Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Mediensysteme

Diplomarbeit

Multiresolution Displacement Mapping on Subdivision Surfaces

Przemyslaw Musialski
Matrikelnummer: 992181

1. Gutachter: Prof. Dr. Charles A. Wüthrich
2. Gutachter: Prof. Dr. Klaus Gürlebeck

Datum der Abgabe: 08.06.2007

Weimar - Wien - Mai 2007

Abstract

In computer graphics subdivision surfaces is a technique to create smooth surfaces out of coarse polyhedral nets. Moreover it is capable of creating excellent high-density tessellations on todays' hardware in real-time. Additionally, due to the recursive generation nature it is also ideally suited for adding geometric detail in different resolutions.

When modeling real world surfaces it is possible to use image samples to recover wrinkled characteristics of a material and apply these on dense discrete meshes in the form of vertices displacement. Material characteristics are a mixture of features of different sizes which can be recovered by a frequency decomposition of an input height map. These sub-bands can be successively applied on different resolutions of the subdivision evaluation process in order to achieve naturally looking objects.

Although this can be done in software, the resulting amount of data and the processing time are prohibitively large. This thesis presents a method for computing displaced subdivision surfaces on the GPU that can render overlapping patches of the surface independently from each other without loss of mesh coherence. Thus, this method is ideally suited for on-the-fly generation of geometric detail as demanded by the viewing position of a rendering application.

Zusammenfassung

Subdivision Surfacs ist eine Technik in der Computer Graphik, die es möglicht macht glatte Oberflächen aus groben Netzen zu generieren. Zusätzlich ermöglicht sie es exzellente, hoch aufgelöste Tesselierungen auf heutigen Grafikbeschleunigern in Echtzeit zu erstellen. Des Weiteren sind Subdivision Surfaces wegen ihrer rekursiven Natur ideal dazu geeignet um geometrische Details auf ihren verschieden Auflösungen anzubringen.

Um echt wirkende Oberflächen modellieren zu können ist es mit Zuhilfenahme von Bildmustern möglich, faltige und runzelige Charakteristiken der Materialien wiederherzustellen, und diese in Form von Verschiebungen von Oberflächenpunkten auf das Netz anzubringen. Die Materialcharakteristiken sind eine Mischung von Eigenschaften von verschiedenen Größen und können mit einer Spektralanalyse aus einem Musterbild wiederhergestellt werden. Diese Spektralkomponenten können schrittweise auf nacheinander folgende Netzauflösungen angewendet werden um natürlich wirkende Objekte zu kreieren.

Obwohl das Ganze in Software umgesetzt werden kann, die resultierende Datenmenge wie auch die Bearbeitungszeit sind untragbar groß und lang. Diese Diplomarbeit präsentiert eine Methode zur Berechnung von Displaced Subdivision Surfaces auf einem 3D-Beschleunigungsprozessor, welche ein Netz aus überlappenden Stücken unabhängig voneinander rendern kann ohne dabei die Kontinuität der Oberfläche zu stören. Aus diesem Grund ist diese Methode ideal dazu geeignet, geometrische Details in Echtzeit und auf die Bedürfnisse der Szenen- und Kameraeinstellung angepasst, zu erstellen.

Acknowledgments

I would like to express my gratefulness to all those who who contributed to this work. I would especially like to thank my supervisor Prof. Dr. Charles Wüthrich for his advice, support and patience not only during the writing of this thesis but also over all the years at the Bauhaus-University.

A thank you to Dr. Robert Tobler for giving me the opportunity to complete this work at the VRVis Research Center in Vienna and for his support there.

A further very special thank you to Chrystoph Toll for the discussions, hints, help, patience and just his being there for me at the Bauhaus-University.

Finally I would like to thank Karina for the support in the last days and my mother without whom none of this would be possible.

Danksagung

Ich möchte mich bei all denen die zu dieser Arbeit beigetragen haben recht herzlich bedanken. Ganz besonders möchte ich meinem Betreuer, Prof. Dr. Charles Wüthrich danken. Dieser Dank gilt nicht nur für die Betreuung, Unterstützung und Geduld während der Zeit der Diplomarbeit, sondern auch für meine gesamte Studienzeit an der Bauhaus-Universität in Weimar.

Ein Dankeschön gebührt auch Dr. Robert Tobler dafür, dass er mir die Möglichkeit gegeben hat diese Arbeit am VRVis Forschungszentrum in Wien zu fertigen und mich dort unterstützt hat.

Ein weiteres, ganz besonderes Dankeschön gebührt Chrystoph Toll für die Gespräche, seine Ratschläge und Hilfe, seine Geduld und auch sein Dasein für mich an der Bauhaus-Universität.

Zum Abschluss möchte ich mich bei Karina für die Unterstützung in den letzten Tagen und bei meiner Mutter, ohne welchen nichts von dem hier möglich gewesen wäre, herzlich bedanken.

Contents

List of Figures

1. Introduction

1.1. Motivation

Real-life surfaces are very rarely completely smooth. Most natural surfaces exhibit many levels of roughness and coarseness, especially if they are inspected in detail. Therefore many computer-generated models of real-life objects that are rendered with perfectly smooth surfaces appear sterile and artificial. Over the years, computer graphics have developed several approaches to reduce this problem. One of the methods in this category is bump mapping introduced by Blinn [Bli78], followed by its improved descendant, parallax mapping [OBM00]. Both of these methods do not influence the actual geometry of the underlying model, but change its rendered appearance.

On the other hand, a whole area of computer graphics research deals with subdivision surfaces, a powerful tool for creating objects with smooth surfaces. Due to their nature, subdivision surfaces can be either rendered using successively finer approximations, or evaluated exactly for selected surface parts. Since subdivision surfaces can be generated from base geometry of any topology, they are very easy to handle in the modeling process and have therefore become the tool of choice in applications which aim for the highly realistic appearance of smooth models.

In this thesis a method to apply high resolution geometric detail on subdivision surfaces in in real-time closeup views will be presented. The additional surface detail can be generated artificially or based on an analysis of a real surface sample. During the rendering process, it can be applied automatically in the form of multi-band displacement maps on the different subdivision levels of the underlying surface.

In order to extend the variety of appearance which can be achieved with this approach, the application of the surface detail can be performed locally at each subdivision step and can be modified according to various static and dynamic object properties. The limit of this modification is of course set by the speed and capabilities of the rendering hardware.

1.2. Goal of the Thesis

The main goal of this thesis is to develop a method for generating naturally looking subdivision surfaces. It is a part of the basic research project GeomeTree [VG04] of the VRVis Company in Vienna, which aims at developing a wide variety of techniques for vegetation visualization. It is split into several sub-areas beginning with a rendering of complete landscapes of forests to creating small details in extreme closeup views. In previous work Catmull-Clark subdivision surfaces over a set of loosely joined meshes have been implemented by the author [MTM07]. In the current thesis the focus has been set on small features on the otherwise completely smooth surfaces. In order to achieve this, it become apparent very quickly that only very high tessellated surfaces provide a background for synthesizing naturally looking details. While subdivision surfaces as implemented in software do not allow for such high resolutions in reasonable time, the decision was made to implement this technique in graphics hardware.

In order to generate details on the surface, the technique of displacement mapping seems to be the tool of choice. While displacement is usually applied on already tessellated meshes, in terms of recursive subdivision evaluation it seems to be an interesting issue that takes advantage of the multiresolution representation of the surface. Thus, the approach of developing a procedural displacement in different resolutions leaning on reconstruction of patterns on given samples will be attempted.

1.3. Organization of the Thesis

The thesis is organized in six main chapters. After this introduction, the next chapter will give an elaborate introduction into the theoretical backgrounds of the techniques used in this work. First of all subdivision surfaces will be presented in terms of their relation to splines and further on, their recursive evaluation method as well as a tool for their analysis will be addressed. Next, the technique of displacement mapping will be described as well as some important issues of offset surfaces in the differential geometry. Finally, a technique of image analysis will be described, which allows to decompose a sample into its frequency spectrum in the spacial domain.

Chapter 3 offers an overview of other subdivision evaluation methods as well as several attempts of displaced subdivision surfaces. Following that, chapter 4 concentrates on the actual method developed in this thesis. It will introduce the concepts and the implementation in detail. Chapter 5 provides an evaluation of the developed technique in both, performance as well as visual quality issues. Finally, chapter 6 discusses the developed technique, open questions and the problems that arose during the development.

Additionally, in the very end color plates present figures of the obtained results in high quality.

2. Theoretical Background

This chapter will present mathematical foundations for techniques used in this work. After a brief general overview of the subdivision approach, a comprehensive introduction into the cubic B-spline based Catmull-Clark scheme will be given. Next, the technique of displacement mapping will be described, and finally – also used in this work – the decomposition of images into their particular frequency bands will be addressed.

2.1. Subdivision Surfaces

Subdivision surfaces is an approach to describe a surface by using a sequence of consecutively refined polygonal nets out of an initial model. Like the initial polygonal model, the surface can be of any shape, size and topology – it is not limited to a rectangular patch. Unlike the input, the surface itself is – depending on the particular subdivision scheme – usually perfectly smooth. The subdivision technique for surfaces has been introduced quite some time ago by Catmull and Clark [CC78] as well as Doo and Sabin [DS78]. For a long time the theoretical foundation of the subdivision process was not as thorough as other modeling techniques such as B-splines and the more general NURBS. Thus it took a while for subdivision methods to become widely adopted. In the last decade this has been rectified by the introduction of methods to analyze and evaluate subdivision surfaces at any point [Rei95], [Sta98], a method for extending subdivision surfaces for approximating NURBS [SZSS98], the addition of normal control to subdivision surfaces [BLZ00], and a method to closely approximate Catmull-Clark subdivision surfaces using NURBS patches [Pet00]. A number of other extensions to subdivision surfaces [DKT98, LMH00] have established them as the modeling tool of choice for generating topologically complex, smooth surfaces in many fields of computer graphics.

One of the most known and established subdivision schemes has been introduced by Catmull and Clark in 1978 [CC78]. Due to its properties like i.e. close relation to B-spline patches as well as to its continuity qualities, this work is based on that scheme. Nevertheless, this section will give an overview over general characteristics of subdivision and further on the detailed mathematical background of the Catmull-Clark scheme.

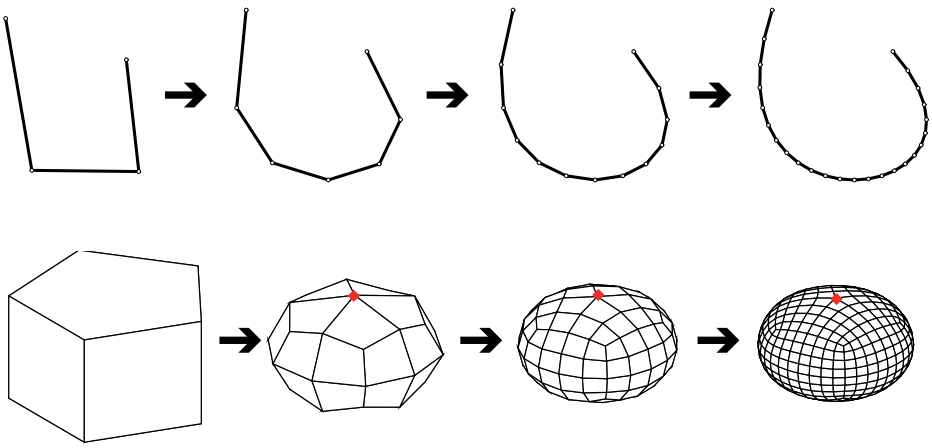


Figure 2.1.: *Subdivision: example in 2D and 3D. Figure from [SZD+98].*

2.1.1. Overview

Although nowadays there exists many different schemes which all have their own individual properties, all are based on a set of common characteristics. Basically, the main goal of subdivision surface techniques is the use of recursive refinement to obtain smooth surfaces out of arbitrary polygonal input (see Figure 2.1). In this section, a short general overview about those characteristics is given. For a better overview about different subdivision schemes refer to Sharp [Sha00] or Maierhofer [Mai02]. For more involved details see Schröder *et al.* [SZD+98] and Warren and Weimer [WW01].

2.1.1.1. Convergence and Continuity

For most subdivision schemes it has been proven that they mathematically build a sequence which finally converges towards a limit surface. Since the Catmull-Clark scheme is derived from B-splines it obviously converges to the bivariate cubic B-spline function at regular mesh regions. Also the second order continuity of the patches is ensured by this fact. Section 2.1.6.3 covers an extended examination of this issue.

A more involved situation appears at extraordinary points, where the surface is not exactly defined by the B-spline bases. This case as well as the continuity issue will be also discussed more in detail in section 2.1.6.4.

2.1.1.2. Approximating and Interpolating Scheme

Subdivision surfaces can be classified into *approximating* and *interpolating* schemes. The difference lies in the way of how the smooth surface wraps the initial control mesh [Sha00].

Approximating. In an *approximating* scheme the vertices of the control net do not lie on the surface itself. At each step of subdivision the existing vertices in the control mesh are moved closer to the limit surface. The benefit of an approximating scheme is that the resulting surface is very fair, having few undulations and ripples. Even if the initial net is of very high frequency with sharp points, the scheme will tend to smooth it out because the sharpest regions move the furthest onto the limit surface. On the other hand, this can be also seen as a drawback. It can be difficult to work with, as it is not easy to envision the end result while building the control net, and it may be difficult to craft more undulating, rippling surfaces as the scheme tends to smooth them out. Catmull-Clark subdivision is an approximating scheme.

Interpolating. In contrast, in an *interpolating* scheme the vertices of the control mesh actually lie on the limit surface. This means that at each step the existing vertices of the mesh are not moved at all and new vertices are placed in a way to move the net against the limit surface. The benefit of this is that it can be much more obvious from the control net what the final surface will look like. However, it can sometimes be rather difficult to get an interpolating surface to look as desired, since the surface can develop unsightly bulges in areas where it strains to interpolate the vertices and still maintain its continuity.

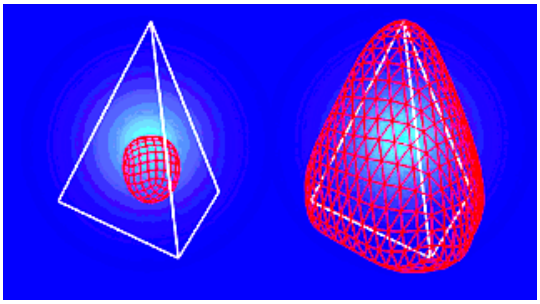


Figure 2.2.: *Two schemes subdivide a tetrahedron. The left scheme is approximating, and the right is interpolating. From Gamasutra [Sha00].*

2.1.1.3. Uniform vs. Nonuniform and Stationary vs. Nonstationary

There exists another set of characteristics that introduces four additional terms. A scheme can be either *uniform* or *nonuniform*, and it can be either *stationary* or *non-stationary*. These terms describe how the rules of the scheme are applied to the surface. If the scheme is uniform, all areas of a control net are subdivided using the same set of rules and the resulting submeshes' control points are equidistant in the parametric space, whereas a nonuniform scheme might subdivide one edge in one way and another adjacent edge in another way. This results in a non-uniform spacing of the control points in the parametric space.

If a scheme is stationary, the same set of rules is used to subdivide the net at each step. A non-stationary scheme, on the other hand, might first subdivide the net one way, and than the next time around use a different set of rules.

The Catmull-Clark subdivision, which is the main subject to this work is a uniform and stationary scheme.

