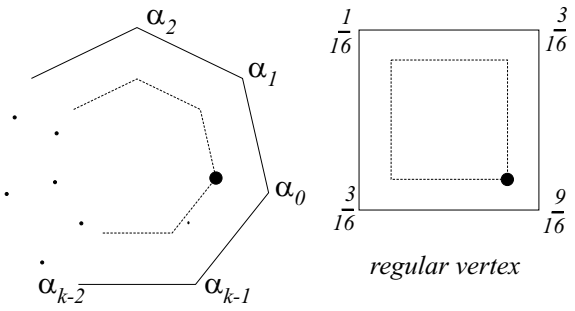


Mask for boundary vertices

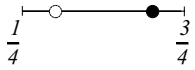
Figure 4.13: *Doo-Sabin subdivision. The coefficients are defined by the formulas* $\alpha_0 = 1/4 + 5/4k$ *and* $\alpha_i = (3 + 2\cos(2i\pi/k))/4k$*, for* $i = 1 \ldots k-1$*. Another choice of coefficients was proposed by Catmull and Clark:* $\alpha_0 = 1/2 + 1/4k$*,* $\alpha_1 = \alpha_{k-1} = 1/8 + 1/4k$*, and* $\alpha_i = 1/4k$ *for* $i = 2 \ldots k-2$*.*

A remarkable property of both Midedge and Doo-Sabin subdivision is that the interior rules, at least in the regular case, can be decomposed into a sequence of averaging steps, as shown in Figures 4.15 and Figures 4.16

In both cases the averaging procedure generalizes to arbitrary meshes. However, the edge averaging procedure, as it was established in [18], does not result in well-behaved surfaces, when applied to arbitrary meshes. In contrast, centroid averaging, when applied to arbitrary meshes, results precisely in the Catmull-Clark variant of the Doo-Sabin scheme. Another important observation is that centroid averaging can be applied more than once. This idea provides us with a different view of a class of quadrilateral subdivision schemes, which we now discuss in detail.



Mask for interior vertices



Mask for boundary vertices

Figure 4.14: *Midedge subdivision. The coefficients are defined by the formulas* $\alpha_i = 2\sum_{j=0}^{\bar{n}} 2^{-ji}\cos\frac{2\pi ij}{k}$, $\bar{n} = \lfloor\frac{n-1}{2}\rfloor$ *for* $i = 0\ldots k-1$

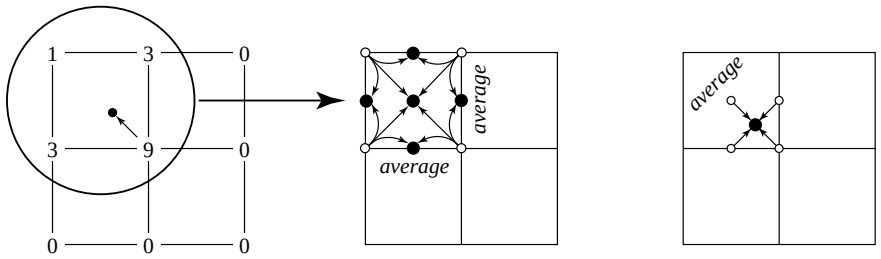


Figure 4.15: *The subdivision stencil for Doo-Sabin subdivision in the regular case (left). It can be understood as midpoint subdivision followed by averaging. At the averaging step the centroid of each face is computed; then the barycenters are connected to obtain a new mesh. This procedure generalizes without changes to arbitrary meshes.*

4.7 Uniform Approach to Quadrilateral Subdivision

As we have observed in the previous section, the Doo-Sabin scheme can be represented as midpoint subdivision followed by a centroid averaging step. What if we apply the centroid averaging step one more time? The result is a primal subdivision scheme, in the regular case coinciding with Catmull-Clark. In the irregular case the stencil of the resulting scheme is the same as the stencil of Catmull-Clark, but the coefficients α and β used in the vertex rule are different. However, the new coefficients also result in

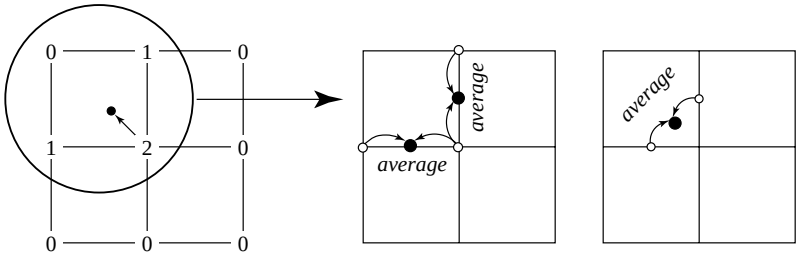


Figure 4.16: *The subdivision stencil for Midedge subdivision in the regular case (left). It can be understood as a sequence of averaging steps; at each step, two vertices are averaged.*

a well-behaved scheme producing surfaces only slightly different from Catmull-Clark.

Clearly, we can apply the centroid averaging to midpoint-subdivided mesh any number of times, obtaining in the regular case splines of higher and higher degree. Similar observations were made independently by a number of people: [20, 34, 29, 30, 39].

For arbitrary meshes we will get subdivision schemes which have higher smoothness away from isolated points on the surface. Unfortunately, smoothness at the extraordinary vertices (for primal schemes) and at the centroids of faces (for dual schemes) remains, in general, C^1.

Our observations are summarized in the following table:

centroid averaging steps	scheme	smoothness in regular case
0	midpoint	C^0
1	Doo-Sabin	C^1
2	Catmull-Clark	C^2
3	Bi-Quartic	C^3
4	...	...

Biquartic subdivision scheme is a new dual scheme that is obtained by applying three centroid averaging steps after midpoint subdivision, as illustrated in Figure 4.17. As this scheme was not discussed before, we discuss it in greater detail here.

Generalized Biquartic subdivision. The centroid averaging steps provide a nice theoretical way of deriving a new scheme, however, in practice we may want to use the complete masks directly (in particular, if we have to implement adaptive subdivision). Figure 4.17 shows the support of the stencil for Biquartic b-spline subdivision in the regular case (leftmost stencil).

Note that Biquartic subdivision can be implemented with very little additional work, compared to Doo-Sabin or Midedge. In an implementation of dual subdivision, vertices are organized as quadtrees. It

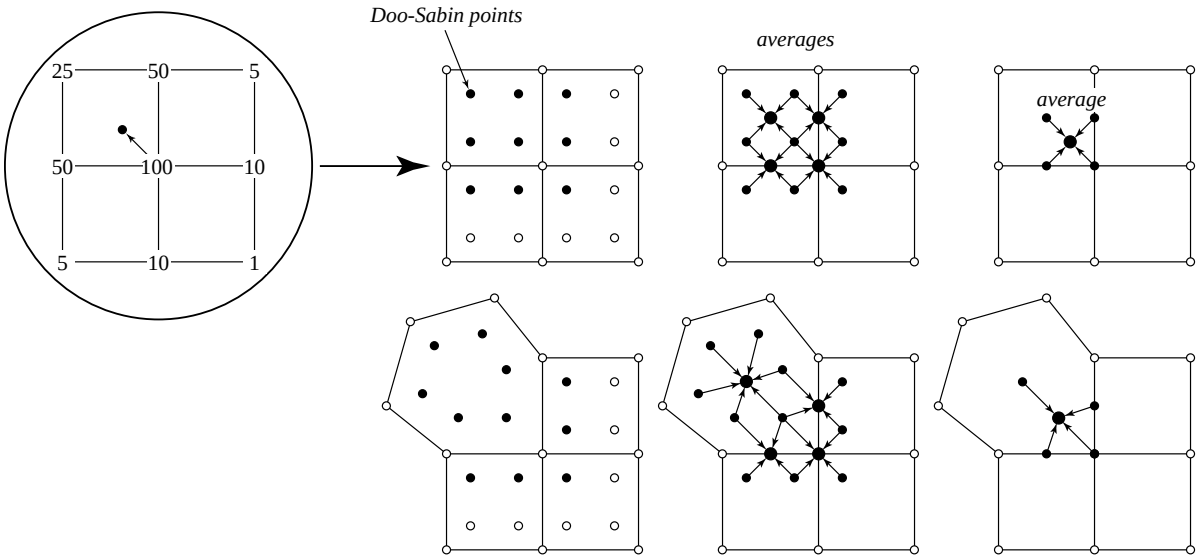


Figure 4.17: *The subdivision stencil for bi-quartic b-splines (top row for the regular setting) can be written as a sequence of averaging steps. In a first step Doo-Sabin points are computed. These are subsequently averaged twice to arrive at the final point. This effects a factorization of the original mask (left) into a sequence of pure averaging steps. The same procedure is repeated using as an example a setting in which one incident face has valence* $\neq 4$ *(bottom row).*

is then natural to compute all four children of a given vertex at the same time. Considering the stencils for Doo-Sabin or the Midedge scheme we see that this implies access to all vertices of the faces incident to a given vertex. If these vertices have to be accessed we may as well use non-zero coefficients for all of them for each child to be computed. Qu [23] was the first to consider a generalization of the Biquartic B-splines to the arbitrary topology setting. He derived some conditions on the stencils but did not give a concrete set of coefficients. Repeated centroid averaging provides a simple way to derive the coefficients. It is possible to show that the resulting scheme is C^1 at extraordinary vertices. Assuming that only one of the incident faces for a vertex is extraordinary, we can write the subdivision masks for vertices near extraordinary faces in a more explicit form. There are three different masks for the four children (Figure 4.18). This is in contrast to the Doo-Sabin and Midedge schemes which have only one mask type for all children (modulo rotation). Vertices incident to the extraordinary faces contribute

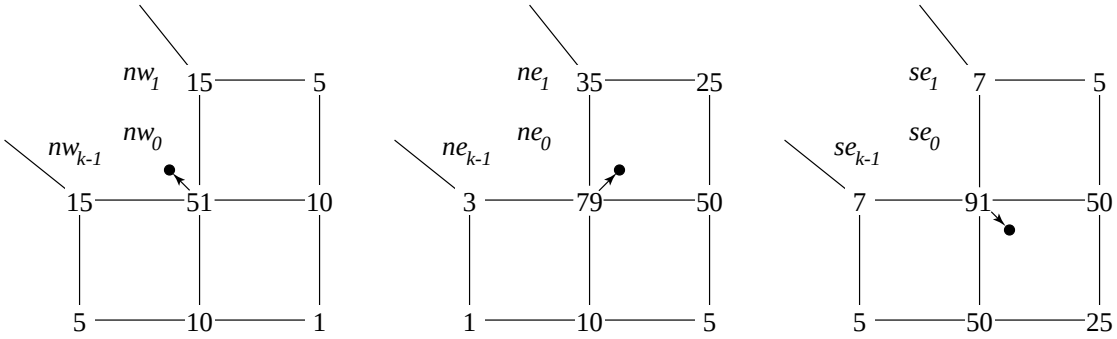


Figure 4.18: *Generalized Biquartic compound masks for the north-west (nw), north-east (ne), and south-east (se) children of the center vertex. The south-west mask is the reflected (along the diagonal) version of the ne mask. All weights must be normalized by* $1/256$ *and the weights for the extraordinary vertices must be added. They are given in equation 4.5.*

additional weights as

$$\begin{aligned} nw_i &= \frac{64}{k} + 48w_i + 16w_{i-1} + 16w_{i+1} \\ ne_i &= 32w_i + 16w_{i-1} \\ se_i &= 16w_i, \end{aligned} \tag{4.5}$$

where w_i are the Doo-Sabin weights, $i = 0, \ldots, k-1$ and indices are taken modulo k.

4.8 Comparison of Schemes

In this section we compare different schemes by applying to a variety of meshes. First, we consider Loop, Catmull-Clark, Modified Butterfly and Doo-Sabin subdivision.

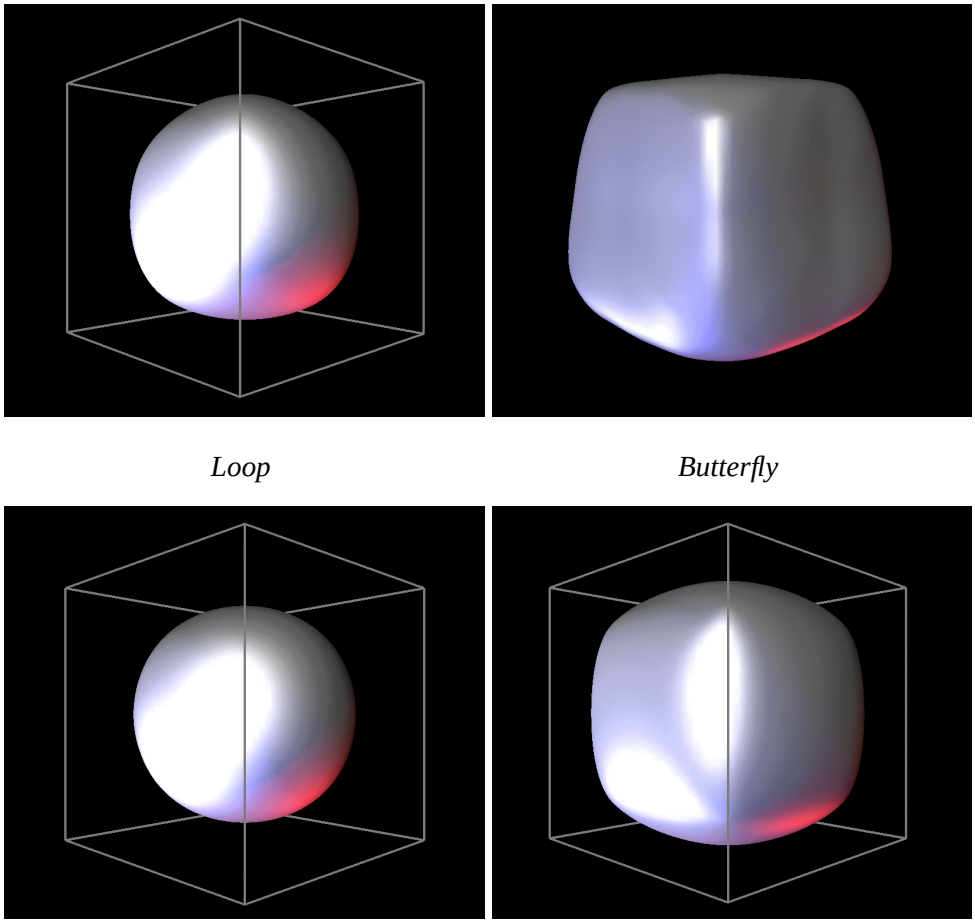
Figure 4.19 shows the surfaces obtained by subdividing a cube. Not surprisingly, Loop and Catmull-Clark subdivision produce more pleasing surfaces, as these schemes reduce to C^2 splines on a regular mesh. As all faces of the cube are quads, Catmull-Clark yields the nicest surface; the surface generated by the Loop scheme is more asymmetric, because the cube had to be triangulated before the scheme could be applied. At the same time, Doo-Sabin and Modified Butterfly reproduce the shape of the cube more closely. The surface quality is worst for the Modified Butterfly scheme, which interpolates the original mesh. We observe that there is a tradeoff between interpolation and surface quality: the closer the surface is to interpolating, the lower the surface quality.

Figure 4.20 shows the results of subdividing a tetrahedron. Similar observations hold in this case. In addition, we observe extreme shrinking for the Loop and Catmull-Clark subdivision schemes. This

is a characteristic feature of approximating schemes: for small meshes, the resulting surface is likely to occupy much smaller volume than the original control mesh.

Finally, Figure 4.21 demonstrates that for sufficiently "smooth" meshes, with uniform triangle size and sufficiently small angles between adjacent faces, different schemes may produce virtually indistinguishable results. This fact might be misleading however, especially when interpolating schemes are used; interpolating schemes are very sensitive to the presence of sharp features and may produce low quality surfaces for many input meshes unless an initial mesh smoothing step is performed.