2.1.1.4. Dual vs. Primal

Further on, the refinement rules can be either based on vertex insertion (*primal*) or on corner cutting (*dual*). The basic difference is given by the way how the integer coordinates of the basis functions are positioned in parametric space. In *primal* schemes the coefficients remain on their positions on the integer grid ($\mathbb{Z}$) and in k subdivision steps, the new vertices are placed on the grid $\frac{1}{2^k}\mathbb{Z}$.

In this case, when the supported interval of the basis function has an odd magnitude, the translates of the basis functions lie on the midpoints of segments. To inset new points, the grid has to be shifted in order to provide proper positions. Hereby, the function $D(\mathbb{Z})$ performs a shift of the interval defined by $\frac{1}{2^k}\mathbb{Z}$ into the midpoints of it. In these terms, the new inserted vertices can be placed at $D(\frac{1}{2^k}\mathbb{Z})$ positions of the parametric grid, and a scheme with this structure is called *dual*. Geometrically it can be interpreted as 'corner cutting' where the original vertex is 'cut off' and the resulting gap is closed by joining the former midpoints of the segments by a new edge (refer Figure 2.3, right hand side).

A more detailed discussion about the definition of the subdivision procedure and its basis functions will be supplied in section 2.1.3.

2.1.1.5. Input and Output Polygons

Another characteristic of a scheme, albeit less significant than the prior ones, is whether it is triangular or quadrilateral. As the names would imply, a triangular scheme operates

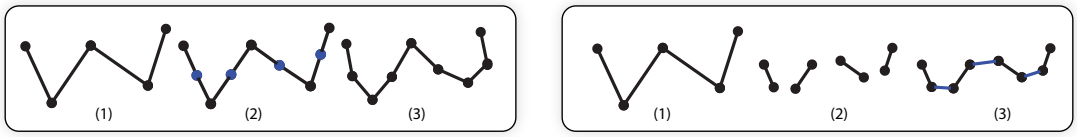


Figure 2.3.: *Primal and dual subdivision shown in a two-dimensional case. Left: vertex insertion (primal). Right: corner cutting (dual)*

.

on triangular control nets, and a quadrilateral scheme operates on quadrilateral nets. Clearly, it would be inconvenient if one has to restrict oneself to these primitives when building models. Therefore, most quadrilateral schemes (including the Catmull-Clark, which will be discussed here) have rules for subdividing n-sided polygons. For triangular schemes one generally needs to split the polygons into triangles before handing them over to be subdivided, which is easy to do, since every polygon can be subdivided into triangles. One downside is that for some schemes, the way the polygons are broken can change the limit surface, although the changes are usually minor.

Convex Polygons. In order to obtain proper limit surface approximation, it should be mentioned that only *convex polygons* ensure that the resulting surface remain manifold. Although the algorithm will also perform for any input, concave polygons would introduce mesh fold-overs or self-intersections. Biermann *et al.* [BLZ00] have dealt with this problem on the boundaries of meshes and have introduced a way to handle concave corners on boundaries while Zorin [YZ01] has developed a scheme for non-manifold meshes. In general subdivision, as with Catmull-Clark, there is no solution for handling concave faces in the interior of the input domain except a prior split into convex faces.

2.1.2. Subdivision of Piecewise Polynomial Curves and Surfaces

After a short overview of the subdivision approach has been given, this section will present the basic idea behind subdivision on the example of piecewise polynomial curves.

The goal of designing models on the computer is very often to create objects defined by smooth and curved outlines. In computer graphics curved geometry has become commonplace during the last 40 years. Its advantages are — beside its aesthetic appearance — the ability to describe a huge class of objects and additionally their rather easy way of control. There are many types of curve representations i.e. explicit, implicit or parametric. In this thesis only the latter ones will be addressed.

As *piecewise polynomials* one can denote a class of curves and surfaces which are defined by polynomial parametric equations of a given order (which is usually at most 4) and a number of *control points*. A long curve, path or a wide surface can be composed of several smaller 'pieces' that are connected together at their endpoints. One important feature of piecewise polynomial curves is the continuity at the junctions between the endpoints of the 'pieces'. In most cases it is desirable to achieve a smooth transition from one curve to another, such that the parametric space is continuous to some order n (called *parametric continuity* C^n). According to this, a curve is said do be C^1 continuous, if the tangent vectors are equal in both magnitude and direction at the join point. More precisely, not only the tangents but also the first derivatives at this point must be equal[1]. If the tangent vectors have the same direction but have different magnitudes, than the curve is said to be G^1 continuous (*geometric continuity*). According to the number of equal derivatives of the curves on a junction yields the degree n of the particular continuity. Usually such curves are called *splines*, a term derived from the flexible spline devices used by shipbuilders and draftsmen to draw smooth shapes.

There are many different approaches for defining splines in computer science (i.e. Catmull-Rom, Hermite, B-Spline, NURBS). However, this thesis will focus on B-spline curves and surfaces of fourth order and the associated subdivision schemes, especially the Catmull-Clark scheme for smooth surfaces.

In the previous section a short overview of different subdivision properties has been presented. Since one important subject of this work is recursive subdivision, following sections will go more into the mathematical background of subdivision and derive a straight-forward example how subdivision is related to parametric polynomial curves and surfaces. Further on, subsection 2.1.3 will present a more formal definition of issues mentioned here.

2.1.2.1. Curves in Matrix Form

In this section (piecewise) polynomials will be explained. It is intended to explain the mathematical notation used in this work. It will be introduced with the example of a linear polynomial curve in $\mathbb{R}^3$. Usually, polynomial curves will be expressed in their matrix form as follows:

$$p(u) = \mathbf{U}^T\mathbf{M}\mathbf{G} = \begin{bmatrix} u & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \end{bmatrix} \tag{2.1}$$

[1] There is a difference between tangent and C^1 continuity. The latter one is stronger, since there are cases where exactly equal tangents do not ensure C^1 continuity [SZD+98]

where $\mathbf{U}^T$ denotes the interpolation variable and $\mathbf{M}$ the coefficient matrix for the particular curves' *basis functions*. These functions, also often called *blending functions* define the influence of each control point on the curve. If this functions are uniformly spaced on the parametric axis – which means that the spacing and the supported interval of each one is always exactly the same – they are also called *scale functions* and the curve is referred to as *uniform*.

In this particular instance the blending functions are lines in interval $[0..1]$ and their sum is always 1:

$$\left.\begin{array}{l} b_0(u) = 1-u \\ b_1(u) = u \end{array}\right\} \text{for } u \in [0,1]$$

Rewriting those functions in matrix form yields:

$$\begin{bmatrix} 1-u \\ 0+u \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -u \\ u \end{bmatrix} = u \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} u & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$$

which delivers the desired variable vector $\mathbf{U}^T$ and the coefficients matrix $\mathbf{M}$.

Finally, vector $\mathbf{G}$ denotes the *geometric points*. The expression above shows a linear interpolation between two points P_0 and P_1 in the interval $[0..1]$. The parameter variable u has been chosen to clearly distinguish it from the global euclidean space variables x, y and z where the geometric control points P are defined. So each point $P \in \mathbb{R}^3$ is basically a vector of the form $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T \in \mathbb{R}^3$. For surfaces the parametric space will be extended by the variable v respectively.

To obtain a geometric position on a curve at some particular u the equation 2.1 can be expanded as follows:

$$p(u) = \begin{bmatrix} u & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \end{bmatrix} = \begin{bmatrix} (1-u) & u \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \end{bmatrix} = (1-u)P_0 + uP_1$$

Finally, the curve in equation 2.1 can be also expressed in an equivalent and often used summation notation:

$$p(u) = \sum_{i=0}^{1} b_i(u) P_i \qquad \forall\, i \in \mathbb{N}$$

2.1.2.2. DeCasteljau Subdivision for Bézier Curves

This subsection shows the close relation between subdivision schemes and parametric curves (which will be later extended to surfaces) in the example of the DeCasteljau algorithm. It is a method for recursive construction of Bézier curves. A more comprehensive derivation of curves will be given in a later section, this one is intended to show the basic idea without further background.

Bézier curves were introduced in 1962 by the French engineer Pierre Bézier, who used them to design automobile bodies. They rely on Bernstein polynomials as basis functions and can be controlled rather intuitively by a number of control points. For the cubic case used here, four points are necessary to express a piece of the curve, where P_0 and P_3 are the beginning and end points and P_1 and P_2 are two interior points. The vectors $P_1 - P_0$ and $P_2 - P_3$ form the tangents of the piecewise continuous curve at P_0 and P_3 respectively. This is shown in Figure 2.4.

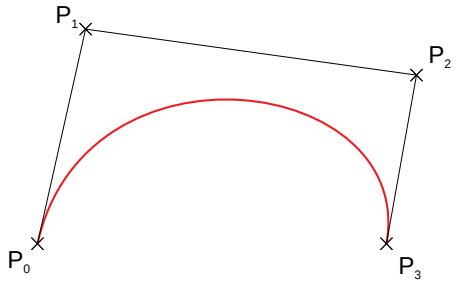


Figure 2.4.: *Cubic Bézier curve, Figure from [Wik07].*

Affine Transformations. To demonstrate an interesting albeit simple geometric interpretation of the subdivision procedure, the cubic Bézier curve should be considered as a flat arc on a two-dimensional plane. Now the DeCasteljau algorithm can be applied in order to construct the limit curve recursively. This procedure evaluates the positions of the points on the curve by bisecting the vectors defined by the four control points:

$$P' = \frac{1}{2}\left(\frac{1}{2}\left(\frac{1}{2}(P_1 - P_0) - \frac{1}{2}(P_2 - P_1)\right) - \frac{1}{2}\left(\frac{1}{2}(P_1 - P_2) - \frac{1}{2}(P_2 - P_3)\right)\right)$$

This works not only on the bisectors and can be also applied on other parametric values. Nonetheless, by considering the bisectors this procedure can be seen as an affine transformation of the control points in order to obtain a new set of points defining a further subsegment of the curve. Figure 2.5 left-most shows the trapezoid specified by the four points and the gray highlighted smaller trapezoid is a affine transformation of the superior one. According to this, a transformation matrix **S** can be defined to gain the

positions for new control points for the generated sub-segments. Since the transformation is mirror-symmetric with respect to the bisection of the vector $P_2 - P_1$, the resulting rule is a set of matrices:

$$\mathbf{S} = \left\{ \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{bmatrix}, \begin{bmatrix} \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \right\},$$

where the first matrix generates the left-hand side segment and the second matrix the right-hand side respectively. Figure 2.5 shows three recursions of the subdivision performed by a recursive application of by **S** defined affine transformations on each of the curves' segments.

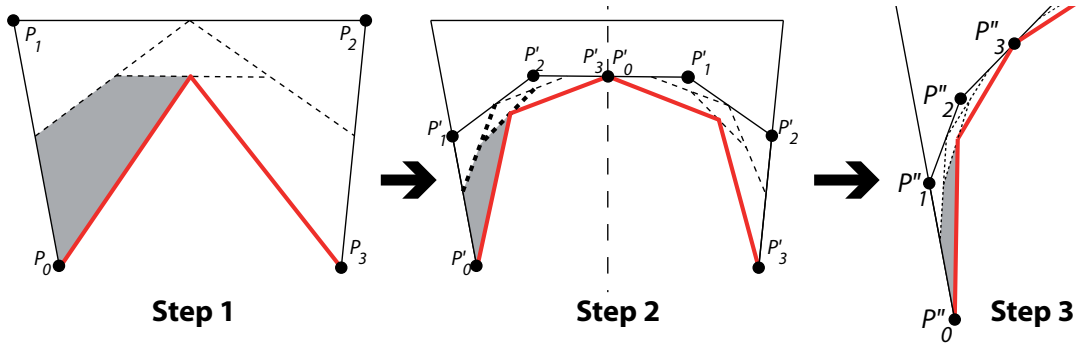


Figure 2.5.: *Three steps of recursive subdivision of a Bézier curve by the DeCasteljau algorithm. Each gray highlighted trapezoid represents an affine transformation of the left segment of its forefather in previous step. The last picture is shown partially and enlarged for better readability.*

In this way of constructing the curve, it is obvious that it can be evaluated uniformly over the whole length or even locally, by depth-first application of the matrix on chosen segments. Furthermore, the entire curve can be stated by the following subdivision recursion equation:

$$\mathbf{G}_k = \mathbf{S}\mathbf{G}_{k-1} \tag{2.2}$$

where k denotes the subdivision recursion and $\mathbf{G}$ the vector holding the control points. This equation will be the subject of many interesting relations in this thesis.

2.1.3. Subdivision of cubic B–Splines

Another class of piecewise polynomial curves are B-splines. They are, among other properties, a generalization of Bézier curves. Unlike the latter, B-splines provide local control and C^2 continuity everywhere. As a trade-off it can be mentioned that B-splines are also more difficult to grasp intuitively because they do not generally interpolate a set of points, but only approximate their positions.

In this section B-splines will be introduced in a rather detailed way where the presented 'low level' information will later be used to derive a general subdivision scheme for B-splines of any order. Further on, the thesis will focus on cubic B-splines and the bivariate surfaces defined by the tensor product of univariate splines. Moving ahead this way, the Catmull-Clark subdivision will be derived in the next section.

2.1.3.1. Basis Functions for cubic B-Splines

As described in section 2.1.2 parametric curves are defined by their geometric control points **G**, a coefficients matrix of the blending functions **M** and the interpolating variable **U** (refer to equation 2.1). Alternatively they can be stated in the summation notation:

$$p(u) = \sum_{i}^{n} b_i(u) P_i \qquad \forall\, i \in \mathbb{Z}\,.$$

In this representation the blending functions $b_i(u)$ which define the local influence of the control points P_i on the curve can be researched in more detail. In following the properties of basis functions will be derived.

Repeated Integration. First of all the basis functions can be generated via repeated integration of the *B-spline basis function of order one* [2] which is a simple constant straight line $y = 1$ in interval $[0..1]$ and zero elsewhere (also referred as Haar scaling function):

$$b^{[1]}(u) = \begin{cases} 1 & \text{if } u \in [0..1) \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Higher order basis functions are defined via repeated integration of the function of the previous order:

[2] The term *order* is basically equivalent the the term *degree* and refers to the highest exponent of the polynomial which defines a curve. In this thesis *order* will be denoted as one higher than the *degree* for the reason not to start indexing basis functions with zero. Thus the order is equal to the number of blending functions which define a piece of a curve of the respective degree.

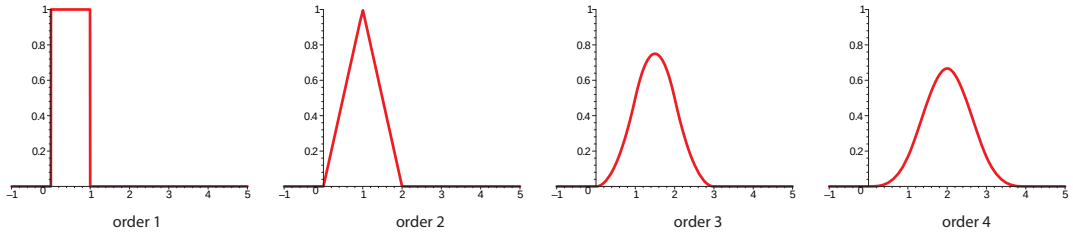


Figure 2.6.: *B-spline basis functions of orders 1,2,3, and 4.*

$$b^{[n]}(u) = \int_0^1 b^{(n-1)}(u-t)\,\mathrm{d}t\ . \tag{2.4}$$

Convolution. Equation 2.4 can also be rewritten as a convolution operation on two functions, defined as $f(u) = (g \otimes h)(u) = \int g(u)h(u-\tau)\,\mathrm{d}\tau$:

$$b^{[n]}(u) = \left(b^{[1]} \otimes b^{[n-1]}\right)(u) = \int_{-\infty}^{\infty} b^{[1]}(u)b^{[n-1]}(u-t)\,\mathrm{d}t\ . \tag{2.5}$$

This is possible because the first term $b^{[1]}$ as defined in equations 2.4 and 2.3 multiplied with $b^{[n-1]}$ inside the integral always constrains the integration for t to the bounds $[0..1]$. Finally, by applying the primitives, the Cox-DeBoor [DeB72] recursive formula can be derived:

$$b^{[n]}(u) = \frac{u}{n-1}b^{[n-1]}(u) + \frac{n-u}{n-1}b^{[n-1]}(u-1)\ . \tag{2.6}$$

Cubic B-spline. With this equation the basis functions for B-splines of any order n can be easily computed. For the interesting case of cubic B-spline (order 4, degree 3) the basis is:

$$b^{[4]}(u) = \begin{cases} \frac{u^3}{6} & \text{if } u \in [0..1) \\ -\frac{u^3}{2} + 2u^2 - u + \frac{2}{3} & \text{if } u \in [1..2) \\ \frac{u^3}{2} - 4u^2 + 10u - \frac{22}{3} & \text{if } u \in [2..3) \\ -\frac{u^3}{6} + 2u^2 - 8u - \frac{32}{3} & \text{if } u \in [3..4) \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

Now, to create and control a piecewise cubic curve $p(u)$ at least four control points have to be defined. To obtain proper blending functions for each of the control points,

functions presented in equation 2.7 have to be translated along the u-axis: $b^{[4]}(u-i)$. This can be observed in Figure 2.7, where the curve is fully defined for $-3 \leq i \leq 0$ by four scale functions b_i in range $[0..1]$ which belong to the control points P_i respectively.

For the four control points, referred as the vector $\mathbf{G} = \begin{bmatrix} P_i & P_{i+1} & P_{i+2} & P_{i+3} \end{bmatrix}^T$, the coefficients matrix $\mathbf{M}$ can be obtained by shifting $b_i(u-i)$ from equation 2.7 by $(u-i)$, where i is in $-3 \leq i \leq 0$. Than, by taking for each i the coefficients $b_i(u)$ at $[0..1)$ from equation 2.7 thay can be accumulated to the a matrix such that

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ P_{i+3} \end{bmatrix} . \tag{2.8}$$

Alternatively, there are also other ways to derive the matrix $\mathbf{M}$. For instance by setting up a linear equations system inspired by the property where the curve has to be C^2 continuous which implies, that its first and second derivatives have to match too. These properties supplemented by the fact that the blending functions sum to 1 give 16 linear independent equations. The solution of the system delivers the coefficients matrix . This method is presented in [Len03].

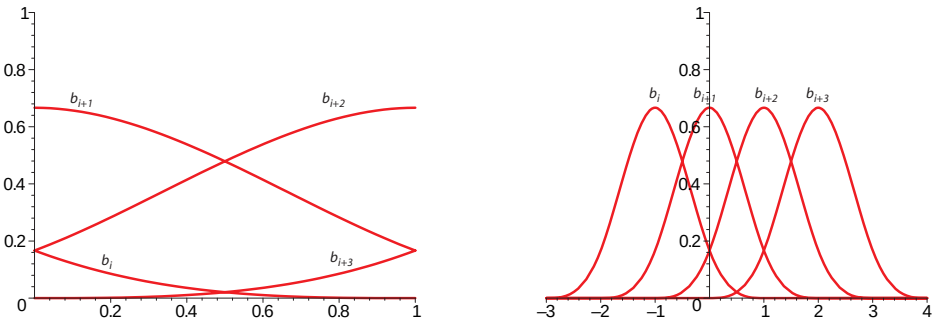


Figure 2.7.: *Cubic B-spline scale function. Left hand side plotted in range* $[0..1]$*, right hand side, in range* $[-3..4]$.

Linear Combination. Thus the curve $p(u)$ can also be seen as a linear combination of a number of shifted basis functions b_i (as in figure 2.7) each multiplied with a control point from $\mathbf{G}$. Due to the uniform spacing of the shifts, each blending function b_i has an influence on exactly the same range of the curve and also for this reason the curve is called *uniform*. Furthermore, to express a piece of the curve, the number of control

points (and blending functions) must be at least the same as the order of the curve. So the minimal curve of order n has the form:

$$p^{[n]}(u) = \sum_{i=0}^{n-1} b^{[n]}(u-i)P_i \quad \text{with all } i \in \mathbb{Z} \tag{2.9}$$

which is equal to equation 2.8 for order $n = 4$.

2.1.3.2. Subdivision Scheme for cubic B-splines

In the previous section the basis functions for construction of piecewise continuous uniform cubic B-splines have been derived, such that the obtained curve is C^2 continuous over arbitrary number of control points.

Now, an appropriate subdivision scheme will be presented, similar to the DeCasteljau algorithm for Bézier curves in section 2.1.2.2. However, in this case the subdivision scheme will be presented in general form for all B-spline bases and any number of control points. In the end it will be applied without loss of generality on the cubic B-spline curve and in following on the bivariate cubic B-spline patch.

As an interesting fact one should note that due to the recursive definition of the B-spline bases (equation 2.4), also the subdivision operation can be constructed recursively. The demanded operation will be referred as *subdivision mask* $s(u)$ (and associated *subdivision matrix* $\mathbf{S}$) which should possess the following property (see equation 2.2):

$$\mathbf{G}^{k+1} = \mathbf{S}\mathbf{G}^{k} \tag{2.10}$$

The superscript k denotes the refinement recursion (or subdivision level) of euclidean points $P_i \in \mathbf{G}$.

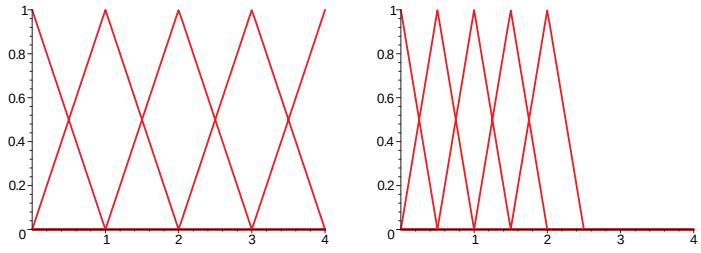


Figure 2.8.: *Translation and dilation. Left: B-Spline Basis $b^{[2]}(u-i)$ translated by i along the u axis., Right: the dilation $b^{[2]}(2u-i)$ for the same number of translations.*

Halving the Grid. Similar to the DeCasteljau algorithm, each uniform B-Spline segment can be always subdivided in the middle in two pieces over half of its interval.

This can be interpreted as halving the grid of the parametric axis u from integer values $\mathbb{Z}$ into $\frac{1}{2}\mathbb{Z}$. This result can be achieved by replacing the value of u by $2u$. Performing this transformation causes 'shrinking' of the basis function to the half of its width while the supported interval also halves. This is called *dilation* of the coordinate axis and forms the dilated scale function $b(2u)$ with knots at half-integer values $\frac{1}{2}\mathbb{Z}$. Figure 2.8 presents this dilation for a set of shifted scale functions $b(u-i)$ and $b(2u-i)$ respectively.

In fact, this is not yet the desired result, since dilating the basis function changes the shape of the curve itself, where it should be subdivided. To counteract the expansion, a linear combination of the dilated translated basis functions $b(2u-i)$ has to be found, which represents again exactly function $b(u)$.

Refinement Relation. The next goal is to find a linear combination of $b^{[n]}(2u-i)$, which will be referred as the *refinement relation*, that reconstructs $b(u)$ again. It has the form

$$b^{[n]}(u) = \sum_i s_i^{[n]} b^{[n]}(2u-i), \quad \text{for all } i \in \mathbb{Z}\,. \tag{2.11}$$

Obviously, for the first and second order cases, equation 2.11 is satisfied by:

$$b^{[1]}(u) = b^{[1]}(2u) + b^{[1]}(2u-1) \tag{2.12}$$

and

$$b^{[2]}(u) = \frac{1}{2}b^{[2]}(2u+1) + b^{[2]}(2u) + \frac{1}{2}b^{[2]}(2u-1)$$

respectively.

The associated subdivision mask is $s^{[1]} = \begin{bmatrix} 1 & 1 \end{bmatrix}$ for first order case. For the second order case, the mask is obviously $s^{[2]} = \begin{bmatrix} \frac{1}{2} & 1 & \frac{1}{2} \end{bmatrix}$. Figure 2.9 depicts this relation.

Subdivision Mask Function. Since the scale functions $b^{[n]}$ have been generated recursively (see section 2.1.3.1) it is obvious that the subdivision mask must also be subject to a recurrence. But first, before deriving this recurrence, the subdivision mask should be considered as a function of $\vartheta \in \mathbb{Q}$, where ϑ has to be distinguished from the continuous parametric domain variable u. Defining $s(\vartheta)$ as:

$$s(\vartheta) = \sum_i s_i \vartheta^i \quad \text{for all } i \in \mathbb{Z}$$

the subdivision masks $s^{[1]}$ and $s^{[2]}$ can be expressed as functions in following way:

$$s^{[1]}(\vartheta) = 1 + \vartheta$$

and

$$s^{[2]}(\vartheta) = \frac{1}{2} + \vartheta + \frac{1}{2}\vartheta^2$$

respectively. Note, that for $i \in \mathbb{Z}$, ϑ can have negative exponents and it is not necessarily a polynomial. The function $s(\vartheta)$ has been introduced to serve as a algebraic expression for all masks of any order, but does not have any special geometric interpretation. It is also often referred as the masks' *generating function*.

Now, having defined the subdivision mask function $s(\vartheta)$, for all B-spline bases $b^{[n]}$ can be stated:

***Theorem* 1:** For all $n \geq 1$, the subdivision mask $s^{[n]}(\vartheta)$ for the uniform B-spline basis function $b^{[n]}(u)$ of order n satisfies the recurrence

$$s^{[n]}(\vartheta) = \frac{(1+\vartheta)^n}{2^{n-1}} \tag{2.13}$$

To obtain this formula, the generation scheme for all $b^{[n]}$ has to be recalled, as stated in equation 2.5 in section 2.1.3.1, where the basis functions has been defined as a recurring convolution of each following order with the very first order function $b^{[1]}$, which is a unit rectangle. This can be formulated as a series of convolutions of the first order basis:

$$b^{[n]}(u) = \left(b^{[1]} \otimes b^{[n-1]}\right)(u) = \otimes_{i=1}^{n} b^{[1]}(u) \; .$$

Now, $b^{[1]}(u)$ can be substituted by the linear combination of its dilated translated functions from equation 2.12:

$$b^{[n]}(u) = \otimes_{i=1}^{n} b^{[1]}(u) = \otimes_{i=1}^{n} \left(b^{[1]}(2u) + b^{[1]}(2u-1)\right) \; . \tag{2.14}$$

Because of the linearity, time shift and time scaling invariance of the convolution operation [Wor07] one can state following for f,g and h with $m = f(t) \otimes g(t)$:

$$\begin{aligned} f \otimes (g+h) &= f \otimes g + f \otimes h \\ f(t-i) \otimes g(t-k) &= m(t-i-k) \\ f(2t) \otimes g(2t) &= \frac{1}{2} m(2t) \; . \end{aligned}$$

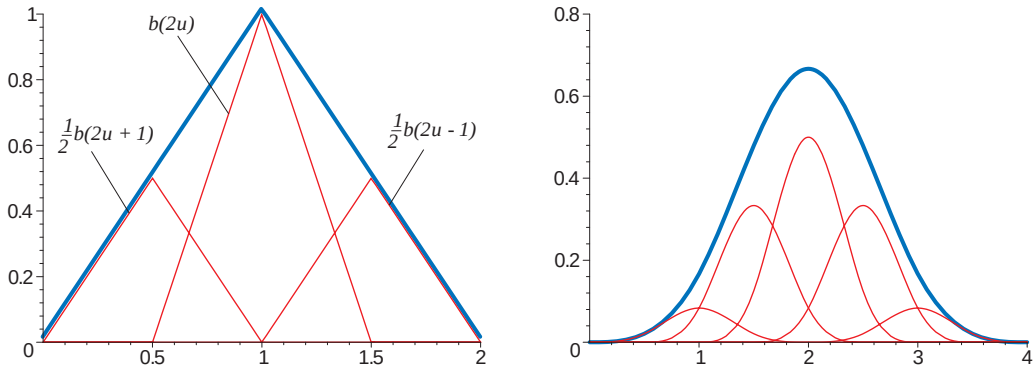


Figure 2.9.: *Refinement relation. The red (thin) curves are the translated dilated linear components multiplied by the subdivision mask s. The blue (thick) curves are their sum. Left: the basis function* $b^{[2]}$. *Right: basis function* $b^{[4]}$.

According to these axioms, the equation 2.14 can be reformed using the binomial theorem into a sum of functions $b^{[1]}(2u)$ and $b^{[1]}(2u-1)$:

$$b^{[n]} = \frac{1}{2^{n-1}} \sum_{i=0}^{n} \binom{n}{i} b^{[n]}(2u-i) \; .$$

This equation can be expanded and the mask in equation 2.13 can be formed. For the inductive integral proof of theorem 1 refer to Warren and Weimer [WW01] page 34, theorem 2.1.

Subdivision Matrix. Having the subdivision mask, for each order n of uniform B-spline the proper subdivision matrix $\mathbf{S}$ can by easily calculated. For the case of cubic B-splines, the mask is:

$$s^{[4]}(\vartheta) = \frac{1+4\vartheta+6\vartheta^2+4\vartheta^3+\vartheta^4}{8}$$

and the associated matrix $\mathbf{S}$:

$$\mathbf{S} = \frac{1}{8}\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 4 & 4 & 0 & 0 & 0 & \cdot \\ \cdot & 1 & 6 & 1 & 0 & 0 & \cdot \\ \cdot & 0 & 4 & 4 & 0 & 0 & \cdot \\ \cdot & 0 & 1 & 6 & 1 & 0 & \cdot \\ \cdot & 0 & 0 & 4 & 4 & 0 & \cdot \\ \cdot & 0 & 0 & 1 & 6 & 1 & \cdot \\ \cdot & 0 & 0 & 0 & 4 & 4 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} . \tag{2.15}$$

Note that the matrix is a bi-infinite matrix and the columns are two-shifts of the mask s. This two-shift results from the fact that by re-parametrization of u to $2u$ the shifting distance changes from i to $\frac{1}{2}i$. So to gain a knot, which was translated by 1 in the original function, on the new parametric grid resolution of $\frac{1}{2}\mathbb{Z}$ a distance of 2 half-integers has to be taken.

Lane-Riesenfeld Algorithm. Lane and Riesenfield have introduced a general method for computing the next level of points by upsampling the grid to obtain $\mathbf{G}^{k+1}$ followed by a sequence of midpoint averaging [LR80].