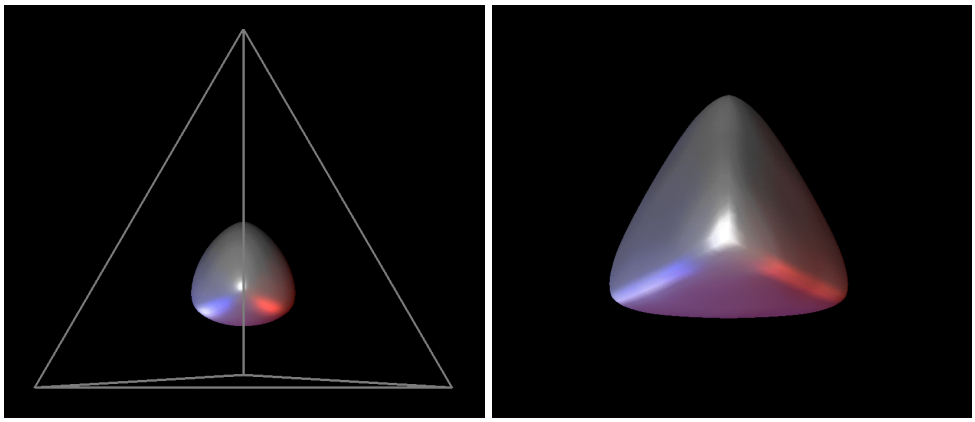
Overall, Loop and Catmull-Clark appear to be the best choices for most applications, which do not



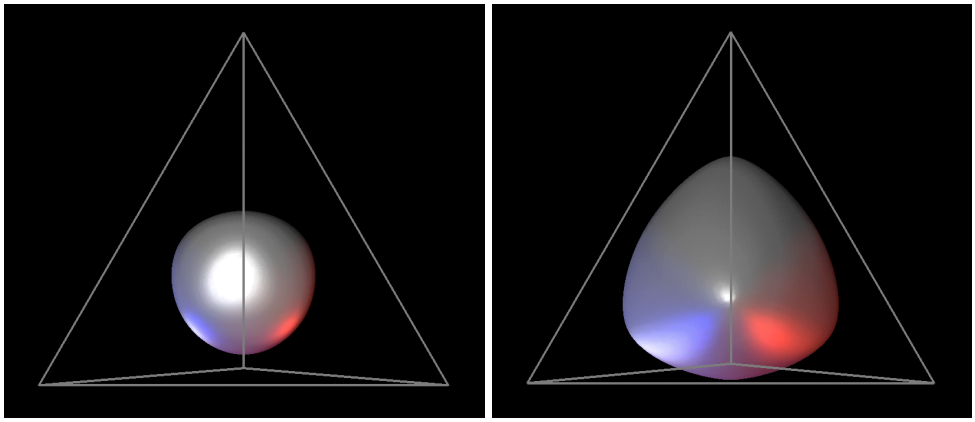
Loop *Butterfly*

Catmull-Clark *Doo-Sabin*

Figure 4.19: *Results of applying various subdivision schemes to the cube. For triangular schemes (Loop and Butterfly) the cube was triangulated first.*



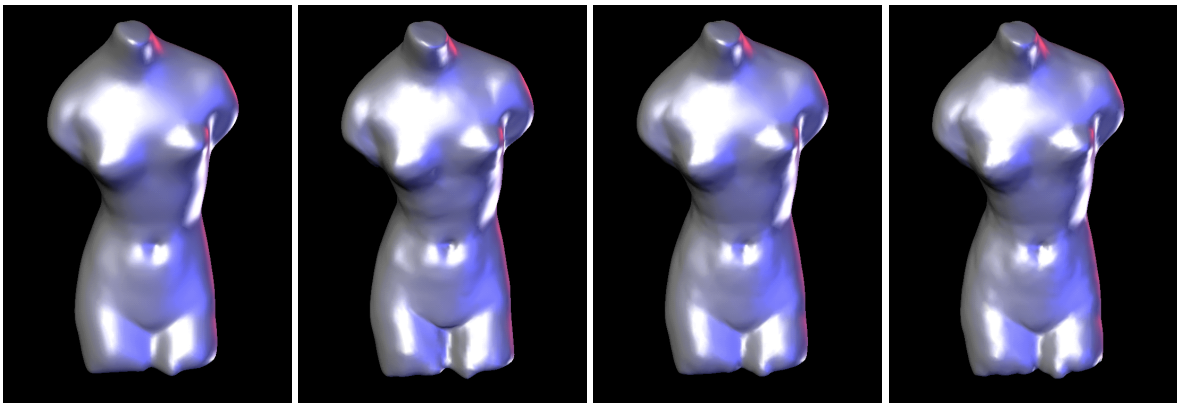
Loop *Butterfly*



Catmull-Clark *Doo-Sabin*

Figure 4.20: *Results of applying various subdivision schemes to a tetrahedron.*

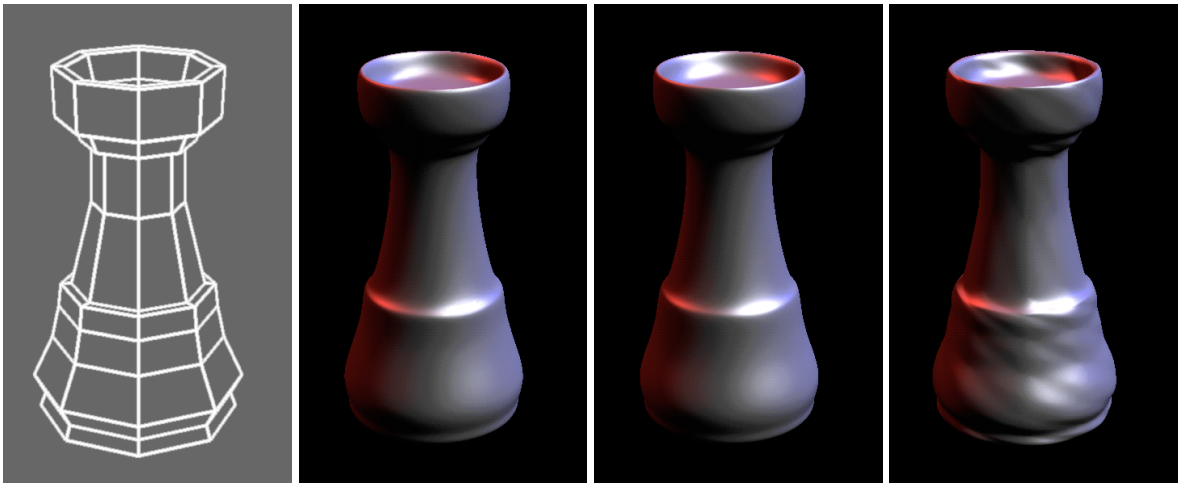
require exact interpolation of the initial mesh. The Catmull-Clark scheme is most appropriate for meshes with a significant fraction of quadrilateral faces. It might not perform well on certain types of meshes, most notably triangular meshes obtained by triangulation of a quadrilateral mesh (see Figure 4.22). The Loop scheme performs reasonably well on any triangular mesh, thus, when triangulation is not objectionable, this scheme might be preferable. There are two main reasons why a quadrilateral scheme may be preferable: natural texture mapping for quads, and a natural number of symmetries (2). Indeed, many objects and characters have two easily identifiable special directions (“along the axis of the object” and “perpendicular to the axis”). The mesh representing the object can be aligned with these directions. Objects with three natural directions, that can be used to align a triangular mesh with the object, are much



Loop *Butterfly* *Catmull-Clark* *Doo-Sabin*

Figure 4.21: *Different subdivision schemes produce similar results for smooth meshes.*

less common.



Initial mesh *Loop* *Catmull-Clark* *Catmull-Clark,after triangulation*

Figure 4.22: *Applying Loop and Catmull-Clark subdivision schemes to a model of a chess rook. The initial mesh is shown on the left. Before the Loop scheme was applied, the mesh was triangulated. Catmull-Clark was applied to the original quadrilateral model and to the triangulated model; note the substantial difference in surface quality.*

4.8.1 Comparison of Dual Quadrilateral Schemes

Dual quadrilateral schemes are the only class of schemes with several members: Doo-Sabin, Midedge, Biquartic. In this section we give some numerical examples comparing the behavior of different dual quadrilateral subdivision schemes.

Much about a subdivision scheme is revealed by looking at the associated basis functions, i.e., the result of subdividing an initial control mesh which is planar except for a single vertex which is pulled out of the plane. Figure 4.23 shows such basis functions for Midedge, Doo-Sabin, and the Biquartic scheme in the vicinity of a k-gon for $k = 4$ and $k = 9$. Note how the smoothness increases with higher order. The distinction is already apparent in the case $k = 4$, but becomes very noticeable for $k = 9$.

Figure 4.24 provides a similar comparison showing the effect of different dual quadrilateral subdivision schemes when the control polyhedron is a simple cube (compare to 4.19). Notice the increasing shrinkage with increasing smoothness. Since averages are convex combinations, the more averages are cascaded the more shrinkage can be expected.

Figure 4.25 shows a pipe shape with boundaries showing the effect of boundaries in the case of Midedge, Doo-Sabin and the Biquartic scheme.

Finally, Figure 4.26 shows the control mesh, limit surface and an adaptive tesselation blowup for a head shape.

4.9 Tilings

The classification that we have described in the beginning of the chapter, captures most known schemes. However, new schemes keep appearing, and some of the recent schemes do not fit well into this classification. It can be easily extended to handle a greater variety of schemes, if we include other refinement rules, in addition to vertex and face splits.

The starting point for refinement rules are the *isohedral tilings* and their dual tilings. A tiling is called isohedral, or Laves, if all tiles are identical, and for any vertex the angles between successive edges meeting at the vertex are equal.

In general, there are 11 tilings of the plane, shown in Figure 4.27; their dual tilings, obtained by connecting the centers of the tiles are called Archimedean tilings, and are shown in Figure 4.28. Archimedean tilings consist of regular polygons. We will refer to Laves and Archimedean tilings as regular tilings. Generalizing the idea of refinement rules to arbitrary regular tilings, we say that a refinement rule is an algorithm to obtain a finer regular tiling of the same type from a given regular tiling. This definition

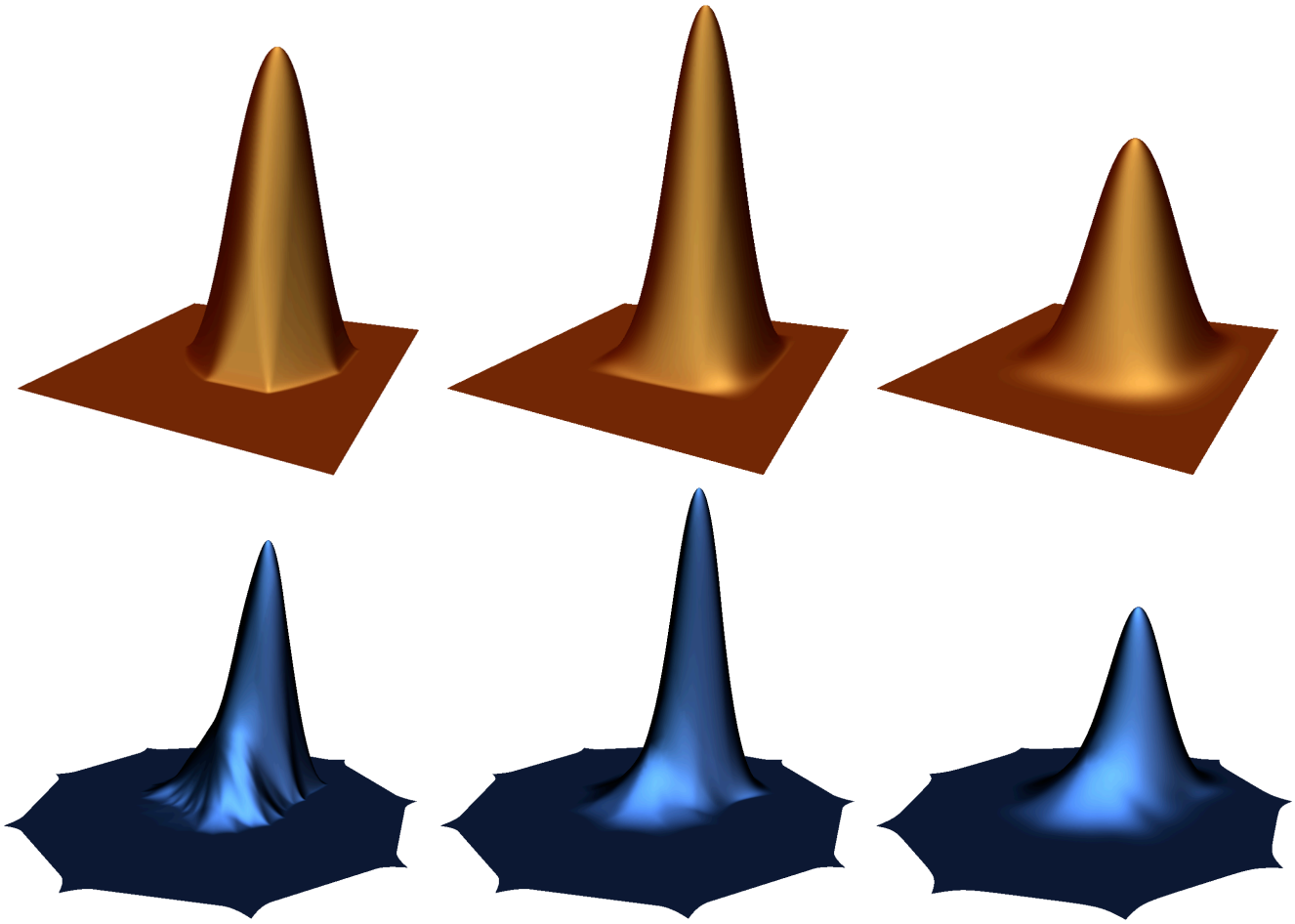


Figure 4.23: *Comparison of dual basis functions for a 4-gon (the regular case) on top and a 9-gon on the bottom. On the left the Midedge scheme (Warren/Habib variant), followed by the Doo-Sabin scheme and finally by the Biquartic generalization. The increasing smoothness is particularly noticeable in the 9-gon case.*

is quite general, and it is not known what all possible refinement rules are. The finer tiling is a scaled version of the initial tiling; the scaling factor can be arbitrary. For vertex and face splits, it is 2.

In practice, we are primarily interested in refinement rules that generalize well to arbitrary meshes. Face and vertex splits are examples of such rules. Three more exotic refinement rules have been considered: honeycomb refinement, $\sqrt{3}$ refinement and bisection.