The equation 2.10

$$\mathbf{G}^{k+1} = \mathbf{S}\mathbf{G}^{k}$$

can be rewritten in as a sum as follows:

$$\mathbf{G}^{k+1} = P_i^{k+1} = \sum_j s_{i-2j}^{[m]} \; P_j^k \quad \forall i,j,k \in \mathbb{N} \,, \tag{2.16}$$

where P_i and P_j are the control points in vectors $\mathbf{G}$ at levels $k+1$ and k respectively and $s^{[m]}$ is the subdivision mask of order m. The shift of $2j$ in the subdivision mask results from the previously mentioned issue that the dilated integer gird $\frac{1}{2}\mathbb{Z}$ on level $k+1$ has to be matched with the grid $\mathbb{Z}$ on level k, which is achieved by doubling the steps at $k+1$. Now, the vectors P can also be seen as a generating function of ϑ of the form $P(\vartheta) = P_i\vartheta^i$ with ϑ^i as the coefficients variable. This would result in the coefficients vector of the form:

$$P(\vartheta) = P_0\vartheta^0 + P_1\vartheta^1 + P_2\vartheta^2 + \ldots + P_n\vartheta^n \,.$$

With this formulation the sum of pairwise products in equation 2.16 can be expressed as a product of the generating functions $P(\vartheta)$ and $s(\vartheta)$ by upsampling the coefficients

vector at level k. By upsampling it is meant to introduce 'dummy' points between the existing ones, which can be done by squaring ϑ such that

$$P(\vartheta^2) = P_0\vartheta^0 + P_1\vartheta^2 + P_2\vartheta^4 + \ldots + P_n\vartheta^{2n}$$

and rewriting the equation 2.16 as

$$P^{k+1}(\vartheta) = s^{[m]}(\vartheta)P^k(\vartheta^2) \quad .$$

Next geometric points can be computed by rolling out the equation above. This method is known as Lane-Riesenfeld Theorem and has been introduced in the year 1980. This is also a generalization the the Chaikins corner cutting algorithm for quadratic bases [Cha74], and moreover, it has been recently proved in a new, 'blossoming' way by Vouga and Goldman in [VG07]. For formal proofs refer to [LR80, VG07, WW01].

Invariant Neighborhood. Finally, the support for the subdivision should be mentioned. A B-spline curve might be defined over an infinite number of control points, which is the reason of the bi-infinity of the matrix $\mathbf{S}$. On the other hand the curve has also a local definition, where it is differentiable and can be subdivided only over a particular interval. In the case of cubic B-spline, the invariant neighborhood is 5, and it can be obtained from the number of the non-zero entries in each row of the subdivision matrix, which is 2 for odd points and 3 for even points. This implies that extra control points to the left and right of the current interval has to be taken into account in order to obtain the desired curve-piece. For cubic B-spline at the origin in the interval $[-1..1]$ the invariant neighbors are depicted in Figure 2.10. As one can see, at each level one more control point left and right outside the interval $[-1..1]$ is needed to continue on the next subdivision level. Three initial points would be not enough.

This fact can be also abused for piecewise definition of subdivision surfaces over independent mesh pieces. This work takes advantage of this issue, which will be explained in detail in chapter 4.

2.1.4. Toward the Catmull-Clark Scheme

In the previous section the subdivision of univariate uniform cubic B-splines has been presented. Now, a straightforward step will be the extension of the scheme to bivariate cubic B-spline patches. These patches are defined as the result of a tensor product of two B-spline curves

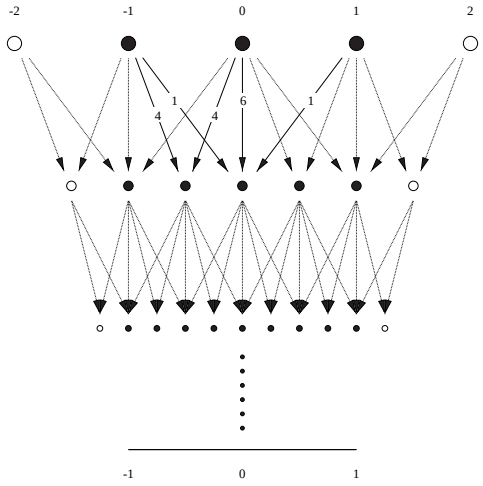


Figure 2.10.: *Invariant Neighborhood of a cubic B-Spline. It takes 5 control points at the coarsest level to determine the behavior of the subdivision limit curve over the two segments to the origin. Figure from [SZD+98].*

$$p(u,v) = p_1(u) \times p_2(v) = \sum_i \sum_j b_i(u) b_j(v) P_{i,j} \qquad \forall\, i,j \in \mathbb{N} \tag{2.17}$$

The parametric space is extended by the direction v, where $p_1(u)$, $p_2(v) \in \mathbb{P}$ are two linear independent tensors, such that they span a two-dimensional vector space $\mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P}^2$. This is also called tensor of second order and the resulting vector space is $\mathbb{P}^2$.

Matrix Notation. Similar as curves in previous sections, also patches can be expressed in the matrix representation. The second parametric variable v kept in the vector $\mathbf{V}^T = \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}$, such that

$$p(u,v) = \mathbf{U}^T \mathbf{M} \mathbf{G} \mathbf{M}^T \mathbf{V} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \mathbf{M} \mathbf{G} \mathbf{M}^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} .$$

Matrix $\mathbf{M}$ is the in section 2.1.3.1 derived coefficients matrix for cubic B-splines:

$$\mathbf{M} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

Matrix $\mathbf{G}$ are the geometric points (control points). The arrangement of the points on the patch is depicted in Figure 2.11 for $i = 0, j = 0$.

$$\mathbf{G} = \begin{bmatrix} P_{i,j} & P_{i,j+1} & P_{i,j+2} & P_{i,j+3} \\ P_{i+1,j} & P_{i+1,j+1} & P_{i+1,j+2} & P_{i+1,j+3} \\ P_{i+2,j} & P_{i+2,j+1} & P_{i+2,j+2} & P_{i+2,j+3} \\ P_{i+3,j} & P_{i+3,j+1} & P_{i+3,j+2} & P_{i+3,j+3} \end{bmatrix} \tag{2.18}$$

2.1.4.1. Topological Issues

Before subdividing the patches, some topological issues have to be discussed. In the previous section curves have been subdivided, where the topology is rather simple, since each two following control points – say *vertices* – are connected to a segment. Now the situation changes and the connectivity between vertices become a regular, rectangular grid, where each vertex is connected to four segments – say *edges* – and four in a circle connected vertices build a *face*. On the borders of the domain, exceptions to this regularity arise once again. To keep the bicubic B-spline patch in terms of its topological connectivity, it will be defined as a connected graph, which will be called *mesh* M.

Definition 1: Let a mesh M be a set of vertices V, edges E and faces F

$$\mathsf{M} = \{\mathsf{V}, \mathsf{E}, \mathsf{F}\},$$

where V is a set of $N \in \mathbb{N}$ vertices v, such that each vertex is defined as a vector $\mathbf{v}$ in $\mathbb{R}^3$ with its euclidean coordinates $\mathbf{v} = [x \quad y \quad z]^T$. Furthermore, the following is satisfied:

1. An edge $\mathsf{e} \in \mathsf{E}$ is a connection between two vertices v_i and v_j each $\in \mathsf{V}$, $i, j \in [0..N]$.
2. A cyclic path connecting a sequence of at least three vertices $\mathsf{v} \in \mathsf{V}$ forms a face $\mathsf{f} \in \mathsf{F}$:

$$\mathsf{f} = \{\mathsf{v}_i, \mathsf{v}_j, \mathsf{v}_k, ..., \mathsf{v}_l\} \quad \forall\, i, j, k, l \in [0..N].$$

The number of edges e_k connected to one vertex v is called valence val_{v}. The number of vertices forming a face f is called val_{f}.

If valence of all vertices in a mesh satisfies $val_{\mathsf{v}} = 4$, the mesh is called a regular mesh M_{Reg}.

In fact, definition 1 is quite loose, but sufficient enough to define the topological refinement of a bicubic B-spline patch. Assigning the control points in the matrix **G** as vertices, one can define a mesh M_{BS}, where the subscript *BS* denotes B-spline. Furthermore, two more restrictions have to be made to deal with a finite number of control points, like the 4 by 4 sized patch:

Definition 2: If the mesh M_{Reg} is defined by a finite number of vertices, lets define two more types of vertices:

1. If vertex v has valence $val_{\mathsf{v}} = 3$, call it border vertex v_{Bor}.
2. If vertex v has valence $val_{\mathsf{v}} = 2$, call it corner vertex v_{Cor}.

All other vertices are called v_{Reg} or simply v.

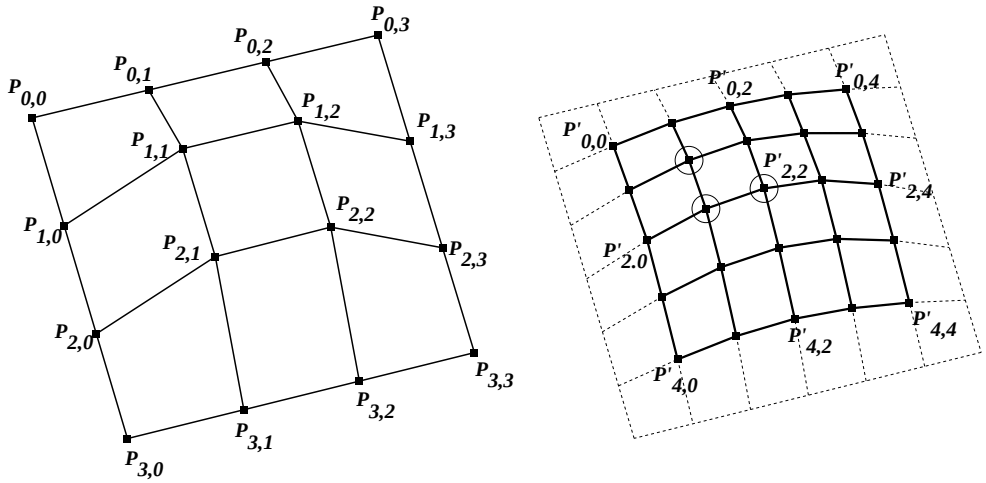


Figure 2.11.: *Subdivision of a 4 by 4 B-spline patch. Left: the original patch at level k. Right: emphasized region mark the new patch at level $k+1$. The three points $P_{2,2}^{k+1}$, $P_{2,1}^{k+1}$ and $P_{1,1}^{k+1}$ superimposed by white circles are treated in section 2.19. For better readability, new inserted points P^{k+1} have been substituted by P'.*

Manifoldness. The definitions given above do not express any rules for the connectivity of the mesh. In computer graphics the usual way to specify the connectivity rules for polygonal surfaces is *manifoldness*. A *manifold* is a surface on which at any point the local neighborhood has a topology equivalent to a disc. This assumption can be extended by introducing a *manifold with boundary* which is a surface with all points having the topological equivalence to either a disc or a half-disc [Mai02].

According to this stated specification, each edge $\mathsf{e}_i \in \mathsf{E}$ in a mesh M has to be a junction between exactly two faces $\mathsf{f} \in \mathsf{F}$ except at boundary of the total domain M, where the edges must have exactly one incident face f. Further on, each vertex $\mathsf{v}_i \in \mathsf{V}$ has to be surrounded by a full closed loop of faces f – again except at a boundary, where the faces must build a fan.

Topological Subdivision. Now the (topological) subdivision of $p(u,v)$ can be stated in the terms of the mesh M_{BS}. Recalling the idea of subdivision from previews sections and assuming control points as vertices and segments as edges, each edge e_i can be split into two at its midpoint by inserting a new vertex v_{e_i}. Each face f can be broken down into four pieces by introducing a new vertex v_f at its centroid, say the average of its four vertices $\mathsf{v}_f = \frac{1}{4}(\mathsf{v}_0 + \mathsf{v}_1 + \mathsf{v}_2 + \mathsf{v}_3)$ and connecting this new vertex v_f with the surrounding new edge vertices v_{e_i}. Indeed, the geometric position of v_f does not have any impact yet, since it will be obtained later during the subdivision matrix appliance. Finally, the old vertices will keep their neighborhood. The entire operation will be called *topological refinment* or *topological subdivision* of a mesh $\mathcal{TS}(\mathsf{M})$. This procedure is pictured in figure 2.12.

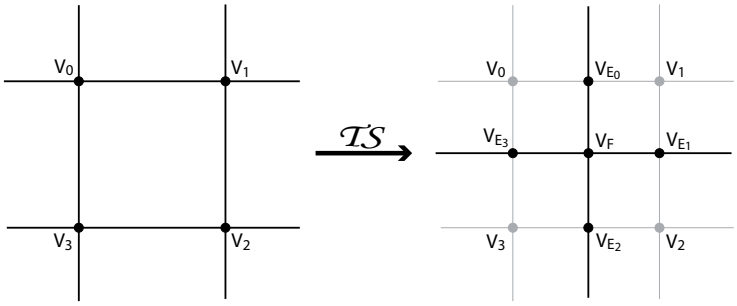


Figure 2.12.: *Topological subdivision of one face* F_i *of mesh* M_{BS}.

2.1.4.2. Applying Subdivision Matrix.

In the last section the parametric function $p(u,v)$ was defined as a mesh of control points. It gives the advantage that the control points in matrix $\mathbf{G}$ can be classified according to their neighborhood. Especially in context of bivariate subdivision this is useful to specify the matrix $\mathbf{G}^{k+1}$ at next subdivision level.

Recall equation 2.18, where the control points are arranged and the idea of halving the coordinate grid from section 2.1.3.2. Similar as in the univariate case, the parametric grid of $p(u,v)$ has to be halved, and the basis functions will have to be defined as linear combination of their translated dilated functions $b(2u-i)$. To achieve this, similar as in

the approach of Lane-Riesenfeld [LR80] (see section 2.1.3.2), the grid is filled up with 'empty' points at each second position. Rewriting it as a matrix $\mathbf{G}^{k+1}$ for four original points yields:

$$\mathbf{G}^{k+1} = \begin{bmatrix} P_{i,j} & P_{i,j+1} & P_{i,j+2} & P_{i,j+3} & P_{i,j+4} \\ P_{i+1,j} & P_{i+1,j+1} & P_{i+1,j+2} & P_{i+1,j+3} & P_{i+1,j+4} \\ P_{i+2,j} & P_{i+2,j+1} & P_{i+2,j+2} & P_{i+2,j+3} & P_{i+2,j+4} \\ P_{i+3,j} & P_{i+3,j+1} & P_{i+3,j+2} & P_{i+3,j+3} & P_{i+3,j+4} \\ P_{i+4,j} & P_{i+4,j+1} & P_{i+3,j+2} & P_{i+4,j+3} & P_{i+4,j+4} \end{bmatrix}$$

Note, that all points $P \in \mathbf{G}^{k+1}$ are in fact P^{k+1} and were replaced by P for better readability.

Since the two parametric dimensions u and v of the patch are linear independent, each direction can be proceeded partially in terms of the tensor product. Also the already in section 2.1.3.2 derived subdivision matrix $\mathbf{S}$ (equation 2.15) can be reused for each dimension to obtain positions for the control points.

It has to be mentioned, that since the subdivision matrix $\mathbf{S}$ is bi-infinite it is designed to perform on infinite manifold surfaces. While this is impossible in practice, in this particular example only the interior control points of a finite patch $\bar{p}(u,v)$ will be affected by the subdivision. For this example, the particular patch $\bar{p}(u,v)$ is defined by 4 by 4 control points, such that the border points will be not concerned. Without loss of the generality, the subdivision matrix will be applied only on a part of the patch. This is sufficient to examine the different types of control points. Figure 2.11 depicts this on the left-hand side by the emphasized lines.

The particular points $\mathbf{G}^{k+1}$ can be obtained by the Kronecker product of the matrices:

$$\mathbf{G}^{k+1} = \mathbf{S}\mathbf{G}^{k}\mathbf{S}^{T} \tag{2.19}$$

which is:

$$\mathbf{G}^{k+1} = \frac{1}{8}\begin{bmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \\ 0 & 0 & 4 & 4 \end{bmatrix} \begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} & P_{0,3} \\ P_{1,0} & P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,0} & P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,0} & P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix} \frac{1}{8} \begin{bmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \\ 0 & 0 & 4 & 4 \end{bmatrix}^{T}$$

By multiplying equation 2.19 one obtains three different basic formulas for three different types of points in the mesh. The three example points $P_{2,2}^{k+1}$, $P_{2,1}^{k+1}$ and $P_{1,1}^{k+1}$ calculated below are shown in Figure 2.11, right hand side, superimposed by white circles.

- *Face-points*, which are geometrically the centroid of each polygon:

$$P_{2,2}^{k+1} = \frac{P_{1,1} + P_{2,1} + P_{1,2} + P_{2,2}}{4}$$

- For the *edge-points*, which are an average of its vertices and the newly introduced face points:

$$P_{2,1}^{k+1} = \frac{P_{1,0} + P_{2,0} + 6\,(P_{1,1} + P_{2,1}) + P_{1,2} + P_{2,2}}{16}$$

- Finally, the *vertex-points*, which can be obtained by proper weighting of the current vertex, neighboring edge-points and newly introduced face-points:

$$P_{1,1}^{k+1} = \frac{P_{0,0} + 6P_{1,0} + P_{2,0} + 6\,(P_{0,1} + P_{1,1} + P_{2,1}) + P_{0,2} + 6P_{1,2} + P_{2,2}}{64}$$

This presented bivariate subdivision is actually the Catmull-Clark scheme, however only applicable on exactly regular and infinite meshes. In practice this would be insufficient, since only a rather limited class of objects can be modeled by such surfaces. Additionally it would imply a significant overhead for the modeling process and constrain the designer to keep in mind those restrictions. To overcome these limitations some extensions to the scheme have to be taken into account.

2.1.5. General Catmull-Clark Rules

Catmull and Clark have shown in their original work [CC78] that the derived scheme is not only restricted to domains with quadrangle input faces. Indeed, their scheme is able to deal with an input with arbitrary polygons, although it is still a generalization of the bivariate cubic B-spline surfaces.

The key ingredient to this extension is the introduction of handling of boundaries and irregular vertices in polygonal surface domains. This will be described in next section.

2.1.5.1. Extraordinary Vertices

In literature it has become commonplace to denote irregular points as *extraordinary vertices*. There are no limitations to the valences of such vertices which means that the general Catmull-Clark rules accept all polygonal input domains. As mentioned in section 2.1.1.5 and 2.1.4.1 only convex faces will ensure the manifoldness of the resulting surface, albeit – some degenerations might be indeed intended during the modeling process. The algorithm is not sensitive for those cases and proceeds as usual, thus it will compute degenerated centroids for concave faces.

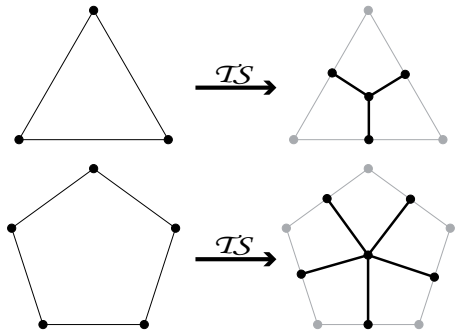


Figure 2.13.: *Topological subdivision of arbitrary faces in the Catmull-Clark scheme. Top: triangle, Bottom: pentagon. As a result, only quadrangles are introduced into new mesh. Note, that in case different to a quadrangle a new extraordinary vertex is inserted too.*

After the very first step of the subdivision, the mesh is split into quadrangles only. Also only in this step, new extraordinary vertices are introduced at faces different to quadrilaterals as demonstrated in Figure 2.13. In all following recursions, the algorithm proceeds exactly in the same manner by splitting each quad into four new pieces and no irregularities are brought in any longer. This means that the number of extraordinaries is rather limited and can be easy calculated out of the initial domain. In subsequent steps the face space complexity tends to $(2^k) \times (2^k)$ faces and the vertex space complexity of one quadrangle to $(2^k+1) \times (2^k+1)$ for k recursions.

The interesting issue at irregular vertices is the question of the convergence and the continuity. Catmull and Clark proposed a scheme to compute proper weights around vertices as shown in Figure 2.14. In this scheme the weights can be balanced by the coefficients β and γ, whereby the proposed ones are $\beta = \frac{3}{2k}$ and $\gamma = \frac{1}{4k}$, where k denotes the valence. This is for the case of valence equal to four exactly the rule derived in previous section, for all other cases the surface do not match a regular B-spline anymore but maintains at least C^1 continuity. This situation is more sophisticated and has been proven by *eigen analysis* of the subdivision matrix by Ulrich Reif [Rei95] and will be the subject of discussion in section 2.1.6.4.

2.1.5.2. General Formulas

According to equation 2.19 and in consideration of the extraordinary vertex properties as discussed in previous section, one can indeed state very simple general formulas, which always provide a proper position for a vertex with arbitrary valence. The weights for particular vetrices are shown in Figure 2.15 and can be stated in the following three formulas for the face- edge- and vertex points respectively, where k denotes the sub-

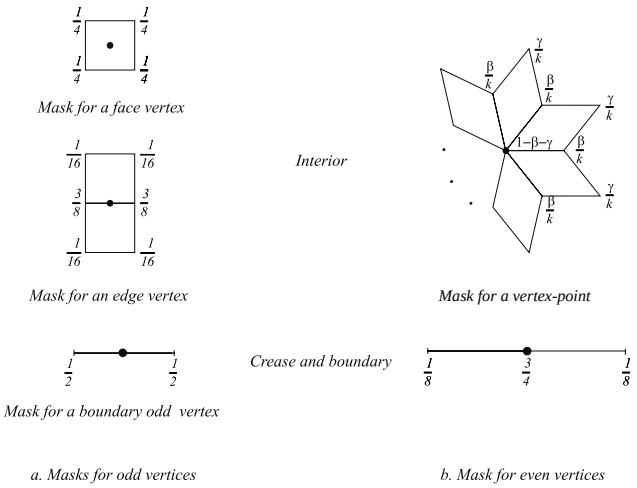


Figure 2.14.: *Catmull-Clark subdivision masks. Top: Catmull and Clark [CC78] suggest the following coefficients for rules at regular- and extraordinary vertices:* $\beta = \frac{3}{2k}$ *and* $\gamma = \frac{1}{4k}$. *Bottom: rules on borders of the mesh, which follow the quadric B-spine as in Chaikins algorithm [Cha74]. Image from [SZD+98].*

division recursion, m the face vertex count, n the vertex-valence and j the indicies of particular elements:

$$\mathbf{f}_j^{k+1} = \frac{1}{m}\sum_i^m \mathbf{v}_i^k \tag{2.20}$$

$$\mathbf{e}_j^{k+1} = \frac{1}{4}\left(\mathbf{v}^k + \mathbf{e}_j^k + \mathbf{e}_{j-1}^k + \mathbf{f}_j^{k+1}\right) \tag{2.21}$$

$$\mathbf{v}^{k+1} = \frac{n-2}{n}\mathbf{v}^k + \frac{1}{n^2}\sum_j^n \mathbf{e}_j^k + \frac{1}{n^2}\sum_j^n \mathbf{f}_j^{k+1} \tag{2.22}$$

2.1.5.3. Borders and Creases.

Up to now the subdivision process has been discussed for the case of an infinite manifold mesh. In practice all meshes located in finite computer memory have to be somehow constrained. This issue has been a subject for several researchers and in the nineties DeRose [DKT98] followed by Biermann [BLZ00] have introduced rules to deal with boundaries of manifold surfaces.

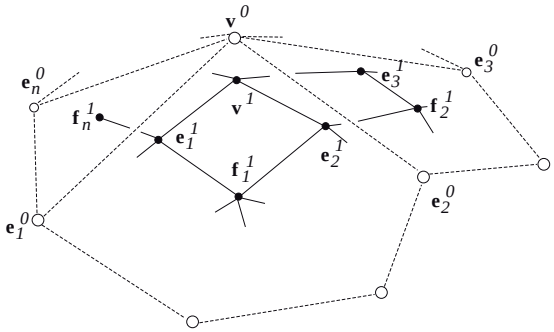


Figure 2.15.: *Catmull-Clark formulas for vetices of arbitrary valence* $val_{\mathbf{v}}$. *Courtesy Tony DoRose et al. [DKT98]*

Infinitely sharp creases. As mentioned, a local neighborhood of a vertex in a closed manifold mesh can be topologically equivalent to either a disc or a half-disc. DeRose *et al.* [DKT98] introduced in their work a definition for infinitely and semi-sharp creases. While the first ones are also very well suited to handle domain boundaries, the second ones are useful to define semi-sharp kinks on user-chosen edges.

First of all the constellations of possible vertex-edge combinations can be classified for a vertex as follows:

- *crease vertex:* a vertex with exactly two connected crease edges
- *corner vertex:* a vertex with three or more connected crease edges
- *dart vertex:* a vertex with exactly one crease edge

whereby an edge can be tagged either as a crease or not. For these specified cases different subdivision rules can be defined in order to obtain sharp edges. DeRose *et al.* [DKT98] propose to place crease edge-points exactly on the bisectors of the original edges:

$$\mathbf{e}_j^{k+1} = \frac{\mathbf{v}^k + \mathbf{e}_j^k}{2} \tag{2.23}$$

and the crease vertices using the following distribution:

$$\mathbf{v}^{k+1} = \frac{\mathbf{e}_j^k + 6\mathbf{v}^k + \mathbf{e}_{j+1}^k}{8}. \tag{2.24}$$

In the case of corner vertices the particular vertex is left at its original position:

$$\mathbf{v}^{k+1} = \mathbf{v}^k \tag{2.25}$$

and finally the dart-vertices are handled exactly the same as the usual ones by equation 2.22.

These masks are also depicted in Figure 2.14, at the bottom. Note that if these rules are applied on interior edges and vertices, a sharp crease in the mesh will be created.

Semi sharp creases. While the described sharp creases fit to borders of a mesh very well, kinks in the interior of surfaces are very seldom infinitely sharp. For the semi-sharpness DeRose *et al.* present a quite elegant solution. According to the creases classification from above, to each crease edge a floating sharpness value $\mathbf{e}.s$ is assigned. For the vertices a value $\mathbf{v}.s$ is computed as the average of the $\mathbf{e}.s$ from incident edges respectively. During the subdivision the rules for placing new vertices are determined as follows (courtesy from [DKT98]):

- An edge-point corresponding to a smooth edge ($\mathbf{e}.s = 0$) is computed by the rule in equation 2.21
- An edge-point corresponding to an edge of sharpness $\mathbf{e}.s \geq 1$ is computed using the sharp edge rule in equation 2.23
- An edge-point corresponding to an edge of sharpness $0 < \mathbf{e}.s < 1$ is computed using a blend between smooth and sharp edge rules. Specifically, let $\mathbf{v}_{smooth}$ and $\mathbf{v}_{sharp}$ be the edge-points computed by smooth and sharp rules as described above, respectively. The edge point is placed at

$$(1 - \mathbf{e}.s)\mathbf{v}_{smooth} + \mathbf{e}.s\mathbf{v}_{sharp}.$$

- A vertex-point corresponding to a vertex adjacent to non or one sharp edge is computed normally by equation 2.22.
- A vertex-point corresponding to a vertex adjacent to three or more sharp edges is computed using the corner rule from equation 2.24.
- A vertex-point corresponding to a vertex $\mathbf{v}$ adjacent to two sharp edges is computed using the crease vertex rule from equation 2.24 if $\mathbf{v}.s \geq 1$, or a linear blend between the crease vertex and corner rule (equation 2.24) if $\mathbf{v}.s < 1$, where $\mathbf{v}.s$ is the average of the incidence edge sharpnesses.

After subdivision of an edge, the sharpness of the resulting subedges is determined in following way: Let $\mathbf{e}_a, \mathbf{e}_b, \mathbf{e}_c$ denote three adjacent edges of a crease as pictured in Figure 2.16, than the subedges $\mathbf{e}_{ab}$ and $\mathbf{e}_{bc}$ are defined by

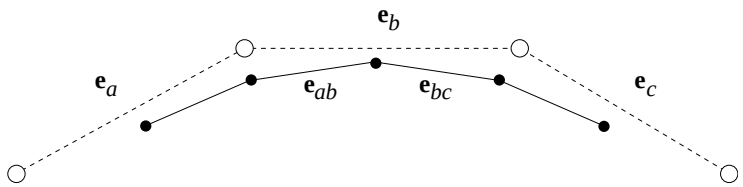


Figure 2.16.: *Scheme for labeling sub-edges. Figure from [DKT98].*

$$\begin{aligned} \mathbf{e}_{ab}.s &= max\left(\frac{\mathbf{e}_a.s+3\mathbf{e}_b.s}{4}-1,0\right) \\ \mathbf{e}_{bc}.s &= max\left(\frac{3\mathbf{e}_b.s+\mathbf{e}_c.s}{4}-1,0\right) \end{aligned}$$

This solution allows to model user determined degrees of smoothness on crease edges. Note that for infinitely high values for $\mathbf{e}.s$ and $\mathbf{v}.s$ the edges become totally sharp as described in the previous paragraph. On the other hand, after each substep a 1 is subtracted in order to weaken the sharpness consecutively and the maximum with 0 is taken to avoid negative values. Furthermore, this blending is derived from the Chaikins curve subdivision algorithm [Cha74] and its subdivision mask is $\left[\frac{1}{2} \quad \frac{3}{4} \quad \frac{3}{4} \quad \frac{1}{2}\right]$ which is derived from the quadratic B-spline basis (refer to section 2.1.3.2).

2.1.6. Parametrization and Analysis of Catmull-Clark Subdivision Surfaces

Until now, subdivision surfaces have been presented in the form of a limit surface of a sequence of refinement steps performed on a polyhedral net. It has been shown in section 2.1.4 that the limit surface is derived from the bi-cubic B-spline surface (Box-spline) and thus it should also inherit its properties of smoothness. This section will deal with the parametrization and the analysis of the limit surface itself. For this reason, first, a general definition of parametrization of regular surfaces will be given and further on, in terms of the derived context, the subdivision surfaces will be inspected. Later on in this section, the eigen analysis approach for the subdivision matrix will be introduced in order to show the convergence at irregular cases.

2.1.6.1. Parametrization of regular Surfaces

Parametrization of surfaces is an important issue in computer graphics. It enables one to express a surface $S \in \mathbb{R}^3$ globally or locally as a 2D plane $U \in \mathbb{R}^2$. This makes it

possible to apply some of the mathematical tools much more conveniently on U, and, furthermore to match 2D structures onto S (i.e. texture mapping).

A *regular surface* $S \in \mathbb{R}^3$ will be defined as a surface, which is at least locally isomorph to a disc, does not exhibit any self-intersections and possesses a unique normal in this local region. Thus, it can be seen as a *manifold*, as defined in section 2.1.4.1. By definition [Car93], a *parametrization* of a surface S at some point $\mathbf{q} \in S$ is a mapping X as follows:

Definition 3: A subset $S \subset \mathbb{R}^3$ is a regular surface, if for each $\mathbf{q} \in S$ there exists an environment $V \in \mathbb{R}^3$ and a mapping $X : U \to V \cap S$ of an open set $U \subset \mathbb{R}^2$ onto $V \cap S$, such that following holds

$$X : U \subset \mathbb{R}^2 \to S \subset \mathbb{R}^3 \tag{2.26}$$

and

1. X is differentiable, such that for

$$X(u,v) = (x(u,v), y(u,v), z(u,v)), \qquad \text{where } u,v \in U,$$

   each $x(u,v), y(u,v), z(u,v)$ has continuous partial derivatives of any order in U.
2. X is homeomorph, such that it always has a unique and continuous inverse mapping $X^{-1} : S \subset \mathbb{R}^3 \to U \subset \mathbb{R}^2$
3. For each $\mathbf{q} \in U$ the differential $\frac{d\mathbf{X}}{d\mathbf{q}} : \mathbb{R}^2 \to \mathbb{R}^3$ is injective.