Honeycomb refinement [7] shown in Figure 4.29, can be regarded as dual to the face split applied to the triangular mesh. While it is possible to design stationary schemes for honeycomb refinement, the

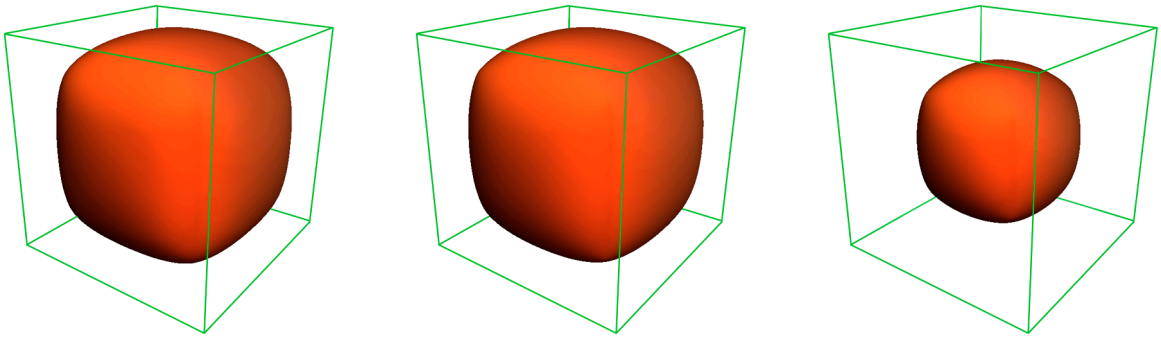


Figure 4.24: *Comparison of dual subdivision schemes (Midedge, Doo-Sabin, Biquartic) for the case of a cube. The control polyhedron is shown in outline. Notice how Doo-Sabin and even more so the Biquartic scheme exhibit considerable shrinkage in this case, while the difference between Midedge and Doo-Sabin is only slight in this example.*

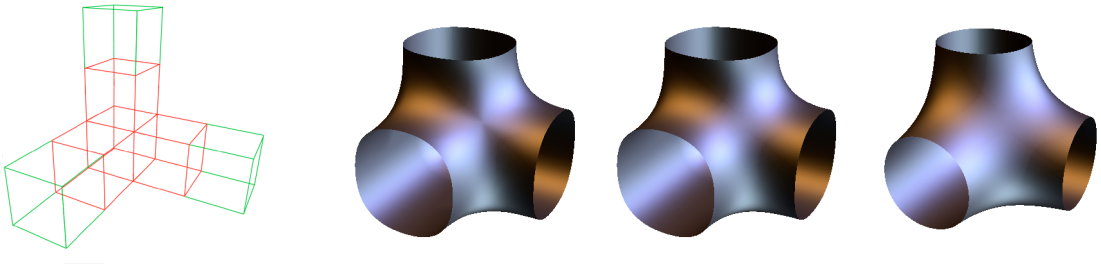


Figure 4.25: *Control mesh for a three legged pipe (left). The red parts denote the control mesh for Midedge and Doo-Sabin, while the additional green section is necessary to have a complete set of boundary conditions for the bi-quartic scheme. The resulting surfaces in order: Midedge, Doo-Sabin, and Biquartic. Note the pinch point visible for Midedge and the increasing smoothness and roundness for Doo-Sabin and Biquartic.*

scheme described in [7] is not stationary.

The $\sqrt{3}$ refinement [11], when applied to the regular triangulation of the plane (3^6 tiling), produces a tiling scaled by the factor $\sqrt{3}$ (Figure 4.30). The subdivision scheme described in [11] is stationary and produces C^2 subdivision surfaces on regular meshes.

Bisection, a well-known refinement technique often used for finite-element mesh refinement, can be used to refine $4--8$ meshes [32, 31]. The refinement process for the regular 4.8^2 tiling is illustrated in Figure 4.31. Note that a single refinement step results in a new tiling scaled by $\sqrt{2}$. As shown in [30], Catmull-Clark and Doo-Sabin subdivision schemes, as well as some higher order schemes based on face or vertex splits, can be decomposed into sequences of bisection refinement steps. Both $\sqrt{3}$ and $4--8$

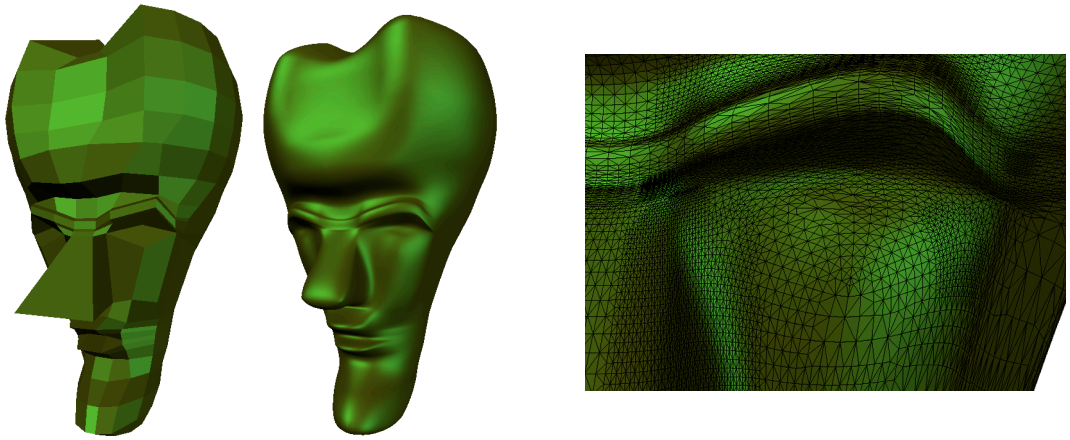


Figure 4.26: *An example of adaptive subdivision. On the left the control mesh, in the middle the smooth shaded limit surface and on the right a closeup of the adaptively triangulated limit surface.*

subdivision have the advantage of approaching the limit surface more gradually. At each subdivision step, the number of triangles triples and doubles respectively, rather then quadruple, as is the case for face split refinement. This allows finer control of the approximation. In addition, adaptive subdivision can be easier to implement, if edge-based data structures are used to represent meshes (see also Chapter 5).

4.10 Limitations of Stationary Subdivision

Stationary subdivision, while overcoming certain problems inherent in spline representations, still has a number of limitations. Most problems are much more apparent for interpolating schemes than for approximating schemes. In this section we briefly discuss a number of these problems.

Problems with curvature continuity. While it is possible to obtain subdivision schemes which are C^2-continuous, there are indications that such schemes either have very large support [24, 21], or necessarily have zero curvature at extraordinary vertices. A compromise solution was recently proposed by Umlauf [22]. Nevertheless, this limitation is quite fundamental: degeneracy or discontinuity of curvature typically leads to visible defects of the surface.

Decrease of smoothness with valence. For some schemes, as the valence increases, the magnitude of the third largest eigenvalue approaches the magnitude of the subdominant eigenvalues. As an example

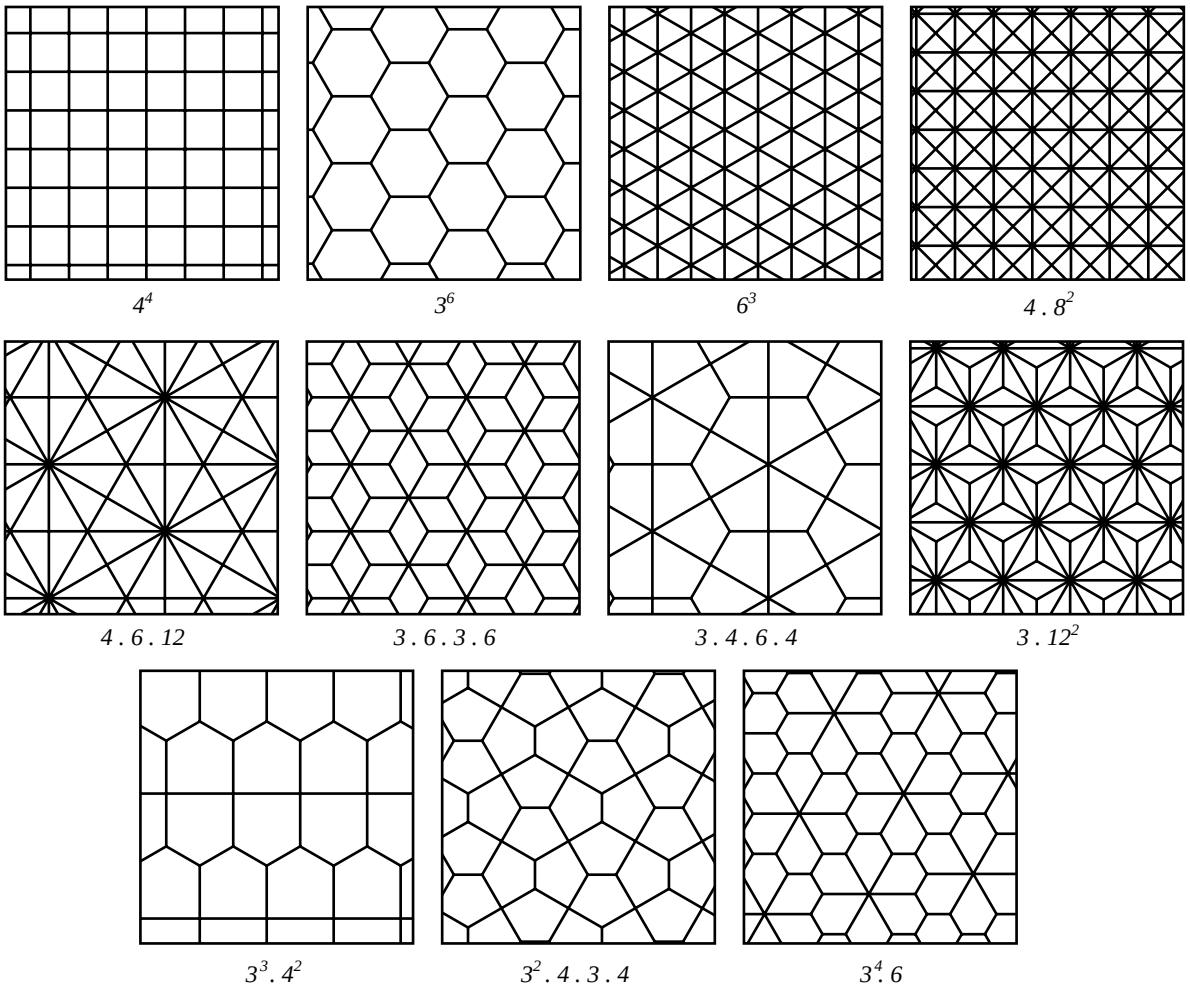


Figure 4.27: *11 Laves (isohedral) tilings.*

we consider surfaces generated by the Loop scheme near vertices of high valence. In Figure 4.32 (right side), one can see a typical problem that occurs because of "eigenvalue clustering:" a crease might appear, abruptly terminating at the vertex. In some cases this behavior may be desirable, but our goal is to make it controllable rather than let the artifacts appear by chance.

Ripples. Another problem, presence of ripples in the surface close to an extraordinary point, is also shown in Figure 4.32. It is not clear whether this artifact can be eliminated. It is closely related to the curvature problem.

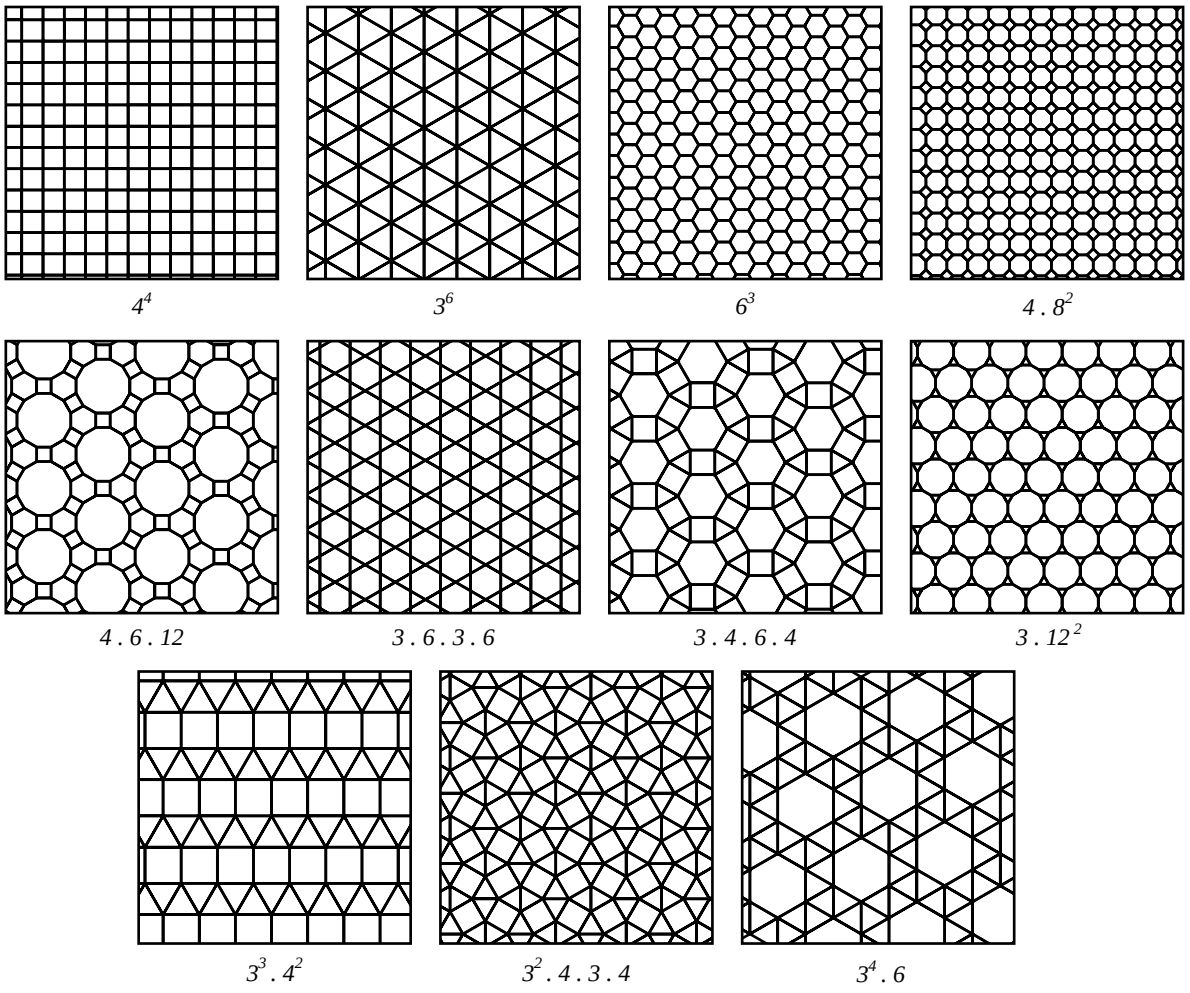


Figure 4.28: *11 Archimedean tilings, dual to Laves tilings.*

Uneven structure of the mesh. On regular meshes, subdivision matrices of C^1-continuous schemes always have subdominant eigenvalue $1/2$. When the eigenvalues of subdivision matrices near extraordinary vertices significantly differ from $1/2$, the structure of the mesh becomes uneven: the ratio of the size of triangles on finer and coarser levels adjacent to a given vertex is roughly proportional to the magnitude of the subdominant eigenvalue. This effect can be seen clearly in Figure 4.34.

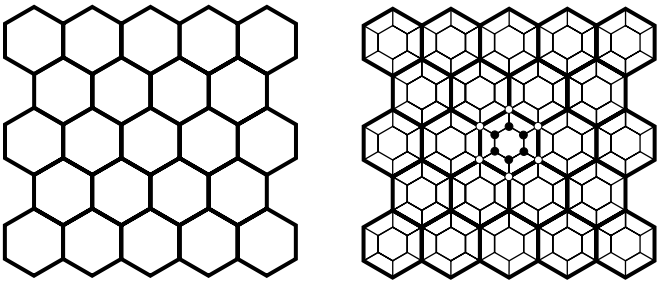


Figure 4.29: *Honeycomb refinement. Old vertices are preserved, and 6 new vertices are inserted for each face.*

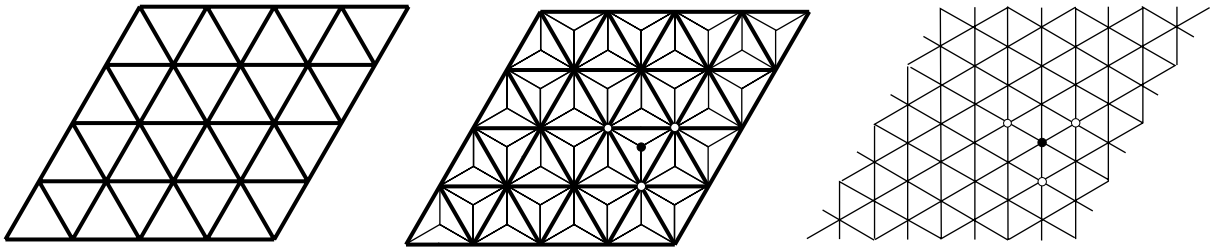


Figure 4.30: $\sqrt{3}$ *refinement. The barycenter is inserted into each triangle; this results in a* 3.12^2 *tiling. Then the edges are are flipped, to produce a new* 3^6 *tiling, which is scaled by* $\sqrt{3}$ *and rotated by 30 degrees with respect to the original.*

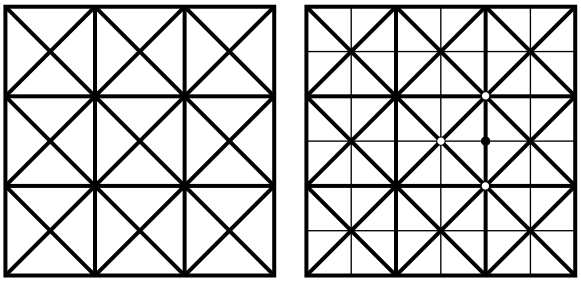


Figure 4.31: *Bisection on a* $4--8$ *tiling: the hypotenuse of each triangle is split. The resulting tiling is a new* $4--8$ *mesh, shrunk by* $\sqrt{2}$ *and rotated by 45 degrees.*

Optimization of subdivision rules. It is possible to eliminate eigenvalue clustering, as well as the difference in eigenvalues of the regular and extraordinary case by prescribing the eigenvalues of the subdivision matrix and deriving suitable subdivision coefficients. This approach was used to derive coefficients of the Butterfly scheme.