Figure 2.17 illustrates this definition. Note that this is not restricted to a portion of the surface S, but is also true for the entire surface, as long as the topological equivalence of S to a disc is given. Otherwise, any manifold surface (refer to section 2.1.4.1) can be parametrized at least locally. The second item of definition 3 defines the bijection of the map, which ensures that the surface does not exhibit self-intersections. On the left-hand side of Figure 2.17, the vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ define a canonical basis with the origin at $\mathbf{q}(u_0, v_0)$. On the right-hand side, the vectors $\mathbf{x}_u$ and $\mathbf{x}_v$ denote the partial derivatives of the surface with respect to the coordinates curves C_u and C_v and are referred as *tangent vectors*. Please note that $\mathbf{x}_u, \mathbf{x}_v$ is a short notation for

$$\mathbf{x}_u = \frac{\partial \mathbf{x}}{\partial u} = \left(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u} \right) = \frac{dX}{d\mathbf{q}}(\mathbf{e}_1) \tag{2.27}$$

$$\mathbf{x}_v = \frac{\partial \mathbf{x}}{\partial v} = \left(\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v} \right) = \frac{dX}{d\mathbf{q}}(\mathbf{e}_2) \tag{2.28}$$

Moreover, the differential

$$\mathbf{J}_q = \frac{dX}{d\mathbf{q}} = dX_q = \begin{bmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial u} & \frac{\partial z}{\partial v} \end{bmatrix} \tag{2.29}$$

has to build a Jacobian matrix $\mathbf{J}_q$ of a full rank, such that at least one of its subdeterminants is such that $\det_J \neq 0$, which ensures that the tangents are linearly independent and actually span a plane (referred as *tangent plane* $T_q(X)$ at $\mathbf{q}$).

Parametrized Surface. In this instance and the entire discussion that follows, the tangent vectors will be assumed as orthonormal. In such a case the vectors form a 2D canonical basis on T_q and all points on the plane can be expressed by their means, as i.e. a curve $C : s \to X(u(s), v(s))$ parametrized by s on a surface $X : U \to S$, if passing through $\mathbf{q} = P(u_0, v_0)$ it can be expressed on the tangent plane as a linear combination of $\{\mathbf{x}_u, \mathbf{x}_v\}$:

$$\frac{dC}{ds}(s) = \mathbf{x}_u(s)\frac{du}{ds}(s) + \mathbf{x}_v(s)\frac{dv}{ds}(s) \quad \Leftrightarrow \quad \mathbf{x}_u u' + \mathbf{x}_v v' = \mathbf{c}' \,. \tag{2.30}$$

In other words, this equation shows that the tangent vector $\mathbf{c}'$ of any curve C passing through $P(u_0, v_0)$ can be always expressed in terms of the tangent plane basis $\{\mathbf{x}_u, \mathbf{x}_v\}$ defined by the parametrization $X : U \to S$.

Unit Normal and Orientation. The cross product of the 2D basis defines the *unit surface normal*, which can be calculated and normalized by

$$\mathbf{n} = \frac{\frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v}}{\left\| \frac{\partial \mathbf{x}}{\partial u} \times \frac{\partial \mathbf{x}}{\partial v} \right\|} \tag{2.31}$$

and builds together with $\{\mathbf{x}_u, \mathbf{x}_v\}$ a 3D canonical basis with the origin at $\mathbf{q}$. This basis will be referred as *local tangent frame* and gives a base for several interesting inspections and applications to the surface.

According to this basis the *orientation* of a surface can be defined in terms of the unit normal vector field of the form $N : U \to \mathbb{R}^3$ which assigns to each point $\mathbf{q} \in U$ a unit normal vector $N(q) = \mathbf{n}_q$. If $N(q)$ can be defined continuously over the entire surface, than the surface is referred as orientable. This is equivalent to that N is differentiable at any point $\mathbf{q} \in U$ and means that all normals point to a uniquely determinable side of a surface. Note that not all regular surfaces have an unambiguous global orientation (i.e. Möbius-Strip does not have one), but for all manifold surfaces the orientation can be defined at least locally.

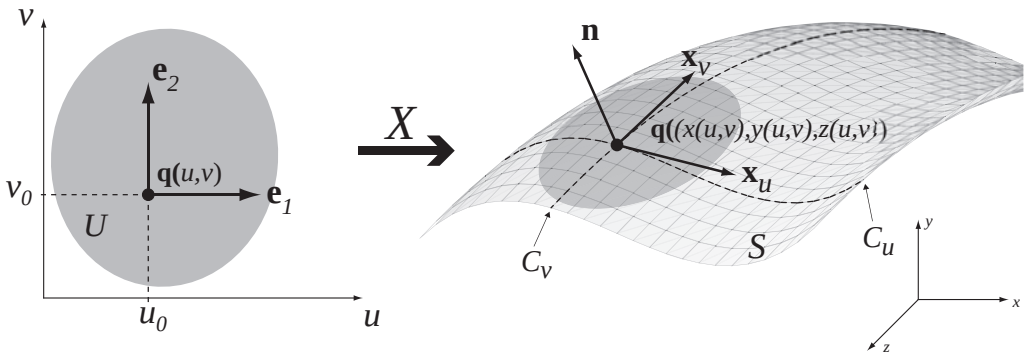


Figure 2.17.: *Local parametrization of a surface S by the mapping $X : U \to S$. Left: vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ symbolize a local canonic basis with the origin at $\boldsymbol{q}(u_0, u_v)$. Right: the vectors $\boldsymbol{x}_u, \boldsymbol{x}_v$ symbolize orthonormal tangents at the point $\boldsymbol{q}$. Vector $\boldsymbol{n}$ is the unit normal at u_0, v_0).*

In terms of the tangent space, several interesting methods in computer graphics have been introduced. Especially the technique of bump mapping [Bli78] as well as displacement mapping which rely on these coordinate frames. Displacement mapping will be the subject of discussion in section 2.2.

2.1.6.2. Natural Parametrization

The parametrization of surfaces as shown in previous section allows the mapping of a local region around any point $\mathbf{q}$ to a plane. To ensure the bijection of the mapping $X : U \to S$ it has been defined that the surface S is not allowed to exhibit any self-intersections in the vicinity of $\mathbf{q}$. This issue has been already addressed several times in sections 2.1.1.5 and 2.1.4.1. While both bivariate B-splines as well as Catmull-Clark subdivision surfaces are not restricted to this constrains in general, for the purpose of the mathematical disquisition on their properties it will be assumed that the surfaces are manifold.

Of course, this assumption restricts the objects somehow and excludes topologically sophisticated shapes [3] – as i.e. the Klein Bottle – from this discussion. Nevertheless, without loss of generality this thesis will treat the surfaces as defined in the previous section and it should be noted that a general parametrization of degenerated cases can be done by mapping the polyhedrons into a higher dimensional space (i.e. 4D) [SZD+98]. Furthermore Ying and Zorin have investigated non-manifold cases in order to introduce an appropriate subdivision method [YZ01].

The limit surface of a Catmull-Clark subdivision process is naturally parametrized by its generation method. Assuming the initial domain for subdivision as a user defined control mesh $\mathsf{M}^{(0)}$ composed of arbitrary polygons, after the first step all faces are split into quads. Now, all regions away form irregular vertices are subdivided in a regular, rectangular pattern. Assuming these regions are lying on a 2D-plane and performing only the topological subdivision $\mathcal{TS}(\mathsf{M})$ (refer to section 2.1.4.1) results in a successive finer grid of points at each step. Finally, at the limit, it delivers a continuous, smooth parametrization. This can be observed in Figure 2.18, bottom, where the white circles mark exact regular regions.

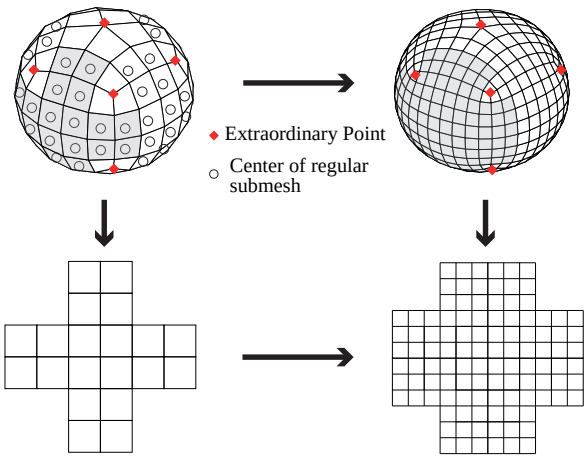


Figure 2.18.: *Natural Catmull-Clark parametrization of a mesh with arbitrary polygonal faces. The faces with white circles mark regular reagions, where the mesh is C^2 continuous. The gray emphasized regions are mapped to the 2D planes in the bottom . Figure courtesy lean on [SZD+98].*

[3] Note that the term *topology* has two meanings depending on the context: (1) the first one is mathematical and describes the connectivity of a shape: number of holes or passages through it (genus) etc. (2) the second meaning refers to the connectivity of a mesh composed of vertices, edges and faces etc.

2.1.6.3. Convergence and Continuity of the Catmull-Clark Surface

In the analysis an usual proof of the uniform convergence of a sequence of functions $p^k(u)$ to a limit function $p^\infty(u)$ is done by proving that for all $\varepsilon > 0$ there exists a sequence element $p^k(u)$ such that all following elements $k > n$ satisfy

$$\left| p^\infty(u) - p^k(u) \right| < \varepsilon \quad .$$

In other words, this defines a 'tube' of the radius of ε around the function p^k and demands that for each radius there exists some sequence element with index n from which on all following sequence elements do not leave the 'tube' anymore.

In the case of standard subdivision the process starts out with a mesh $\mathsf{M}^{(0)}$ composed of vertices, edges, and faces that serves as the base for a sequence of refined meshes $\mathsf{M}^{(0)}, \mathsf{M}^{(1)}, \mathsf{M}^{(2)}, ..., \mathsf{M}^{(k)}$ which converges to a limit surface for $k \to \infty$. A new submesh $\mathsf{M}^{(k+1)}$ of a specific mesh $\mathsf{M}^{(k)}$ is generated by using rules obtained from the subdivision matrix (refer to section 2.1.5 on page 28). New vertices are placed in such a way that the limit surface of the subdivision process satisfy certain continuity constraints, e.g. C^1 or C^2 continuity.

In the specific case of subdivision of uniform B-splines the limit surface is known *a priori* and by proving that some subdivision mask s is equivalent to the function itself is enough to show the convergence of the scheme (refer to section 2.1.3.2).

Going on this way, it can be also deduced that since a sequence p^k converges to a cubic function, the smoothness of the limit p^∞ must be continuous to the same order also.

With a further naive assumption it can be stated that since the bivariate sequence $p^k(u,v)$ is a linearly independent tensor product of two univariate cases, each dimension converges and is smooth for itself and because of the independence of the parametric axes, the spanned surface is so as well. According to this statement, the smoothness of the limit surface at regular regions is C^2 as the bi-cubic tensor product B-spline patch.

It should be mentioned that it is by far not so naive for the general subdivision approach and there have been mathematical tools developed to handle it. For those cases refer to Warren and Weimer [WW01], chapter 3 and 8.

All this is rather obvious for the case of Catmull-Clark scheme for the exactly regular regions. The more involved situation at extraordinary points has been an open question for quite a long time, until Ulrich Reif has presented a method for proving the C^1 continuity at those regions [Rei95] in the year 1995. In a later work of Peters and Reif [PR98] the convergence and continuity of Catmull-Clark surfaces has been proven. After that a number of works have appeared and subdivision has become quite popular in the computer graphics research scene.

2.1.6.4. Eigen Analysis of the Subdivision Matrix

The simple deduction of the convergence and continuity of Catmull-Clark subdivision presented above holds only for regular regions where the scheme approximates exactly a bivariate Box-spline. This section will show the mathematical procedure on how to prove the convergence and continuity of subdivision surfaces in the near of extraordinary points. It it the method of *eigen analysis* of the subdivision matrix. Furthermore, since regular vertices can be seen as a special case of the irregular ones, the tools presented here cover them as well. For more detailed introduction refer to [SZD+98] and for mathematically involved methodology to [WW01], chapter 8.

Eigenvalues and eigenvectors of a matrix. This analysis is based on the linear algebra theory of *eigenvectors* and *eigenvalues* of a square matrix. This section is not intended to provide the full mathematical context of linear algebra which can be obtained i.e. in Strang [Str88], but some rudimentary foundations behind the *eingen-analysis* of a matrix instead (also often referred as *spectral-analysis*).

An eigenvector $\mathbf{v}$ of a matrix $\mathbf{M}$ is a vector such that

$$\mathbf{M}\mathbf{z} = \lambda\mathbf{z}, \tag{2.32}$$

where λ is a scalar referred as eigenvalue corresponding to $\mathbf{z}$. The equation states the special eigenvalue problem of $\mathbf{M}$. Eigenvectors of a matrix $\mathbf{M}$ represent a subspace which allows to express properties of $\mathbf{M}$ on special vectors in terms of scalars. They are i.e. useful to solve differential equations, where the problem can be stated as a linear combination of the eigenvectors and their corresponding eigenvalues with respect to the linearly independent subspace.

The eigenvalues of $\mathbf{M}$ can be obtained by rewriting equation 2.32 into

$$(\mathbf{M} - \lambda\mathbf{I})\mathbf{z} = 0,$$

where $\mathbf{I}$ denotes the identity matrix. To fulfill this equation for a non-trivial case, where the eigenvalues are not 0, the term $\mathbf{M} - \lambda\mathbf{I}$ must be singular. In such a case the eigenvalues can be computed by forming the determinant $\det_M$, which must be (for a non-trivial solution) unequal to 0

$$\det_M(\mathbf{M} - \lambda\mathbf{I}) = 0.$$

This equation considers the *right eigenvectors* which are enough for the analysis performed here. It should be noted that also *left eigenvectors* can be computed as a row vector $\mathbf{z}^T$, such that $\mathbf{z}^T\mathbf{M} = \lambda\mathbf{z}$ is fulfilled instead of 2.32.

The determinant can be computed in various ways (i.e. Sarrus' rule), but finally one will obtain a polynomial of the degree of the rank n of the matrix which has the form

$$\rho(\lambda) = a_n\lambda^n + a_{n-1}\lambda^{n-1} + \ldots + a_1\lambda^1 + a_0.$$

This polynomial is called the *characteristic polynomial* $\rho(\lambda)$ of the matrix $\mathbf{M}$ and its roots deliver the eigenvalues λ_n. Note that not all eigenvalues of a matrix necessarily have to be real numbers.

Furthermore, matrices posses a distinct set of eigenvalues $\lambda_1 ... \lambda_n$ if and only if they have a full set of independent eigenvectors. The subspace spanned by the eigenvectors is referred as the *eigenspace* of $\mathbf{M}$ and such matrices are referred as *non-defective* .

Another useful property is the possibility to define a matrix Λ which contains the eigenvalues on its diagonal and a matrix $\mathbf{Z}$ which contains in its columns the respective eigenvectors. In this terms, following holds

$$\mathbf{MZ} = \mathbf{Z}\Lambda \quad \Leftrightarrow \quad \mathbf{Z}^{-1}\mathbf{MZ} = \Lambda \quad \Leftrightarrow \quad \mathbf{M} = \mathbf{Z}\Lambda\mathbf{Z}^{-1}.$$

The columns in $\mathbf{Z}$ are called the right eigenvectors $\mathbf{z}_i$ and the inverse matrix $\mathbf{Z}^{-1}$ contains the left eigenvectors $\tilde{\mathbf{z}}_i$ in its rows, which fulfill the initial condition $\mathbf{z}^T\mathbf{M} = \lambda\mathbf{z}$.

Moreover, for the powers of $\mathbf{M}^k$ the eigenvalues are $\lambda_1^k, ..., \lambda_n^k$ and the corresponding eigenvectors $\mathbf{z}_1, ..., \mathbf{z}_n$ remain the same as for $\mathbf{M}^1$ and finally:

$$\Lambda^k = \mathbf{Z}^{-1}\mathbf{M}^k\mathbf{Z}.$$

Spectral Analysis. The properties of a subdivision scheme can be examined by performing the eigen analysis of the subdivision matrix $\mathbf{S}$. Catmull-Clark subdivision, like most known schemes possesses a non-defective matrix and thus a linearly independent set of eigenvectors. Actually, the proof of convergence and smoothness must be fulfilled for each vertex valence separately, albeit by using always the same method. The source of this issue lies in the subdivision matrix which changes for each valence. Due to this reason, also the regular subdivision matrix, as derived in section 2.1.3.2 can be put to this examination, although its properties are already well known. Nevertheless, for the reason of simplicity in this thesis the regular subdivision matrix of Catmull-Clar meshes will be examined to demonstrate the methodology used in such analysis.

To perform the analysis of $\mathbf{S}$ first of all the matrix derived in section 2.1.3.2 (refer to equation 2.15 on page 17) has to be rewritten into a form, such that it covers the situation around a particular vertex $\mathbf{v}$ instead covering a bi-infinite patch. This can be done by arranging the neighboring vertices in a circular order around the point of interest such that the control points vector $\mathbf{G}_4$ of valence 4 becomes (see Figure 2.19):

$$\mathbf{G}_4 = \begin{bmatrix} \mathbf{v} & \mathbf{e}_i & \mathbf{e}_{i+1} & \mathbf{e}_{i+2} & \mathbf{e}_{i+3} & \mathbf{f}_i & \mathbf{f}_{i+1} & \mathbf{f}_{i+2} & \mathbf{f}_{i+3} \end{bmatrix}^T , \tag{2.33}$$

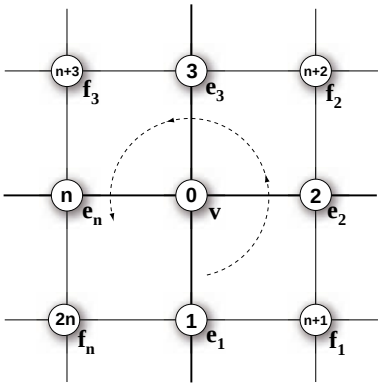


Figure 2.19.: *Indexing scheme for vertices around an extraordinary point. This scheme is independent of the valence of the vertex, since any number n of vertices can by arranged in a circle around the central vertex* ***v****.*

and the subdivision matrix $\mathbf{S}_4$ for the particular valence 4:

$$\mathbf{S}_4 = \frac{1}{16}\begin{bmatrix} 9 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 6 & 6 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 6 & 1 & 6 & 1 & 0 & 1 & 1 & 0 & 0 \\ 6 & 0 & 1 & 6 & 1 & 0 & 1 & 1 & 0 \\ 6 & 1 & 0 & 1 & 6 & 0 & 0 & 1 & 1 \\ 4 & 4 & 4 & 0 & 0 & 4 & 0 & 0 & 0 \\ 4 & 0 & 4 & 4 & 0 & 0 & 4 & 0 & 0 \\ 4 & 0 & 0 & 4 & 4 & 0 & 0 & 4 & 0 \\ 4 & 4 & 0 & 0 & 4 & 0 & 0 & 0 & 4 \end{bmatrix} . \tag{2.34}$$

This matrix can be obtained i.e. by accumulating the general Catmull-Clark equations 2.20, 2.21 and 2.22 from subsection 2.1.5.2 on page 29.

Now, pertaining to equation 2.10 in section 2.1.3.2 on page 17 $\mathbf{G}^{k+1} = \mathbf{S}\mathbf{G}^k$ and recalling the recursive generation of the subdivision mask, it can be also stated that any subdivision level can be obtained from the initial mesh by powers of the matrix $\mathbf{S}$

$$\mathbf{G}^k = \mathbf{S}\mathbf{G}^{k-1} = \mathbf{S}^k\mathbf{G}^1 . \tag{2.35}$$

Further on, since every vertex $\mathbf{v}^k$ is affected only by its first-ring neighbors and its valence and connectivity does not change after the very first step anymore, the resulting vertex $\mathbf{v}^{k+1}$ and its neighborhood is always self-similar to its previous level. This self-repetition is shown in Figure 2.20.

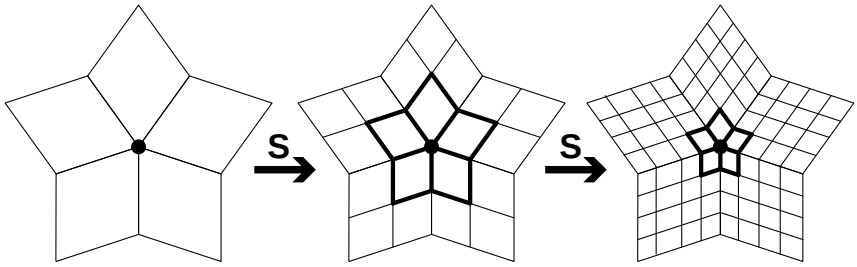


Figure 2.20.: *Extraordinary vertex of valence 5. Left at level k and right at level $k+1$ followed by level $k+2$. The emphasized bold edges mark the neighborhood which directly influences the vertex. It is self-similar for all subdivision steps.*

According to equation 2.35 the limit position of the surface at $\mathbf{v}$ basically depends only on the properties of the local subdivision matrix at $\mathbf{v}$. Its properties can be examined by the spectral analysis. Let $\lambda_0 \geq \lambda_1 \geq \lambda_2 \geq \ldots$ be distinct eigenvalues of $\mathbf{S}$ ordered by their magnitude and $\mathbf{z}_0, \mathbf{z}_1, \mathbf{z}_2, \ldots$ the corresponding eigenvectors. Assuming the eigenspace as linear independent, the positions of the vertex ring $\mathbf{G}^1$ can be expressed as a linear combination of the eigenvectors

$$\mathbf{G}^1 = \sum_{i=0}^{n-1} \mathbf{a}_i \mathbf{z}_i \tag{2.36}$$

where $\mathbf{a}_i$ are the coefficients of this linear system and can be computed by the left eigenvectors $\tilde{\mathbf{z}}$ as $\mathbf{a}_i = \tilde{\mathbf{z}}_i \mathbf{G}^1$ for each input vector $\mathbf{G}^1$. By using equation 2.35 and keeping the recursion in mind as well as the properties of powers of eigenvalues, equation 2.36 can be extended to

$$\mathbf{G}^k = \mathbf{S}^k \mathbf{G}^1 = \sum_{i=0}^{n-1} \lambda_i^k \mathbf{a}_i \mathbf{z}_i \,. \tag{2.37}$$

Convergence. In the current instance, matrix $\mathbf{S}_4$ possesses the distinct set of eigenvalues $\left\{1, \frac{1}{2}, \frac{1}{2}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{16}\right\}$ which are ordered descending. Since the eigenvalues satisfy $|\lambda_0| = 1 > |\lambda_i|$ for $i > 0$ all λ_i^k up from $i > 0$ converge to zero for $k \to \infty$. Therefore, the entire expression converges to the vector $\mathbf{a}_0 \mathbf{z}_0$ for $k \to \infty$. Pertaining to the definition of $\mathbf{G}^1$ in equation 2.33 and to equation 2.37, the only affected point is the first entry in the vector $\mathbf{G}^k$. Thus, the limit point is only determined by the largest eigenvalue of the matrix $\mathbf{S}$, which has in this instance (and all other valences of Catmull-Clark as shown in [Rei95, PR98]) the absolute value of exactly 1. Furthermore, hence the subdivision matrix is affine invariant, which means that all its rows sum to 1, the dominant eigenvector is always $\mathbf{z}_0 = [\ldots, 1, 1, 1, \ldots]$ such that the limit point is $\mathbf{v}^\infty = \mathbf{a}_0$.

This examination shows at its conclusion that if a subdivision matrix $\mathbf{S}$ can be shown to posses a largest eigenvalue $\lambda_0 = 1$, than the one-ring neighborhood around a vertex $\mathbf{v}$ converges to a limit at $\mathbf{v}^k$ for infinite subdivision steps k. This fact has its reason in the properties of a matrix, such that all other eigenvalues vanish for growing $k \to \infty$. Unfortunately, this method is extremely time consuming and inflexible, since every matrix for any valence has to be constructed and proved separately. Therefore, Reif, Peters, Stam and Zorin have extended this analysis to more sophisticated tools, which – though all based on the eigen-basis of the matrix – allow to achieve further and faster results.

2.1.6.5. Exact Evaluation and Derivatives

So far the method for the convergence proof has been presented – which does not ensure automatically a certain continuity of a surface. Since the tools used therefore go either into the spectral analysis of the characteristic map or can be overcome by the discrete Fourier analysis of the subdivision matrix, this thesis will only summarize the results achieved in over two decades of studying subdivision surfaces. Halstead *et al.* [HKD93] have investigated this problem in the year 1993. They have shown with a discrete Fourier analysis of the subdivision matrix (first introduced by Ball *et al.* [BS88]), that the limit surface at $\mathbf{v}$ with the neighborhood as defined in $\mathbf{G}^1$ for each valence n is

$$\mathbf{v}^\infty = \tilde{\mathbf{z}}_0 \mathbf{G}^1 = \frac{n^2 \mathbf{v}^1 + 4\sum_{j=0}^{n-1} \mathbf{e}_j^1 + \sum_{j=0}^{n-1} \mathbf{f}_j^1}{n(n+5)} \,. \tag{2.38}$$

Note that $\tilde{\mathbf{z}}_i$ are the left eigenvalues. Furthermore, the tangent vectors $\mathbf{t}_1^\infty$ and $\mathbf{t}_2^\infty$ at the infinity can also be obtained by this method by taking the second and third eigenvalues into account, where n denotes the particular valence:

$$\mathbf{t}_1^\infty = \tilde{\mathbf{z}}_1 \mathbf{G}^1 \quad \text{and} \quad \mathbf{t}_2^\infty = \tilde{\mathbf{z}}_2 \mathbf{G}^1 \,, \tag{2.39}$$

which equals to the formulas obtained by the discrete Fourier analysis

$$\mathbf{t}_1^\infty = \sum_{j=0}^{n-1} A_n cos\left(\frac{2\pi j}{n}\right) \mathbf{e}_j^1 + (cos\left(\frac{2\pi j}{n}\right) + cos\left(\frac{2\pi (j+1)}{n}\right)) \mathbf{f}_j^1 \tag{2.40}$$

and

$$\mathbf{t}_2^\infty = \sum_{j=0}^{n-1} A_n cos\left(\frac{2\pi j}{n}\right) \mathbf{e}_{j+1}^1 + (cos\left(\frac{2\pi j}{n}\right) + cos\left(\frac{2\pi (j+1)}{n}\right)) \mathbf{f}_{j+1}^1 \tag{2.41}$$

where

$$A_n = 1 + cos\left(\frac{2\pi}{n}\right) + cos\left(\frac{2\pi}{n}\right)\sqrt{2(9 + cos\left(\frac{2\pi}{n}\right))}\,,$$

which is for the regular case $A_4 = 1$. Finally, obviously the normal at the limit is

$$\mathbf{n}^{\infty} = \mathbf{t}_1^{\infty} \times \mathbf{t}_2^{\infty}\,. \tag{2.42}$$

The equations showen above allow one to compute the exact limit points at chosen vetices at any resolution level k. Unfortunately they do not allow to compute points on arbitrary positions, but only on given vertices with specified neighborhood. This constrains the formulas to an exactly computed sparse mesh. Alternatively, they could be applied on vertices, when the final desired resolution has been achieved by the recursive approach, which is usually of very minor benefit. For computation of exact values at any points refer to Joe Stam's article [Sta98] and to a series of works of Zorin beginning with [ZK02].

Continuity. Reif has introduced a functional $\Psi(u,v) = \mathbf{v}^{\infty}(u,v)$, called *characteristic map* in order to perform more flexible analysis at irregular vertices (refer to Figure 2.21). According to this mapping also continuous parametrization around a point of any valence can be defined by the means of the eigenspace. This is possible, because the tangent space can by spanned by the two sub-dominant left eigenvectors $\tilde{\mathbf{z}}_1$ and $\tilde{\mathbf{z}}_2$ as showed in equations 2.39. By the means of the characteristic map he could show the C^1 continuity by proving its regularity and showing that its is injective [Rei95].

Also in terms of the characteristic map, Stam has developed a method to evaluate the subdivision analytically in the vicinity of extraordinary points [Sta98] and Zorin has introduced a new, more flexible and general method to proof the C^1 continuity [Zor00]. More recently, Boier-Martin and Zorin have shown a method for parametrization in the near of irregular vertices, such that the derivatives can be defined anywhere [BMZ04].

These cases as well as the evaluation in the near of extraordinaries will not be considered here due to their sophisticated nature, which would go beyond the scope of this work. For treatment of those refer to sources given in this section [BS88, HKD93, Rei95, PR98, Sta98, Zor00, WW01, ZK02, BMZ04].

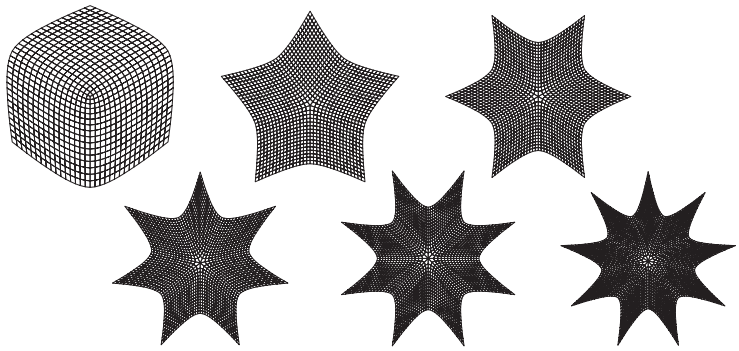


Figure 2.21.: *Characteristic Maps for extraordinary vertices of valence 3,5,6,7,8 and 9. Figure courtesy from [SZD+98].*

2.1.7. Summary

This entire section has introduced the theory behind subdivision surfaces on the particular example of the Catmull-Clark method. It should be mentioned, that not all schemes are derived from piecewise polynomials and not all ensure C^2 continuity. For a general treatment of the subdivision surfaces methods refer to Warren and Weimar [WW01] and other sources proposed in this section.

2.2. Displacement Surfaces

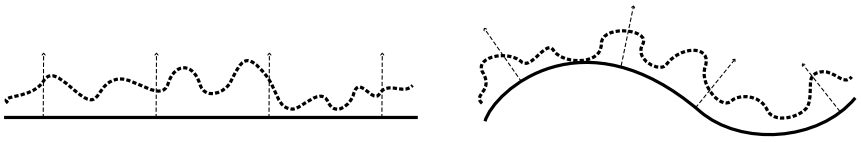


Figure 2.22.: *Two examples of displacement of a continuous surface along normals. Left: trivial case: plane. Right: a smooth curved surface.*

Surface displacement is a quite simple yet powerful technique to achieve interesting deformations. Basically it can be applied to any surface by simply adding an offset value d in a direction at an arbitrary surface position. If it is done constantly over the entire surface, a new surface, called the *offset surface* will be generated. Moreover, since the magnitude of the offset can be varied over the entire domain, the generated surface may exhibit significant difference to its origin. In computer graphics this technique became popular due to the possibility to create naturally looking deformations on smooth surfaces (see Figure 2.22).