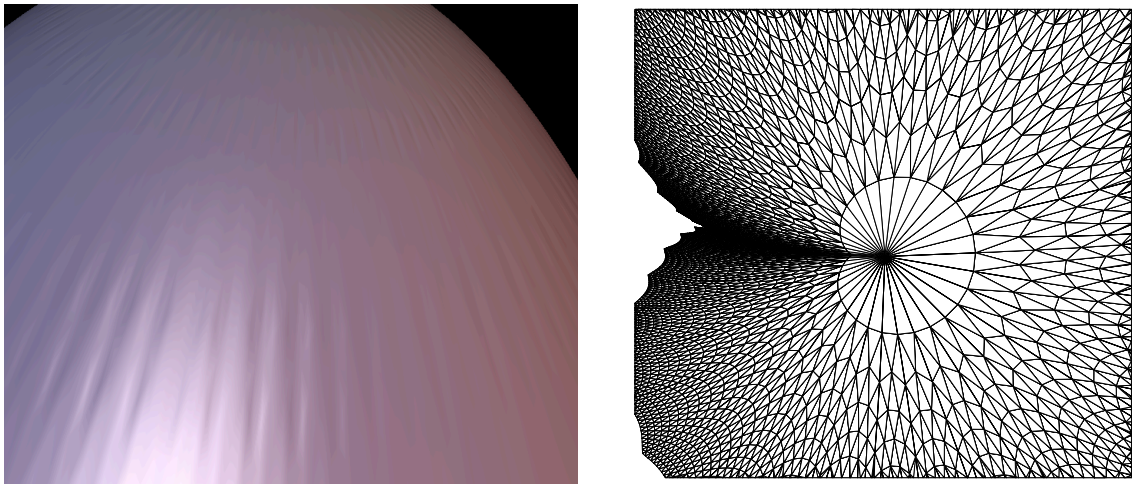


Figure 4.32: *Left: ripples on a surface generated by the Loop scheme near a vertex of large valence; Right: mesh structure for the Loop scheme near an extraordinary vertex with a significant "high-frequency" component; a crease starting at the extraordinary vertex appears.*

As expected, the meshes generated by the modified scheme have better structure near extraordinary points (Figure 4.33). However, the ripples become larger, so one kind of artifact is traded for another. It is, however, possible to seek an optimal solution or one close to optimal; alternatively, one may resort to a family of schemes that would provide for a controlled tradeoff between the two artifacts.

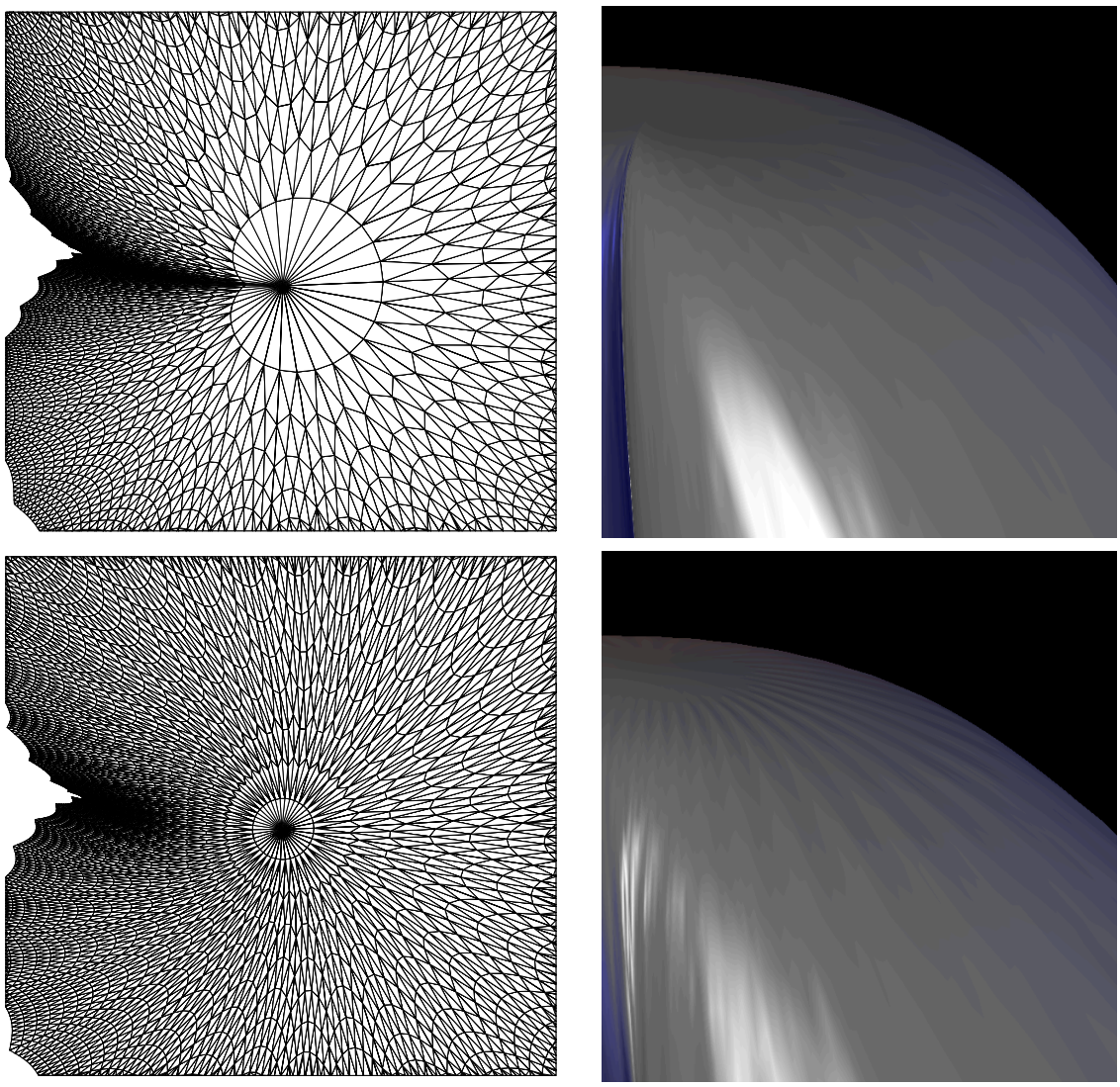


Figure 4.33: *Left: mesh structure for the Loop scheme and the modified Loop scheme near an extraordinary vertex; a crease does not appear for the modified Loop. Right: shaded images of the surfaces for Loop and modified Loop; ripples are more apparent for modified Loop.*

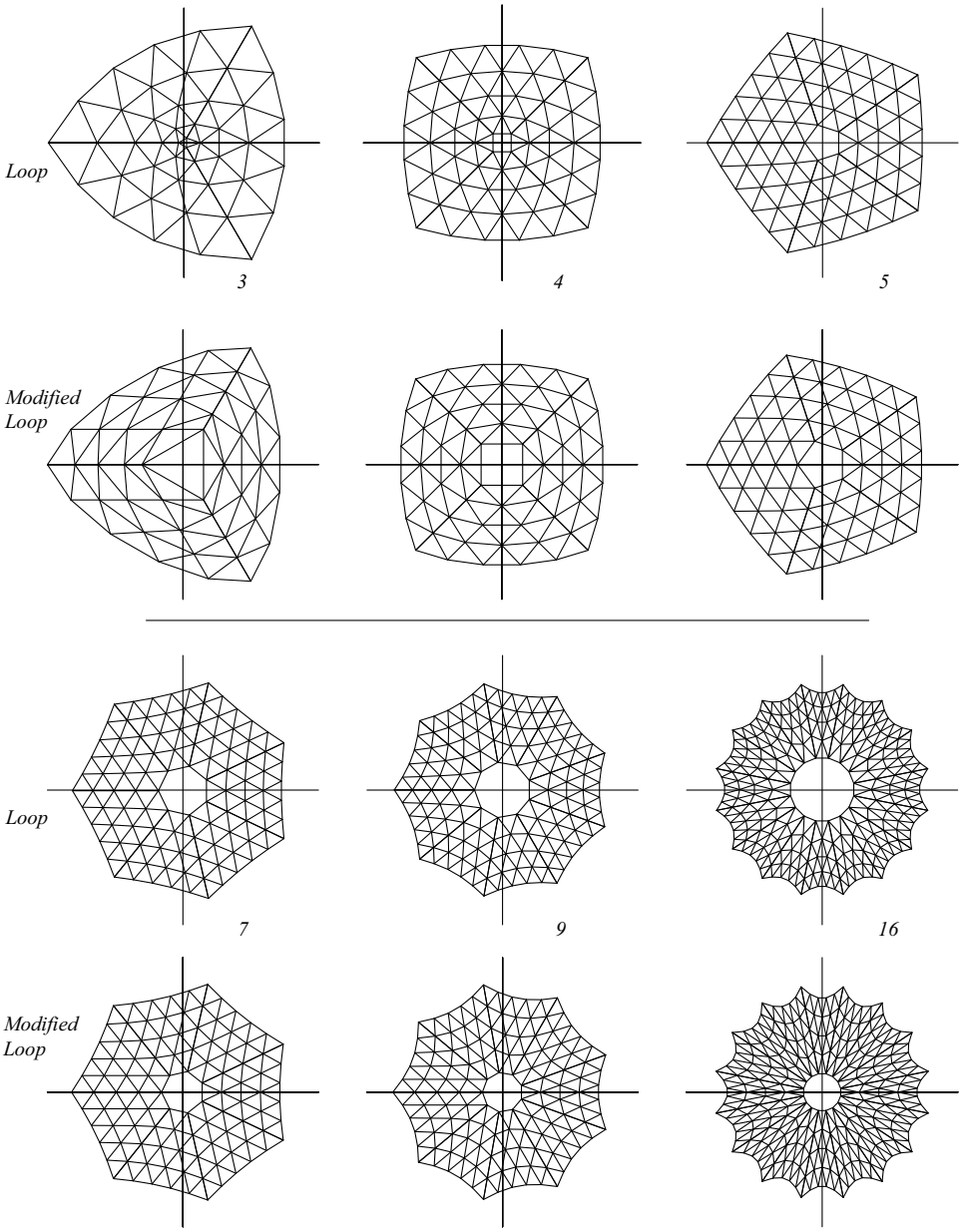


Figure 4.34: *Comparison of control nets for the Loop and modified Loop scheme. Note that for the Loop scheme the size of the hole in the ring (1-neighborhood removed) is very small relative to the surrounding triangles for valence 3 and becomes larger as k grows. For the modified Loop scheme this size remains constant.*

Chapter 5

Implementing Subdivision and Multiresolution Surfaces

Denis Zorin, New York University
Peter Schröder, Caltech

5.1 Data Structures for Subdivision

In this section we briefly describe some considerations that we found useful when choosing appropriate data structures for implementing subdivision surfaces. We will consider both primal and dual subdivision schemes, as well as triangle and quadrilateral based schemes.

5.1.1 Representing Arbitrary Meshes

In all cases, we need to start with data structures representing the top-level mesh. For subdivision schemes we typically assume that the top level mesh satisfies several requirements that allow us to apply the subdivision rules everywhere. These requirements are

- no more than two polygons share an edge;
- all polygons sharing a vertex form an open or closed neighborhood of the vertex; in other words, can be arranged in such an order that two sequential polygons always share an edge.

A variety of representations were proposed in the past for general meshes of this type, sometimes with some of the assumptions relaxed, sometimes with more assumptions added, such as orientability of the

surface represented by the mesh. These representations include winged edge, quad edge, half edge end other data structures. The most common one is the winged edge. However, this data structure is far from being the most space efficient and convenient for subdivision. First, most data that we need to store in a mesh, is naturally associated with vertices and polygons, not edges. Edge-based data structures are more appropriate in the context of edge-collapse-based simplification. For subdivision, it is more natural to consider data structures with explicit representations for faces and vertices, not for edges. One possible and relatively simple data structure for polygons is

```
struct Polygon{
  vector<Vertex*>   vertices;
  vector<Polygon*>  neighbors;
  vector<short>     neighborEdges;
  ...
}
```

For each polygon, we store an array of pointers to vertices and an array of adjacent polygons (neighbors) across corresponding edge numbers. We also need to know for each edge what the corresponding edge number of that edge is, when seen from the neighbor across that edge. This information is stored in the array `neighborEdges` (see Figure 5.1). In addition, if we allow non-orientable surfaces, we need to

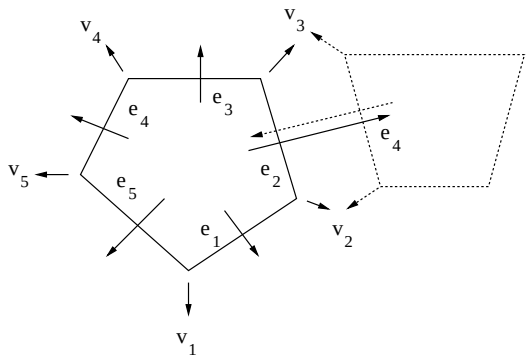


Figure 5.1: *A polygon is described by an array of vertex pointers and an array of neighbor pointers (one such neighbor is indicated in dotted outline). Note that the neighbor has its own edge number assignment which may differ across the shared edge.*

keep track of the orientation of the neighbors, which can be achieved by using signed edge numbers in the array `neighorEdges`. To complete the mesh representation, we add a data structure for vertices to the polygon data structure.

Let us compare this data structure to the winged edge. Let P be the number of polygons in the mesh, V the number of vertices and E the number of edges. The storage required for the polygon-based data structure is approximately $2.5 \cdot P \cdot V_P$ 32-bit words, where V_P is the average number of vertices per polygon. Here we assuming that all polygons have fewer than 2^{16} edges, so only 2 bytes are required to store the edge number. Note that we disregard the geometric and other information stored in vertices and polygons, counting only the memory used to maintain the data structure.

To estimate the value of $2.5 \cdot P \cdot V_P$ in terms of V, we use the Euler formula. Recall that any mesh satisfies $V - E + P = g$, where g is the genus, the number of "holes" in the surface. Assuming genus small compared to the number of vertices, we get an approximate equation $V - E + P = 0$; we also assume that the boundary vertices are a negligible fraction of the total number of vertices. Each polygon on the average has V_P vertices and the same number of edges. Each edge is shared by two polygons which results in $E = V_P \cdot P/2$. Let P_V be the number of polygons per vertex. Then $P = P_V \cdot V/V_P$, and $E = V P_V/2$. This leads to

$$\frac{1}{P_V} + \frac{1}{V_P} = \frac{1}{2}. \tag{5.1}$$

In addition, we know that V_P, the average number of vertices per polygon, is at least 3. It follows from (5.1) that $P_V \leq 6$. Therefore, the total memory spent in the polygon data structure is $2.5 P_V \cdot V \leq 15V$.

The winged edge data structure requires 8 pointers per edge. Four pointers to adjacent edges, two pointers to adjacent faces, and two pointers to vertices. Given that the total number of edges E is greater than $3V$, the total memory consumption is greater than $24V$, significantly worse than the polygon data structure.

One of the commonly mentioned advantages of the winged edge data structure is its constant size. It is unclear if this has any consequence in the context of C++: it is relatively easy to create structures with variable size. However, having a variety of dynamically allocated data of different small sizes may have a negative impact on performance. We observe that after the first subdivision step all polygons will be either triangles or quadrilaterals for all schemes that we have considered, so most of the data items will have fixed size and the memory allocation can be easily optimized.

5.1.2 Hierarchical Meshes: Arrays vs. Trees

Once a mesh is subdivided, we need to represent all the polygons generated by subdivision. The choice of representation depends on many factors. One of the important decisions to make is whether adaptive subdivision is necessary for a particular application or not. To understand this tradeoff we need to

estimate the storage associated with arrays vs. trees. To make this argument simple we will consider here only the case of triangle based subdivision such as Loop or Butterfly. The counting arguments for quadrilaterals schemes (both primal and dual) are essentially similar.

Assuming that only uniform subdivision is needed, all vertices and triangles associated with each subdivided top-level triangle can be represented as a two-dimensional array. Thus, the complete data structure would consist of a representation of a top level mesh, with each top level triangle containing a 2D array of vertex pointers. The pointers on the border between two top-level neighbors point pairwise to the same vertices. The advantage of this data structure is that it has practically no pointer overhead. The disadvantage is that a lot of space will be wasted if adaptive subdivision is performed.

If we do want adaptive subdivision and maintain efficient storage, the alternative is to use a tree structure. Each non-leaf triangle becomes a node in a quadtree, containing a pointer to a block of 4 children and pointers to three corner vertices

```
class TriangleQuadTree{
  Vertex*            v1, v2, v3;
  TriangleQuadTree*  firstChild;
  ...
}
```

Comparison. To compare the two approaches to organizing the hierarchies (arrays and trees), we need to compare the representation overhead in these two cases. In the first case (arrays) all adjacency relations are implicit, and there is no overhead. In the second case, there is overhead in the form of pointers to children and vertices. For a given number of subdivision steps n the total overhead can be easily estimated. For the purposes of the estimate we can assume that the genus of our initial control mesh is 0, so the number of triangles P, the number of edges E and the number of vertices V in the initial mesh are related by $P - E + V = 0$. The total number of triangles in a complete tree of depth n for P initial triangles is given by $P(4^{n+1} - 1)/3$. For a triangle mesh $V_P = 3$ and $P_V = 6$ (see Eq. (5.1)); thus, the total number of triangles is $P = 2V$, and the total number of edges is $E = 3V$.

For each leaf and non-leaf node we need 4 words (1 pointer to the block of children and three pointers to vertices). The total cost of the structure is $4P(4^{n+1} - 1)/3 = 8V(4^{n+1} - 1)/3$ words, which is approximately $11 \cdot V \cdot 4^n$.

To estimate when a tree is spatially more efficient than an array, we determine how many nodes have to be removed from the tree for the gain from the adaptivity to exceed the loss from the overhead. For

this, we need a reasonable estimate of the size of the useful data stored in the structures, otherwise the array will always win.

The number of vertices inserted on subdivision step i is approximately $3 \cdot 4^{i-1}V$. Assuming that for each vertex we store all control points on all subdivision levels, and each control point takes 3 words, we get the following estimate for the control point storage

$$3V\left((n+1)+3n+3\cdot 4^2(n-1)+\ldots 4^n\right) = V\left(4^{n+1}-1\right).$$

The total number of vertices is $V \cdot 4^n$; assuming that at each vertex we store the normal vector, the limit position vector (3 words), color (3 words) and some extra information, such as subdivision tags (1 word), we get $7 \cdot V \cdot 4^n$ more words. The total useful storage is approximately $11 \cdot V \cdot 4^n$, the same as the cost of the structure.

Thus for our example the tree introduces a 100% overhead, which implies that it has an advantage over the array if at least half of the nodes are absent. Whether this will happen, depends on the criterion for adaptation. If the criterion attempts to measure how well the surface approximates the geometry, and if only 3 or 4 subdivision levels are used, we have observed that fewer than 50% of the nodes were removed. However, if different criteria are used (e.g. distance to the camera) the situation is likely to be radically different. If more subdivision levels are used it is likely that almost all nodes on the finest level are absent.

5.1.3 Implementations

In many settings tree-based implementations, even with their additional overhead, are highly desirable. The case of quadtrees for primal triangle schemes is covered in [41] (this article is reprinted at the end of this chapter). The machinery for primal quadrilateral schemes (e.g., Catmull-Clark) is very similar. Here we look in some more detail at quadtrees for dual quadrilateral schemes. Since these are based on vertex splits the natural organization are quadtrees based on vertices *not* faces. As we will see the two trees are not that different and an actual implementation easily supports both primal and dual quadrilateral schemes. We begin with the dual quadrilateral case.

Representation

At the coarsest level the input control mesh is represented as a general mesh as described in Section 5.1.1. For simplicity we assume that the control mesh satisfies the property that all vertices have valence four. This can always be achieved through one step of dual subdivision. The valence four assumption allows

us to use quadtrees for the organization of vertices without an extra layer for the coarsest level. In fact we only have to organize a forest of quadtrees. Each quadtree root maintains four pointers to neighboring quadtrees roots

```
class QTreeR{
  QTreeR*   n[4];      // four neighbors
  QTree*    root;      // the actual tree
}
```

A quadtree is given as

```
class QTree{
  QTree*    p;         // parent
  QTree*    c[4];      // children
  Vector3D  dual;      // dual control point
  Vector3D* primal[4]; // shared corners
}
```

The organization of these quadtrees is depicted in Figure 5.2. Both primal and dual subdivision can

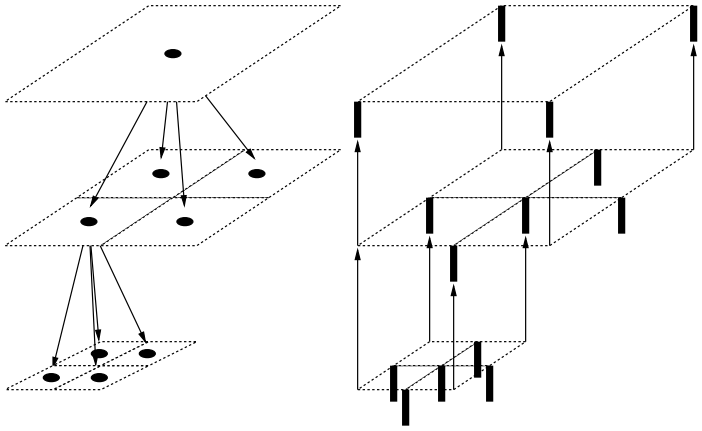


Figure 5.2: *Quadtrees carry dual control points (left). We may think of every quadtree element as describing a small rectangular piece of the limit surface centered at the associated control point (compare to Figure 5.3). The corners of those quads correspond to the location of primal control points (right) in a primal quadrilateral subdivision scheme. As usual these are shared among levels.*

now be effected by iterating over all faces and repeatedly averaging to achieve the desired order of subdivision [34, 30, 39]. Alternatively one may apply subdivision rules in the more traditional setup by

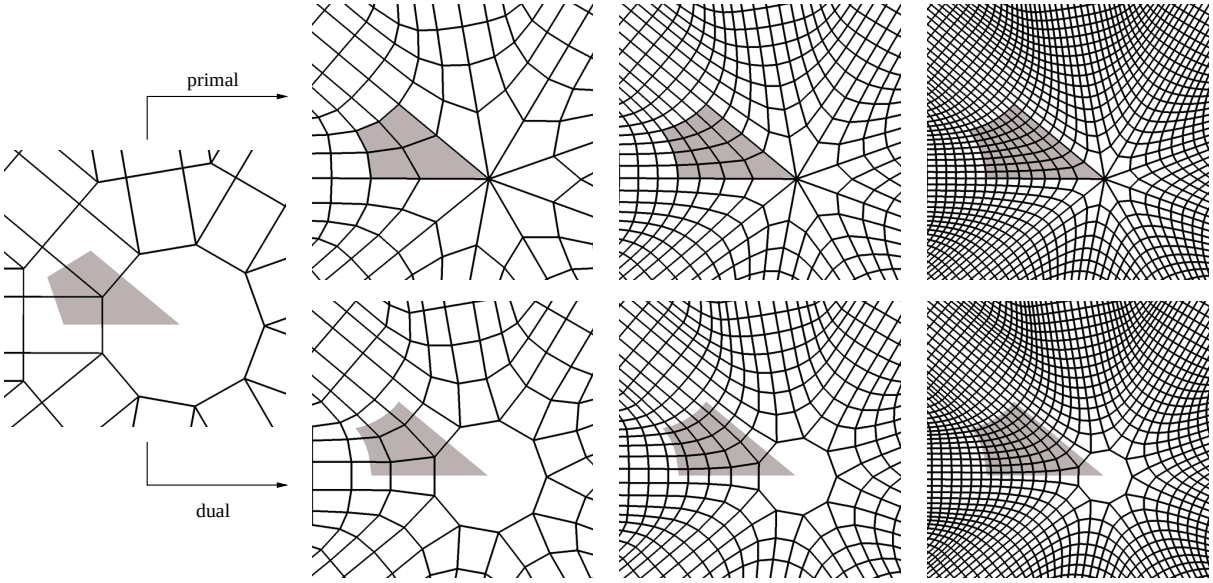


Figure 5.3: *Given some arbitrary input mesh we may associate limit patches of dual schemes with vertices in the input mesh while primal schemes result in patches associated with faces. Here we see examples of the Catmull-Clark (top) and Doo-Sabin (bottom) acting on the same input mesh (left).*

collecting the 1-ring of neighbors of a given control point (primal or dual). Collecting a 1-ring requires only the standard neighbor finding routines for quadtrees [27]. If the neighbor finding routine crosses from one quadtree to another the quadtree root links are used to effect this transition. Nil pointers indicate boundaries. With the 1-ring in hand one may apply stencils directly as indicated in Chapter 4. Using 1-rings and explicit subdivision masks, as opposed to repeated averaging, significantly simplifies boundary treatments and adaptivity.

Boundaries are typically dealt with in primal schemes using special boundary rules (see Chapter 4). For example, in the case of Catmull-Clark one can ensure that the outermost row of control vertices describes an endpoint interpolating cubic spline (see, e.g., [2]). For dual schemes, for example Doo-Sabin, a common solution is to replicate boundary control points (for other possibilities see the references in Chapter 4).

Constructing higher order quadrilateral subdivision schemes through repeated averaging will result in increasing shrinkage. This is true both for closed control meshes (see Figure 4.24) and for boundaries (see Figure 4.25). To address the boundary issue the repeated averaging steps may be modified there or one could simply drop the order of the method near the boundary. For example, in the case of the Biquartic scheme one may use the Doo-Sabin rules whenever a complete 1-ring is not available. This

leads to lower order near the boundary but avoids excessive shrinkage for high order methods. Which method is preferable depends heavily on the intended application.

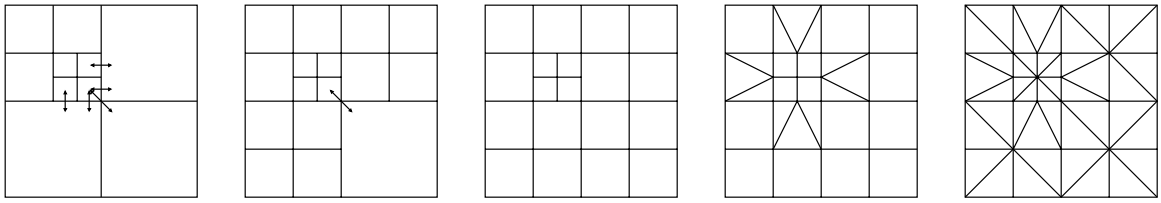


Figure 5.4: *On the left an unrestricted adaptive primal quadtree. Arrows indicate edge and vertex neighbors off by more than 1 level. Enforcing a standard edge restriction criterion enforces some additional subdivision. A vertex restriction criterion also disallows vertex neighbors off by more than 1 level. Finally on the right some adaptive tesselations which are crack-free.*

Adaptive Subdivision, as indicated earlier, can be valuable in some applications and may be mandatory in interactive settings to maintain high frame rates while escaping the exponential growth in the number of polygons with successive subdivisions. We first consider adaptive tesselations for primal quad schemes and then show how the same machinery applies to dual quad schemes.

To make such adaptive tesselations manageable it is common to enforce a restriction criterion on the quadtrees, i.e, no quadtree node is allowed to be off by more than one subdivision level from its neighbors. Typically this is applied only to *edge* neighbors, but we need a slightly stronger criterion covering *all* neighbors, i.e., including those sharing only a common vertex. This criterion is a consequence of the fact that to compute a control point at a finer level we need a complete subdivision stencil at a courser level. for primal schemes, it means that if a face is subdivided, all faces sharing a vertex with it must be present. This idea is illustrated in Figure 5.4

Once a vertex restricted adaptive quadtree exists one must take care to output quadrilaterals or triangles in such a way that no cracks appear. Since all rendering is done with triangles we consider crack-free output of a triangulation only. This requires the insertion of diagonals in all quadrilaterals. One can make this choice randomly, but surfaces appear "nicer" if this is done in a regular fashion. Figure 5.5 illustrates this on the top for a group of four children of a common parent. Here the diagonals are chosen to meet at the center. The resulting triangulation is exactly the basic element of a 4-8 tiling [32]. To deal with cracks we distinguish 16 cases. Given a leaf quadrilateral its edge neighbors may be subdivided once less, as much, or once more. Only the latter case gives rise to potential cracks from the point of view of the leaf quad. The 16 cases are easily distinguished by considering a bit flag for each edge indicating whether the edge neighbor is subdivided once more or not. Figure 5.5 shows the resulting templates

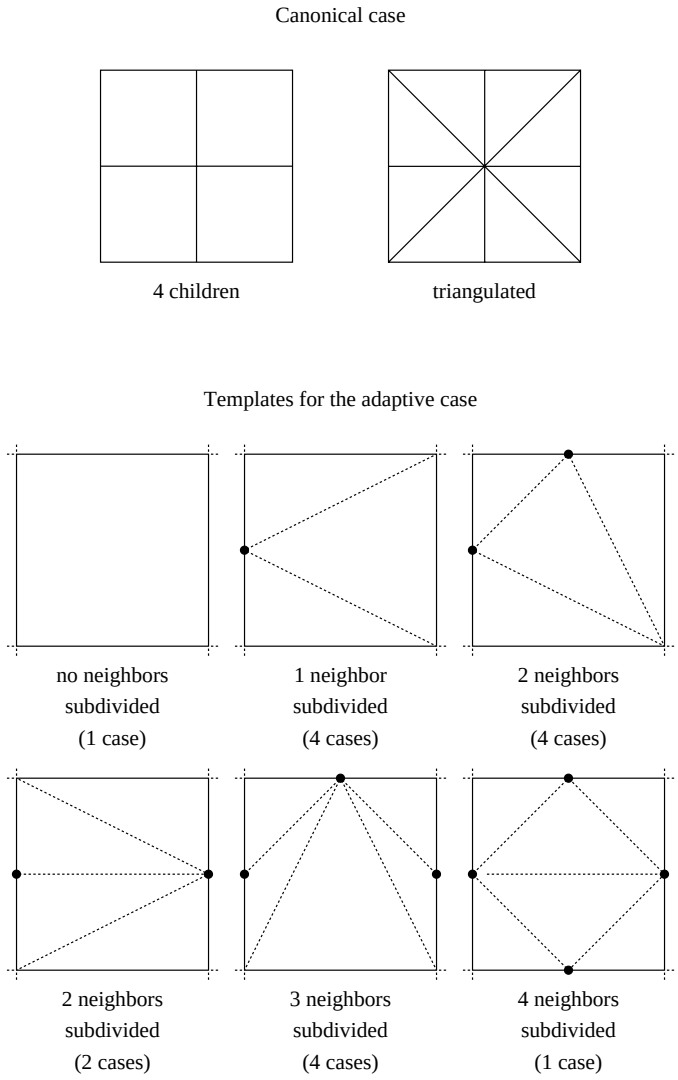


Figure 5.5: *The top row shows the standard triangulation for a group of 4 child faces of a single face (face split subdivision). The 16 cases of adaptive triangulation of a leaf quadrilateral are shown below. Any one of the four edge neighbors may or may not be subdivided one level finer. Using the indicated templates one can triangulate an adaptive primal quad tree with a simple lookup table.*

(modulo symmetries). These are easily implemented as a lookup table.

For dual quadrilateral subdivision schemes crack-free adaptive tesselations are harder to generate. Recall that in a dual quad scheme a quadtree node represents a control point, not a face. It potentially connects to all 8 neighbors (see Figure 5.6, left). Consequently there are 256 possible tesselations de-

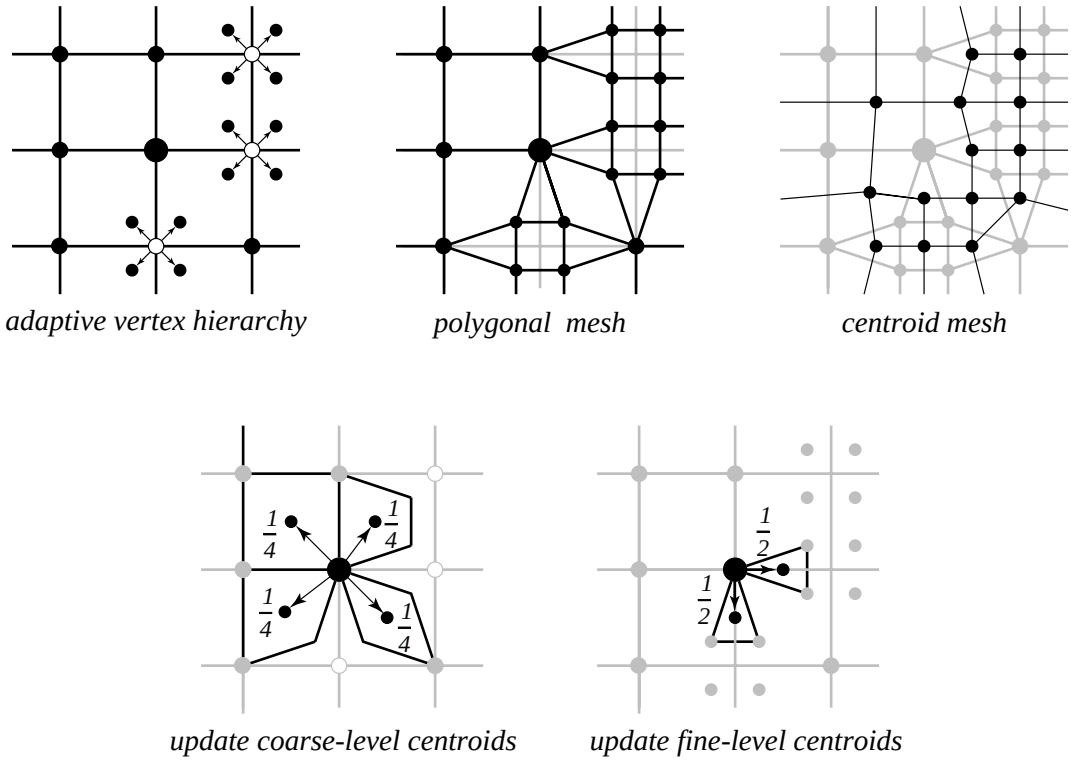


Figure 5.6: *To produce a polygonal mesh for a restricted vertex-split hierarchy (top row, left), rather than trying to generate the mesh connecting the vertices (top row, middle) of the mesh, we generate the mesh connecting the centroids of the faces (top row, right). Centroids are associated with corners at subdivision levels. To compute centroids correctly, we traverse the vertices in the vertex hierarchy, and add contributions of the vertex to the centroids associated with the vertex (bottom row, left) and centroids associated with the corners attached to the children of a neighbor (bottom row, right). The choice of coefficients guarantees that centroids are found correctly.*

pending on 8 neighbor states.

To avoid this explosion of cases we instead choose to draw (or output) a tesselation of the centroids of the dual control points. These live at corners again, so the adaptive tesselation machinery from the primal setting applies. This approach has the added benefit of producing samples of the limit surface for the Doo-Sabin and Midedge scheme. For the Biquartic scheme, unfortunately, limit points are not centroids of faces. Note that this additional averaging step is only performed during drawing or output and does not change the overall scheme. Figure 5.6 (right) shows how to form the additional averages in an adaptive setting. With these drawing averages computed we apply the templates of Figure 5.5 to

render the output mesh. Figure 4.26 shows an example of such an adaptively rendered mesh.

Bibliography

[1] BALL, A. A., AND STORRY, D. J. T. Conditions for tangent plane continuity over recursively generated B-spline surfaces. *ACM Transactions on Graphics 7*, 2 (1988), 83–102.

[2] BIERMANN, H., LEVIN, A., AND ZORIN, D. Piecewise smooth subdivision surfaces with normal control. In *SIGGRAPH 2000 Conference Proceedings* (July 2000), Annual Conference Series, ACM SIGGRAPH, Addison Wesley.

[3] CATMULL, E., AND CLARK, J. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10*, 6 (1978), 350–355.

[4] DOO, D., AND SABIN, M. Analysis of the behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design 10*, 6 (1978), 356–360.

[5] DYN, N., GREGORY, J. A., AND LEVIN, D. A four-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design 4* (1987), 257–268.

[6] DYN, N., LEVIN, D., AND GREGORY, J. A. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics 9*, 2 (April 1990), 160–169.

[7] DYN, N., LEVIN, D., AND LIU, D. Interpolatory convexity-preserving subdivision for curves and surfaces. *Computer-Aided Design 24*, 4 (1992), 211–216.

[8] HABIB, A., AND WARREN, J. Edge and vertex insertion for a class of c^1 subdivision surfaces. presented at 4th SIAM COnference on Geometric Design, November 1995.

[9] HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. Piecewise smooth surface reconsruction. In *Computer Graphics Proceedings* (1994), Annual Conference Series, ACM Siggraph, pp. 295–302.

[10] KOBBELT, L. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Proceedings of Eurographics 96* (1996), Computer Graphics Forum, pp. 409–420.

[11] KOBBELT, L. $\sqrt{3}$ Subdivision. ACM Siggraph. to appear.

[12] LEVIN, A. Boundary algorithms for subdivision surfaces. In *Israel-Korea Bi-National Conference on New Themes in Computerized Geometrical Modeling* (1998), pp. 117–121.

[13] LEVIN, A. Combined subdivision schemes for the design of surfaces satisfying boundary conditions. To appear in CAGD, 1999.

[14] LEVIN, A. Interpolating nets of curves by smooth subdivision surfaces. to appear in SIGGRAPH'99 proceedings, 1999.

[15] LOOP, C. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[16] NASRI, A. H. Polyhedral subdivision methods for free-form surfaces. *ACM Transactions on Graphics 6*, 1 (January 1987), 29–73.

[17] PETERS, J., AND REIF, U. Analysis of generalized B-spline subdivision algorithms. *SIAM Jornal of Numerical Analysis* (1997).

[18] PETERS, J., AND REIF, U. The simplest subdivision scheme for smoothing polyhedra. *ACM Transactions on Graphics 16(4)* (October 1997).

[19] PRAUTZSCH, H. Analysis of C^k-subdivision surfaces at extraordianry points. Preprint. Presented at Oberwolfach, June, 1995, 1995.

[20] PRAUTZSCH, H. Smoothness of subdivision surfaces at extraordinary points. *Adv. Comput. Math. 9*, 3-4 (1998), 377–389.

[21] PRAUTZSCH, H., AND REIF, U. Necessary conditions for subdivision surfaces. 1996.

[22] PRAUTZSCH, H., AND UMLAUF, G. A G^2-subdivision algorithm. In *Geometric Modeling*, G. Farin, H. Bieri, G. Brunnet, and T. DeRose, Eds., vol. Computing Suppl. 13. Springer-Verlag, 1998, pp. 217–224.

[23] QU, R. *Recursive Subdivision Algorithms for Curve and Surface Design*. PhD thesis, Brunel University, 1990.

[24] REIF, U. A degree estimate for polynomial subdivision surface of higher regularity. Tech. rep., Universität Stuttgart, Mathematisches Institut A, 1995. preprint.

[25] REIF, U. Some new results on subdivision algorithms for meshes of arbitrary topology. In *Approximation Theory VIII*, C. K. Chui and L. Schumaker, Eds., vol. 2. World Scientific, Singapore, 1995, pp. 367–374.

[26] REIF, U. A unified approach to subdivision algorithms near extraordinary points. *Computer Aided Geometric Design 12* (1995), 153–174.

[27] SAMET, H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[28] SCHWEITZER, J. E. *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, Seattle, 1996.

[29] STAM, J. On subdivision schemes generalizing uniform b-spline surfaces of arbitrary degree. Submitted for Publication, 2000.

[30] VELHO, L. Using semi-regular 4-8 meshes for subdivision surfaces. to appear in the Journal of Graphics Tools, 2001.

[31] VELHO, L., AND GOMES, J. Quasi 4-8 subdivision surfaces. In *XII Brazilian Symposium on Computer Graphics and Image Processing* (1999), IEEE.

[32] VELHO, L., AND ZORIN, D. 4–8 subdivision. to appear in CAGD, 2000.

[33] WARREN, J. Subdivision methods for geometric design. Unpublished manuscript, November 1995.

[34] WARREN, J., AND WEIMER, H. Subdivision for geometric design. 2000.

[35] ZORIN, D. *Subdivision and Multiresolution Surface Representations*. PhD thesis, Caltech, Pasadena, 1997.

[36] ZORIN, D. A method for analysis of C^1-continuity of subdivision surfaces. *SIAM Journal of Numerical Analysis 37*, 4 (2000), 1677–1708.

[37] ZORIN, D. Smoothness of subdivision on irregular meshes. *Constructive Approximation 16*, 3 (2000), 359–397.

[38] ZORIN, D. Smoothness of subdivision surfaces on the boundary. preprint, Computer Science Department, New York University, 2000.

[39] ZORIN, D., AND SCHRÖDER, P. A unified framework for primal and dual subdivision schemes. to appear in CAGD, 2001.

[40] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interpolating subdivision for meshes with arbitrary topology. *Computer Graphics Proceedings (SIGGRAPH 96)* (1996), 189–192.

[41] ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. Interactive multiresolution mesh editing. ACM Siggraph.

Implementing Subdivision

Data Structures and Other Concerns

Stephen Junkins
stephen.junkins@intel.com
Intel Architecture Labs, Intel Corporation

Agenda

- Assumptions & background
- Control mesh data structures
- Subdivision data structures
- Real time concerns
  - Adaptive subdivision
  - Animation
- Conclusions

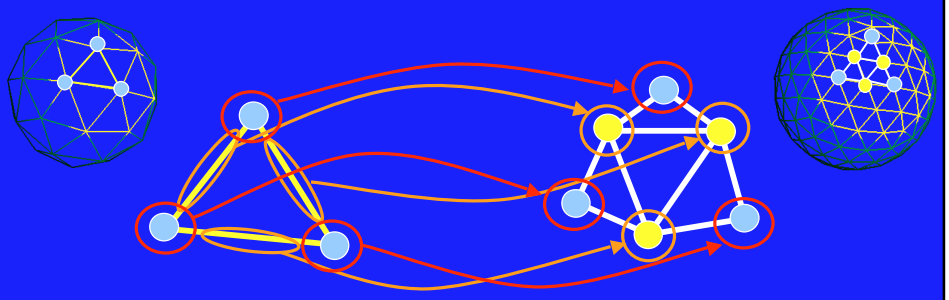
Assumptions

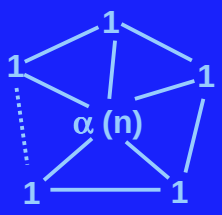
- Real time subdivision
- Triangle based schemes, e.g. Loop, Butterfly
- Assume attributes (position, normal, color, tex coords) stored on vertex
- Gather triangles, vertices from triangles.

Data Structure Requirements

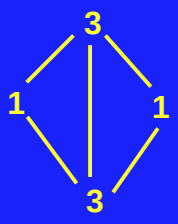
- Provide triangle neighbors of control mesh
- Provide neighbors of subdivision generated triangles
- Provide parent triangles
- Dynamic
- Arbitrary vertex valance
- Identify Non-manifolds

Loop's Subdivision Masks





Vertex Mask



Edge Mask

Stephen Junkins Intel Architecture Labs

Loop's Mask Weights

$$\text{Vertices}: v^{r+1} = \frac{\alpha(n)v^r + v_1^r + \ldots + v_n^r}{\alpha(n) + n}$$

$$\alpha(n) = \frac{n\,(1 - a(n))}{a(n)}$$

$$a(n) = \frac{5}{8} + \frac{(3 + 2\cos(2\prod/n))^2}{64}$$

$$\text{Edges}: v_i^{r+1} = \frac{3v^r + 3v_i^r + v_{i-1}^r + v_{i+1}^r}{8}, i = 1,\ldots n$$

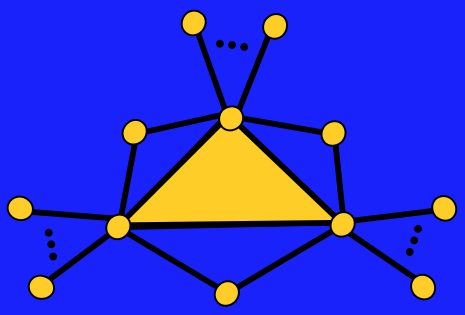
Stephen Junkins Intel Architecture Labs

Control Mesh Data Structures

1. Winged Edge
2. Triangle Pointers

Stephen Junkins Intel Architecture Labs

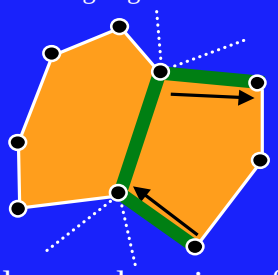
Local neighborhood for Loop's Subdiv of one triangle



Stephen Junkins Intel Architecture Labs

Winged Edge Control Mesh

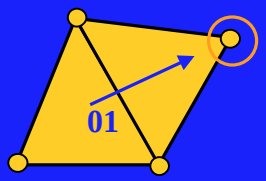
- Winged Edge [Baumgart75]…perhaps overkill
  - Each "edge element" stores:
    1. Starting & Ending vertices
    2. Adjacent faces
    3. Proceeding & following edges in each direction

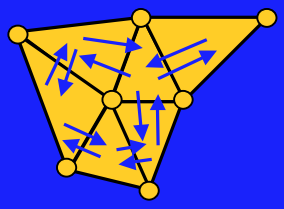


- Oriented traversal around vertices, faces
- O(1) adjacency queries
- Generalized for n-gons…Catmull-Clark Subdiv

Triangle Pointer Based Control Mesh

- Each triangle "side":
  - Pointer to neighbor triangles
  - 2 bits identify vertex, opposite shared edge





- Nice companion to triangle meshes
- Oriented traversals around vertices
- Mem cost: 3 pointers & 6 bits…13 bytes/triangle

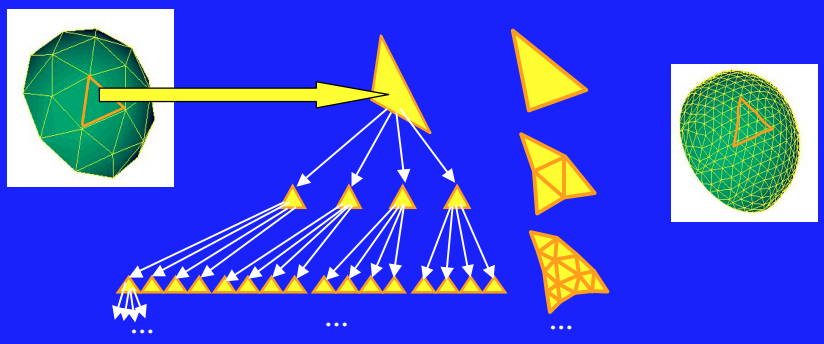
Subdivision Data Structures

1. Triangular quadtree & NCA
2. Lee-Samet Addressing
3. Pulli-Segal Evalution

Subdivision Generated Triangles?

- Need fast construction of neighbors
- Need cheap storage
- O(n) reconstruction for Winged Edge & Triangle pointers…can localize, but ugly
- Adaptive will need parent & Child triangle pointers
- Parent knows neighbors

Restricted Triangular Quadtrees

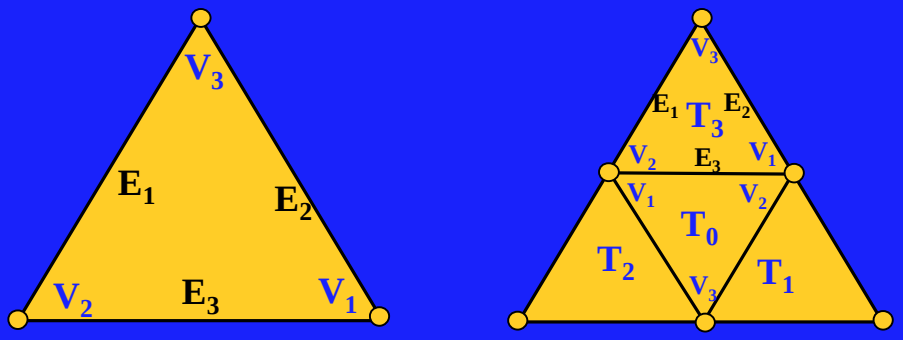


Quadtree Neighbor Finding

- Nearest Common Ancestor (NCA)
  - O(logn)
  - But thankfully: E(1) [Samet89]
- Mem usage:
  - 1 parent ptr, 1 ptr to 4 child block,
- Provides: parent, children, neighbors implicitly

Triangular Quadtree Component Naming

1. Name edges
2. Name vertices
3. Subdivide, name triangles

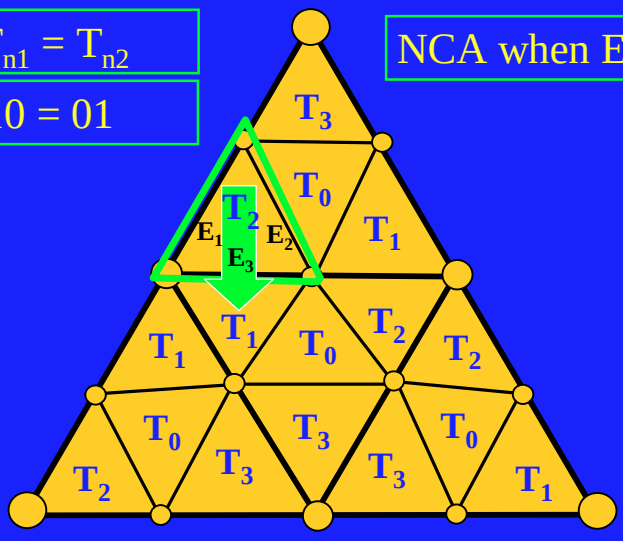


Finding NCA, Finding Neighbor

E_s xor T_{n1} = T_{n2}

11 xor 10 = 01

NCA when E_s = T_{n1}



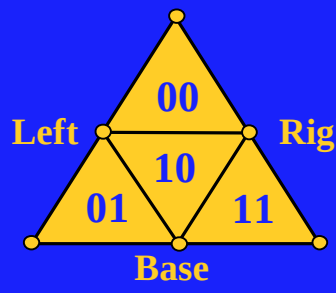
NCA Caveats

- Doesn’t cache well…lots of random access
- Missed branch predictions…P3, P4
- Not suitable for geometry engines

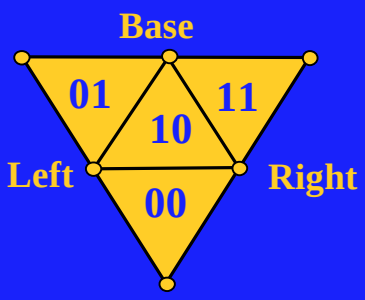
Lee-Samet Addressing

- Binary codes, Oct-Tree like addressing
- Direct addressing of triangles
- No NCA-style Memory traversals
- Neighbors computed with binary operations

Lee-Samet Addressing



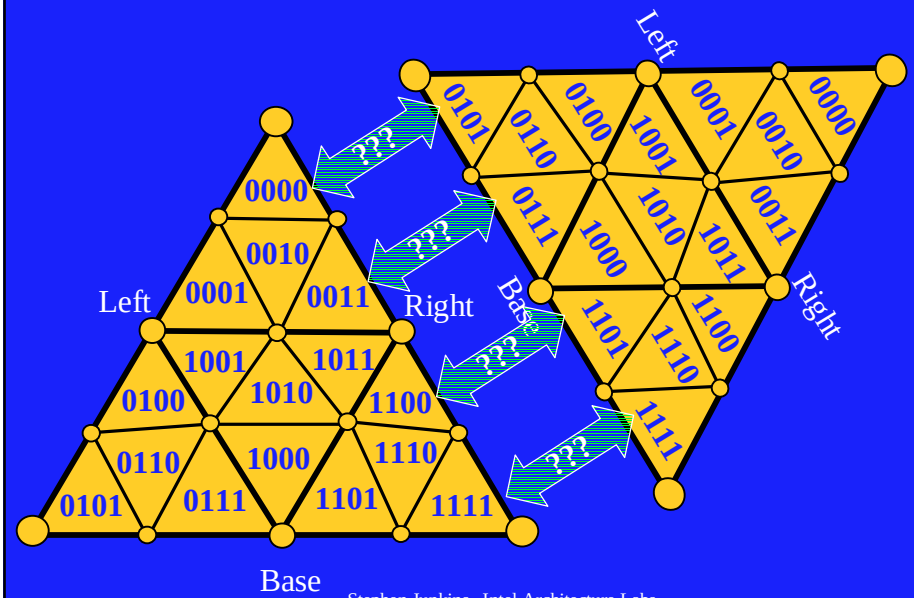
"Tip Up" Labeling



"Tip Down" Labeling

- Concatenate to form address
- Left/Right Easy, Base requires bit munging
- Address faults

Neighbors Across Control Mesh



Lee-Samet-Extended

	Left	Right	Base
Left	**00→01 01→00**	**01→11**	**00→11**
Right	**11→01**	**11→00 00→11**	**00→01**
Base	**11→00**	**01→00**	**01→11 11→01**

```
GetNeighborTriangle (triangle, direction)
    address = ComputeLocalNeighbor(triangle.address, direction)
    If (address.faulted)
        baseNeighbor = GetBaseNeighbor(triangle, direction)
        neighDirection = GetOrientation(triangle, baseNeighbor)
        address = ComputeDistalNeighborAddress (
                triangle.address, direction, neighDirection)
    End if
    Return (triangle[address])
End
```

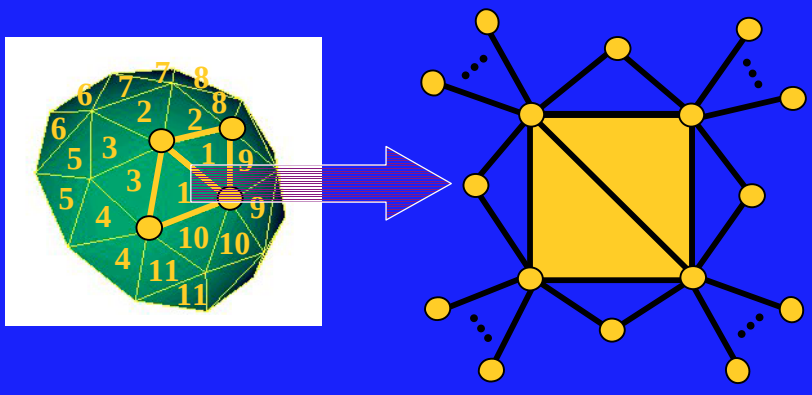

Lee-Samet-Extended Caveats

- Need linear list of triangles
  - Adaptive
  - Hash tables may help
- Base Neighbors and Distal Neighbors
  - ~20 bitwise binary ops
  - Register variables may help
- Performance close to NCA for levels 1-4

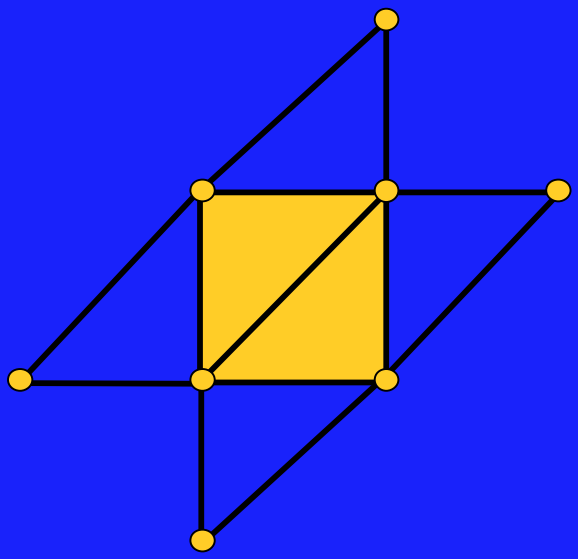
Pulli-Segal Evaluation

- Loop's Subdivision, or 1-area schemes
- Neighbors via 2D arrays
- Very memory efficient, Sliding window
- HW! SGI GE Microcode

Pulli-Segal: Pair up Triangles

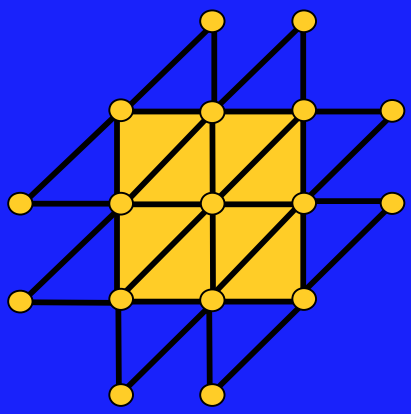


Pulli-Segal Evaluation



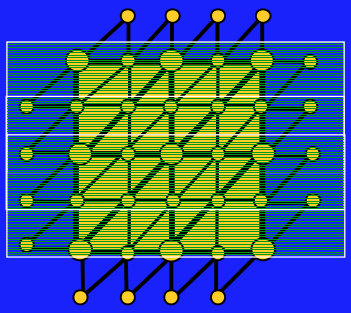
Stephen Junkins Intel Architecture Labs

Pulli-Segal Evaluation



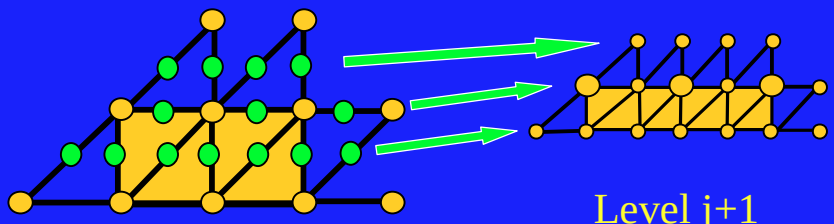
Stephen Junkins Intel Architecture Labs

Pulli-Segal Evaluation



Stephen Junkins Intel Architecture Labs

Sliding Window



Level j

Level j+1

```
SUBDIVIDE (level, last)
    If (level != last)
        Split horz. Edges
        SUBDIVIDE(level+1, last)
        Split vert. & diag. Edges
        SUBDIVIDE (level+1, last)
    Else
        Render
End
```

$Mem_j = 3 X (2^j + 3)$

Stephen Junkins Intel Architecture Labs

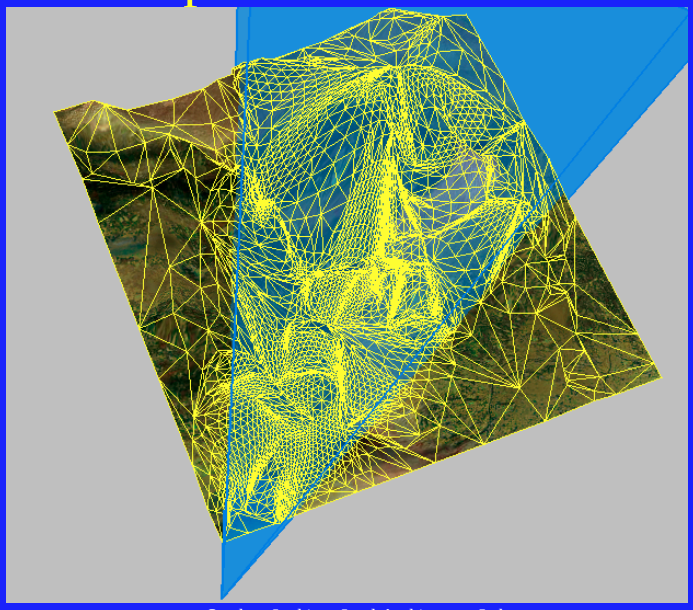
Pulli-Segal Caveats

- Redundant computation, amortized at level 3
- Special case: Control mesh vertices
- Orphan triangles
- Not suitable for Butterfly, etc.

General Performance Concerns

- Lots of memory
  - Approximating vs. Interpolating
- Dynamic LOD:
  - 1-4 levels of subdivision
  - Geo-morphing
- Limit Surface projection
- Vertex arrays not always practical
- Tristrips?
  - Pulli-Segal

Adaptive Subdivision

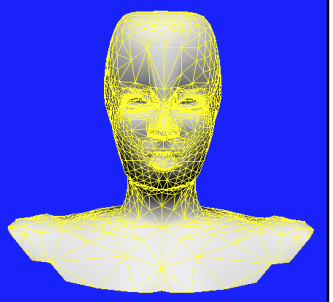


Stephen Junkins Intel Architecture Labs

Adaptive Metric

- Each Triangle: subdivide, consolidate, sustain
- 3 Part Metric:
  - View frustum test
  - Backfacing test
  - Screen space error test:

$$\delta^2\left(\|v-e\|^2-\left((v-e)\cdot n\right)^2\right)\geq k^2\|v-e\|^4$$

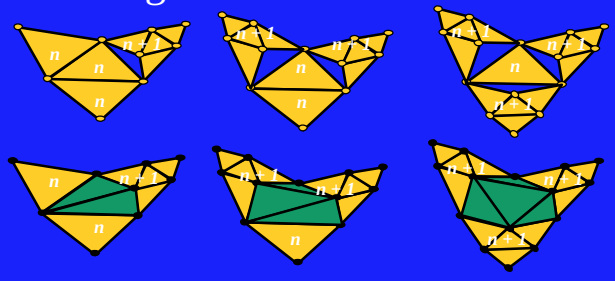


δ - *per triangle surface error*
v –triangle midpoint
e –eyepoint
n - triangle normal
k –pixel tolerance

Stephen Junkins Intel Architecture Labs

Adaptive Issues

- Crackfilling costs



- Metric evaluation costs
- Support triangulation can add up
- Early triangulation, else geo-morph
- Vertex arrays not practical

Subdivision and Animation

- Excellent animation modeling
  - Articulated skeletons
- Re-compute full subdivision every frame
- Subdivide, then Animate
  - Bone weights are another attribute
- Animate, then Subdivide
  - Linear weighting technique

Demo

Conclusions

- Triangle pointers for base mesh
- NCA or P.S. for subdivision triangles
- Adaptive is cool…but payoff is difficult
- Animation is expensive
- HW support?

Bibliography

[Abrash97] Michael Abrash. *Michael Abrash's Graphics Programming Black Book*. Coriolis Group Books, 1997.

[Baumgart75] B. G. Baumgart, A PolyHedron Representation for Compute Vision. *Proceedings AFIPS Nat. Computer Conference*., Vol. 44, (1975).

[Catmull78] Ed Catmull and J. Clark. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer Aided geometric design*, Vol. 10, No 6, (1978).

[DeRose98] Tony DeRose, Michael Kass, and Tien Truong. Subdivision Surfaces in Character Animation. *Computer Graphics Proceedings, ACM SIGGRAPH 98* (1998).

[Dyn78] Nira Dyn and David Levin. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Transactions on Graphics*, Vol. 9, No 2, (1990).

[Hoppe97] Hughes Hoppe. View-Dependent Refinement of Progressive Meshes. *Computer Graphics Proceedings, ACM SIGGRAPH 97* (1997).

[Junkins99] Stephen Junkins. Fast Triangle Neighbor Finding for Subdivision Surfaces. Intel Architecture Labs Internal White Paper. (September, 1999 *)*

[Lee98] Charles Loop. Smooth Subdivision Surfaces Based on Triangles. M.S. Thesis. department of Mathematics. University of Utah. (August 1987).

[Loop87] Michael Lee and Hanan Samet. Navigating through Triangle Meshes implemented as Linear Quadtrees. Tech. Rep. CS-TR-3900. University of Maryland CS Dept. (1998).

[Pulli96] Kari Pulli, and Mark Segal. Fast Rendering of Subdivision Surfaces. *Proceedings of the 7th Eurographics Rendering Workshop*. (June 1996).

[Samet90] Hanan Samet, *Applications of Spatial Data Structures.* Addison-Wesley, Reading, MA, (1990).

[Schröder98] Peter Schröder, Denis Zorin, Tony DeRose, David Forsey, Leif Kobbelt, Michael Lounsbery, Jörg Peters. *Subdivision for Modeling and Animation, ACM SIGGRAPH 2000 Course Notes* (1998).

[Zorin96] Denis Zorin, Peter Schröder, and Wim Sweldens Interpolating Subdivision for Meshes of Arbitrary Topology. *Computer Graphics Proceedings, ACM SIGGRAPH 96* (1996).

Displaced Subdivision Surfaces

Aaron Lee
Princeton University

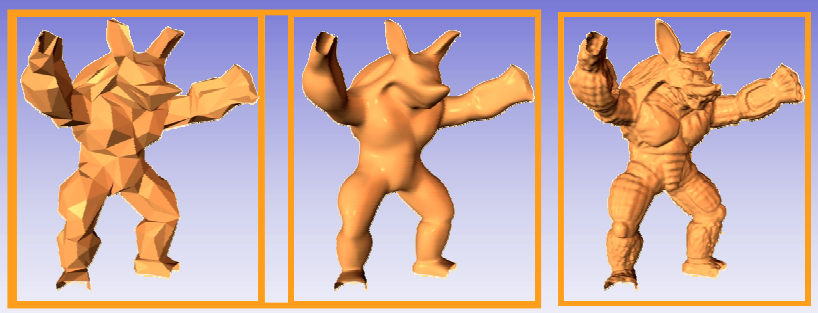
Hugues Hoppe
Microsoft Research

Henry Moreton
Nvidia

1

Our Approach

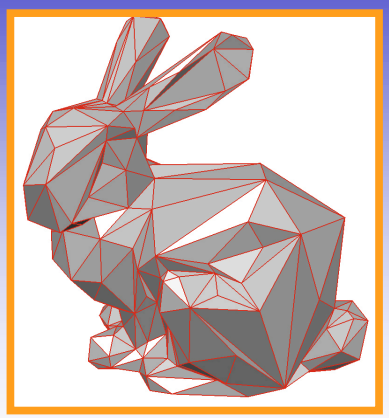
DSS = Smooth Surface ⬎ Scalar Disp Field

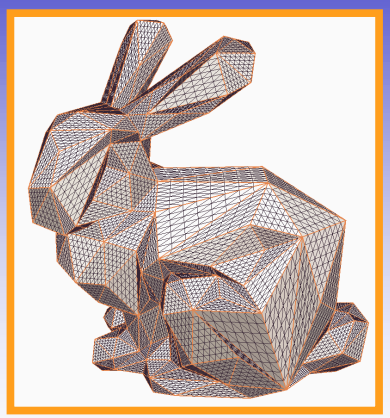


Control mesh | Domain Surface | Displaced Subdivision Surface

2

Representation Overview





3

Advantages of DSS

- Intrinsic parameterization
  - governed by a subdivision surface
  - no storage necessary
  - Capture details as scalar displacement
- Unified representation
  - Same sampling pattern and subdivision rule for geometry and scalar displacement field

4

What is Different About DSS?
vs. displacement maps

DSS are forward-mapped displacement mapped subdivision surfaces with a well-defined magnification filter

- Forward mapping makes sense…
- The magnification filter is the subdivision function.

5

Forward Mapping Makes Sense

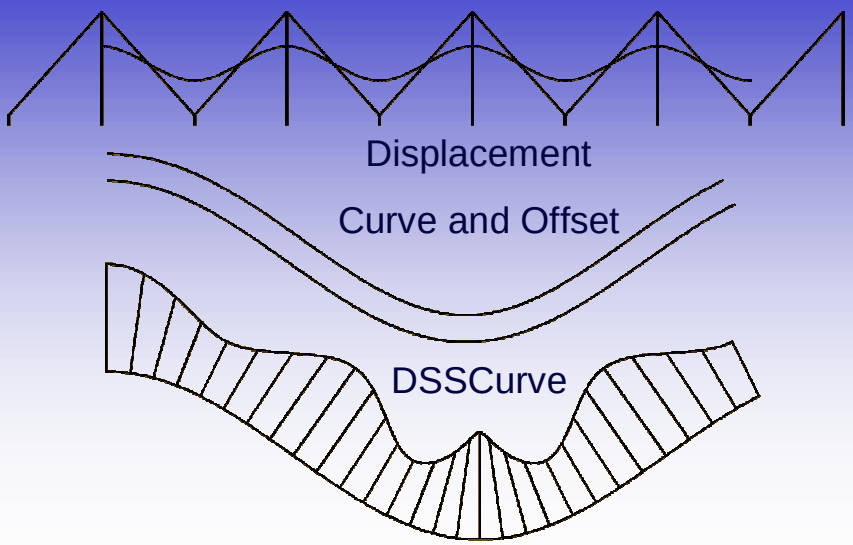
- Sampling occurs in object space
- Midpoint subdivision refinement
- Any filtering is redundant frame-to-frame
- Also true for displaced triangles …

In contrast

- Texture sampling occurs in screen space

6

2D Magnification example…



7

Analytic Behavior

- C^1 continuous everywhere except at extraordinary vertices
- Can be used to calculate bump map

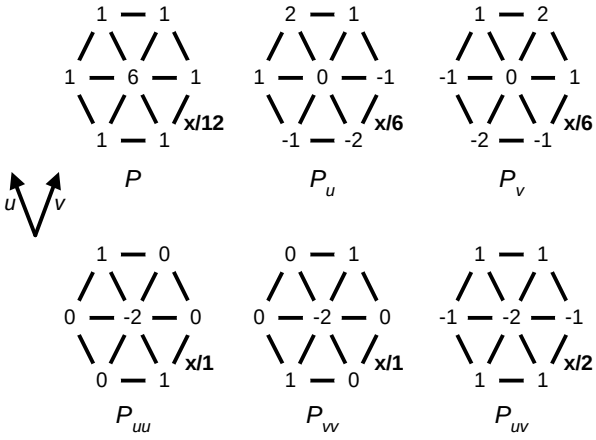
$$\vec{S} = \vec{P} + D\hat{n}$$

$$\vec{S}_u = \vec{P}_u + D_u\hat{n} + D\hat{n}_u$$

$$\hat{n}_u = \frac{\vec{n}_u - \hat{n}(\vec{n}_u \cdot \hat{n})}{\|\vec{n}\|} \quad \text{and} \quad \vec{n}_u = \vec{P}_{uu} \times \vec{P}_v + \vec{P}_u \times \vec{P}_{uv}$$

8

Normal Evaluation



9

Bump Map Rendering



134,656 faces

8,416 faces

526 faces

10

Normal Map Calculation

$$\left\{\vec{n}_{base} \quad \vec{t}_{base} \quad \vec{b}_{base}\right\} \cdot \hat{n}_{map} = \hat{n}_{dss}$$

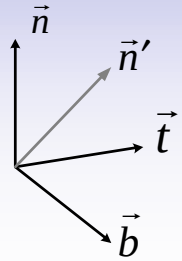
$$\hat{n}_{map} = \left\{\vec{n}_{base} \quad \vec{t}_{base} \quad \vec{b}_{base}\right\}^{-1} \cdot \hat{n}_{dss}$$

11

Bump, Stretch & Squeeze

Static bump map

- Stretching should flatten bumps
- Squeezing increases normal variation



$$\left\{\frac{\vec{S}_u}{\left\|\vec{S}_u\right\|^2} \quad \frac{\vec{S}_v}{\left\|\vec{S}_v\right\|^2} \quad \hat{n}\right\}$$

12

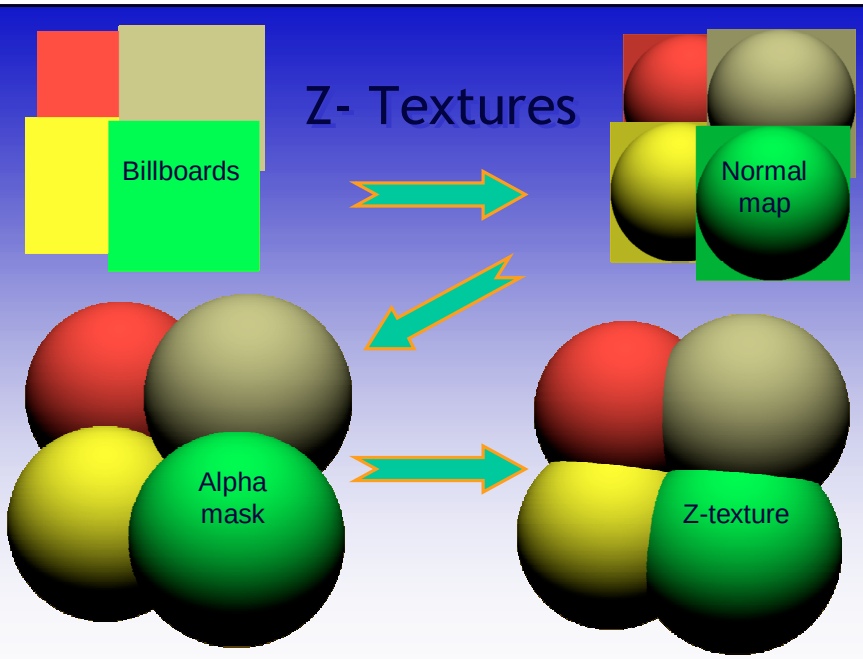
Improvements on Bump Mapping

Bump mapping simulates lighting of a rough surface…

- It does nothing for occlusion…
  - Silhouettes
    - We can't relocate pixels –adaptive tessellation
  - Z-fighting
    - We can adjust Z per pixel.

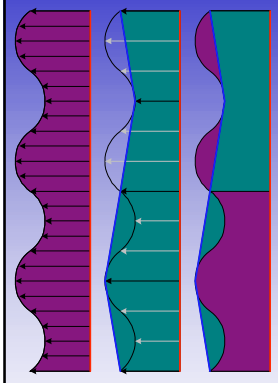
13

Z- Textures



14

Z-texture for Displacement Correction



$$\vec{p}'_e = \vec{p}_e + \Delta\hat{n}_e$$

$$z'_c = z_c + \Delta\left(\hat{n}\cdot\mathbf{T}_{\text{proj}}[3]\right)$$

$$w'_c = w_c + \Delta\left(\hat{n}\cdot\mathbf{T}_{\text{proj}}[4]\right)$$

$$z'_c = z_c + \Delta_{\text{B}}\left(\hat{n}\cdot\mathbf{T}_{\text{proj}}[3]\right) - \Delta_{\text{D}}\left(\hat{n}\cdot\mathbf{T}_{\text{proj}}[3]\right)$$

$$w'_c = w_c + \Delta_{\text{B}}\left(\hat{n}\cdot\mathbf{T}_{\text{proj}}[4]\right) - \Delta_{\text{D}}\left(\hat{n}\cdot\mathbf{T}_{\text{proj}}[3]\right)$$

15

Z-texture Characteristics

- Addresses Z-fighting problems
- Incorrect occlusion



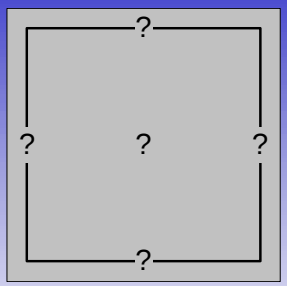
Interaction with shadow buffers…

- Identifier-based shadow buffers are incompatible with z-textures
- Depth-based shadow buffers exploit planar triangles

16

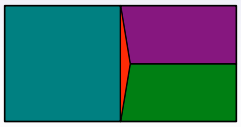
Adaptive Tessellation

- Test edges
- Test interior
- Interpolate for "Flat" vertices
- Draw degenerate triangles



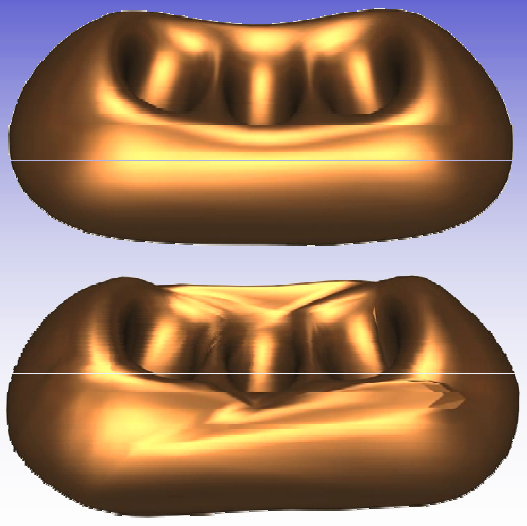
Works for pixel lighting





17

Displaced Triangles Don't Animate Well

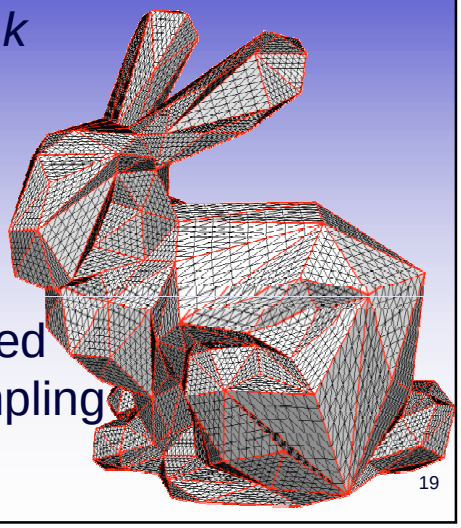


18

Variable *k* is Non-Trivial

Examples are uniform *k*

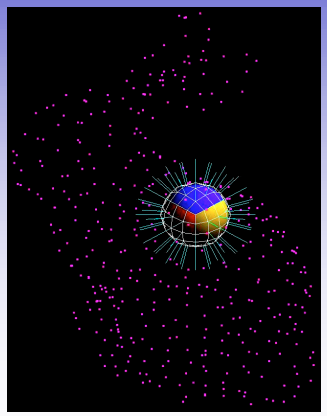
To guarantee smoothness the boundaries of finer regions must be forced to match coarse sampling.



19

DSS Construction

- Published work is top-down fine→coarse
- Range-scan processing better bottom-up.
- Modeling also bottom-up.



20