This thesis is significantly concerned with the application of such deformations on different resolutions of a subdivision hierarchy and this section will elaborate the basic theory behind displacement.

2.2.1. General Displacement Approach

The general approach of displacement mapping is to define for each surface point $\mathbf{p}$ an offset value d. It is usual to apply such displacement on continuous parametrized surfaces such that for each surface position u, v a particular offset value is determined by a bivariate planar function $d(u,v)$.

There are many methods to construct a global parametrization, unfortunately there is not a single one which can be automatically applied to all kinds of objects. Nonetheless in todays computer graphics it has become a must to supply textures to rendered models. Thus it is very often left to the user to define a appropriate *texture coordinates* on a model. For the displacement approach these texture coordinates can be directly reused to define a displacement map $d(u,v)$ as a texture.

Although in the case of subdivision surfaces a natural parametrization is supplied by the scheme itself as described in section 2.1.6, this parametric space do not cover the extraordinary vertices and has to be enhanced by hand in order to achieve a continuous texture map. Once this is done for the initial control domain it can be propagated to all sub-meshes and it is rather easy to determine the position on the surface.

The most obvious possibility to choose the direction vector for displacement is of course the local surface normal $N(u,v)$. As defined in section 3, let the surface S be parameterized by $X : U \subseteq \mathbb{R}^2 \to \mathbb{R}^3$. Following this parametric space, a displacement function is defined as

$$\hat{X}(u,v) = X(u,v) + d(u,v) \cdot N(u,v) \tag{2.43}$$

where $N(u,v)$ is the continuous surface' unit normal as shown in Figure 2.22.

In the case of discrete meshes, the normal can be easily computed on each vertex as the average of its neighboring face normals, which again can be obtained by the cross products of the vectors spanning each face incident to $\mathbf{v}_i$:

$$\mathbf{n}_i = \frac{1}{n} \sum_{j}^{n} \left\| (\mathbf{v}_j - \mathbf{v}_i) \times (\mathbf{v}_{j+1} - \mathbf{v}_i) \right\| ,$$

where n is the vertex valence and j is taken module n. Thus the displacement mapping can be stated for each vertex $\mathbf{v}_i$ as

$$\hat{\mathbf{v}}_i = \mathbf{v}_i + d(\mathbf{tc}_i) \cdot \mathbf{n}_i$$

where $\mathbf{tc}_i$ represents a *texture coordinate* $\mathbf{tc}(u,v)$ in the parametric space. It is not necessary to attach the offset function d to texture coordinates, albeit it is one convenient possibility. Since there are no limitations to the number of parameterizations supplied to a surface, in this chapter the terms *texture coordinates* and *planar parametrization* of a surface will be used interchangeably to denote the latter one, since texture coordinates are a particular instance for a parametrization.

2.2.2. Maximal Displacement Offset

The displacement map $d(u,v)$ itself can be defined in various ways and will be issued in section 2.3. But independently of the source for the offset some important constrains have to be defined in order to ensure that the chosen offset does not lead to undesired surface deformations.

As shown in Figure 2.22, left hand side, displacement can also be applied on a planar surface by equation 2.43. This technique used in the terrain rendering filed of computer graphics is the easiest possible and rather trivial. Since the surfaces' derivatives are trivial, it does not possesses any curvature and the normals direction is constant over the entire domain, basically any kind of offset can by applied without the danger of

destroying the manifoldness. Similar cases appear at topologically trivial shapes such as a sphere.

The situation changes drastically in the case of curved surfaces (see Figure 2.22, left hand side) where, due to the presence of curvature, normals can point to varying directions and surface self-intersections can appear. Thus it is important to formally examine possible situations and define proper constrains to the function d, so that those cases can be avoided.

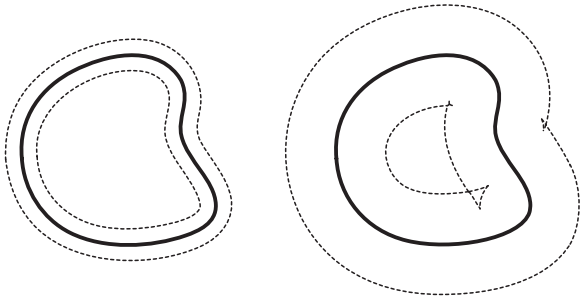


Figure 2.23.: *Surface displacement. Discontinuities on a displaced surface can occur if the offset is higher than the minimum radius of the curvature (left). Figure courtesy from [Ped94].*

Pederson [Ped94] has observed that the mentioned discontinuities can occur where the size of the displacement is higher than the minimum radius of the local curvature. Also Barr [Bar84] has investigated the properties of global and local impacts of deformations on surfaces as well as other researchers, who have dealt with the problem of both *global* and *local self-intersections* of offset curves and surfaces as i.e. Wallner *et al.* [WSM+01] [Wal01] and Elber and Cohen [EC91]. In general one can say that it is not a trivial task to determine a 'safe' offset for all cases, such that it is ensured that no self-intersections will appear. For this thesis, the assumption stated by Pedersen will be taken as sufficient to present a rather straight-forward method to avoid local mesh fold-overs by the means of the local surfaces' curvature.

The possibility of global self-intersections will not be concerned in this thesis at all. Wallner *et al.* [WSM+01] present a possibility to define a maximum offset by the means of the *cut locus surface* (related to *medial axis surface*), which is defined as the set of all centers of all spheres which touch the initial surface in at least two points. The implementation of such method would involve global (pre-)computations of the model and it would be hardly possible to implement it on graphics hardware for real-time rendering. Thus, it is left to the responsibility of the user to supply models which do not exhibit regions sensitive to global self-intersection. For deeper mathematical deliberations on this topic refer to sources mentioned above.

2.2.2.1. Surface Curvature

Usually, as known from differential geometry [Car93], the curvature of a planar parametric curve $C(u) \in \mathbb{R}^2$ at some point $P(u_0)$ can be obtained very easily by taking the second derivative of C at P. The magnitude of the vector $\left|\frac{d^2C}{du^2}(u_0)\right|$ delivers a single value which is the local curvature κ_P. The radius of the curvature can be computed along the curve as a continuous function $r_\kappa(u)$ (if assumed that the curve is parametrized by arc length and everywhere two times differentiable):

$$r_\kappa(u) = \kappa(u)^{-1} \ .$$

Intuitively, it is obvious, that the offset-stripe continuously defined along the surface with the distance of r_κ can not lead to intersections, since the normals $r_\kappa(u)\mathbf{n}(u)$ will never intersect as can be observed in Figures 2.24 and 2.23 left hand side. Elber *et al.* [EC91] and Wallenr *et al.* [WSM+01] provide the formal proofs for this case.

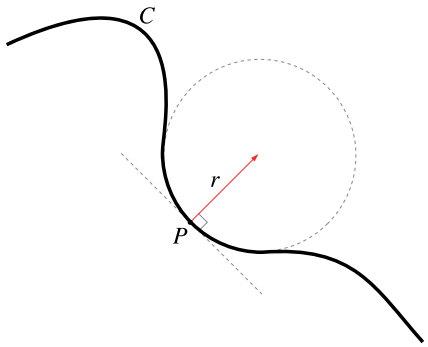


Figure 2.24.: *Curvature radius r_κ defined at a point P on a curve $C \in \mathbb{R}^2$. Figure courtesy from [Wik07].*

Normal Curvature. In the case of a surface $S \in \mathbb{R}^3$ the situation becomes a bit more sophisticated. It is no more a trivial task to determine a curvature on a regular surface. The problem lies in the fact, that at each point P on the surface there exist an infinite number of curves lying on the surface and crossing exactly at $P(u_0, v_0)$, where each of them possesses some curvature. For each of the curves the curvature at P can be obtained by measuring the magnitude of the curves' second derivative at P with respect to the surface normal. This definition is shown in Figure 2.25, where the normal curvature of the curve C is defined by the value κ_n which is equal to the angle $cos(\varphi)$ times the length of the second derivative $\mathbf{c}''$:

$$cos(\varphi) = \frac{N(p)\mathbf{c}''(p)}{|N(p)||\mathbf{c}''(p)|}$$

As an alternative interpretation, the value of κ_n can be obtained by the length of the projection of the vector kn on the surface normal $N(p)$. The sign of the value is defined by the orientation of N [Car93]. Formally, normal curvature is

$$\kappa_n = \mathbf{N}(p)\mathbf{c}''(p)\ . \tag{2.44}$$

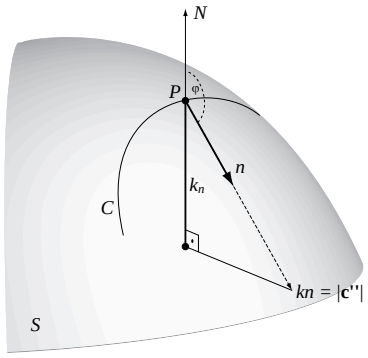


Figure 2.25.: *Normal Curvature Definition.*

Obviously, in this bundle of curves through P there must exist exactly two curves C_1 and C_2 each having either the largest or the smallest curvature (referred as the maximal and minimal *principal curvatures* κ_1 and κ_2). Furthermore, there must be two vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ tangent to the surface S and to the two particular curves C_1 and C_2 respectively. These tangent vectors cross in the point of interest $P(u_0, v_0)$ and define the *principal directions* of κ_1 and κ_2. Except the trivial cases as i.e. a sphere or a cylinder, the particular *principal directions* are not known.

Gauss Map. To define the principal curvatures of a surface at P it is useful to introduce the *gauss map* of a surface S (or better the trace of the surface) to the unit sphere as $N : S \to S^2$, such that $S^2 = \left\{(x,y,z) \in \mathbb{R}^3;\ x^2+y^2+z^2 = 1\right\}$. This mapping allows to express the entire normal vector filed of a surface on the unit sphere. Respectively, each normal $\mathbf{n}_p$ at point $P \in S$ can be seen as $N(p) = (x,y,z)$ on the unit sphere S^2 (refer to Figure 2.26).

The gauss map is obviously differentiable and the differential dN_p can be seen as a linear transformation of the tangent space $T_p(S)$ into the tangent space on the sphere $T_{N(p)}(S^2)$. This means that each tangent plane on S^2 at $N(p)$ is in fact parallel the tangent plane

of P on S (see Figure 2.26). They are shifted by the unit normal length (which is one). Thus the mapping is a self-adjoint mapping of the form $dN_p : T_p(S) \to T_p(S)$. It should be noted that such a mapping is globally only possible on uniquely orientated surfaces.

Recall that a tangent $\mathbf{c}'$ of a curve $C(s) = X(u(s), v(s))$ lying in S (equation 2.30) can be expressed by the means of the local parametrization $X(u,v) : U \to S$ on the tangent plane at $P(0)$ as $\mathbf{c}' = \mathbf{x}_u u' + \mathbf{x}_v v'$. Thus, the point $\mathbf{c}'$ can be expressed as a linear combination of the tangent plane basis $\{\mathbf{x}_u, \mathbf{x}_v\}$ with respect to dN_p as

$$dN_p(\mathbf{c}') = N'(u(s), v(s)) = (a_{11}u' + a_{12}v')\mathbf{x}_u + (a_{21}u' + a_{22}v')\mathbf{x}_v, = N_u u' + N_v v' \,. \quad (2.45)$$

Thus, the basis vectors of the parametrization X can be also stated in terms of dN_p:

$$dN_p(\mathbf{x}_u) = N_u \quad \text{and} \quad dN_p(\mathbf{x}_v) = N_v;$$

and the equation above can be also put into a matrix form:

$$dN_p = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} u' \\ v' \end{bmatrix} . \quad (2.46)$$

This differential is also often referred as the *shape operator* of a surface S and denoted as $\mathcal{S}_p = -dN_p$, where the minus before the operator is caused by the inverted normal direction on the map with respect to the surface orientation.

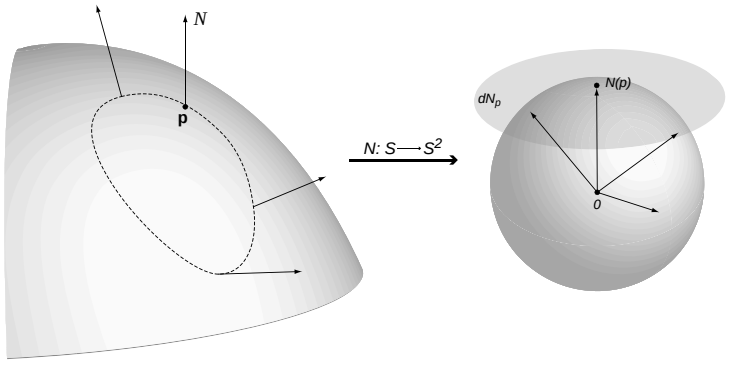


Figure 2.26.: *Gauss map. All unit normals of an orientable surface S can be translated to the origin and form a unit sphere S^2.*

While the curvature is defined by the means of the gauss map, it is not clear yet how to compute the matrix in equation 2.46. To do it, two more essential items of surfaces will be introduced below.

First Fundamental Form. According to differential geometry [Car93], the curvature of a surface S at some point $P(u,v)$ can be stated in terms of the the *first* and *second fundamental forms* of the surface parametrization $X(u,v)$ at P. The first fundamental form is basically a tool to define the inner product $\langle , \rangle_P$ in the local coordinate frame of some point $P(u_0,v_0)$ with respect to a local parametrization $X(u,v)$ (refer to section 2.1.6). In other words it defines a square norm of the tangent vector $\mathbf{c}'$ of some curve $C \subset S$ passing through $P(u_0,v_0)$ in terms of the chosen local tangent space basis $\{\mathbf{x}_u, \mathbf{x}_v\}$. Since the inner product of the Euclidean space is the well known dot product, the notations $\langle \mathbf{a}, \mathbf{b} \rangle$ and $\mathbf{ab}$ will be used from now on interchangeably. Thus, for any parametric curve $c(u,v)$ on a surface $S \in \mathbb{R}^3$, with a tangent vector $\mathbf{c}' = (u', v')$ at $P(u_0,v_0)$ the first fundamental form satisfies:

$$I_p = \left|\mathbf{c}'\right|_p^2 = \left\langle \mathbf{x}_u u' + \mathbf{x}_v v', \mathbf{x}_u u' + \mathbf{x}_v v' \right\rangle_p = \mathbf{x}_u \mathbf{x}_u (u')^2 + 2(\mathbf{x}_u \mathbf{x}_v) u' v' + \mathbf{x}_v \mathbf{x}_v (v')^2 \,,$$

where E, F, G are usually stated as the coefficients of this quadratic form:

$$\begin{aligned} E &= \mathbf{x}_u \mathbf{x}_u \\ F &= \mathbf{x}_u \mathbf{x}_v \\ G &= \mathbf{x}_v \mathbf{x}_v \,. \end{aligned} \tag{2.47}$$

The first fundamental form is thus a distance measure of the length of $C \subset S$ in P and if the curve is parameterized by arc length, it is a unit of it[4].

Second Fundamental Form. Now the second fundamental form of a surface S can be defined by the means of dN_p. To do this recall the definition of a parametric curve $C \subset S$ passing through P and parametrized by s as presented in equation 2.30. Assuming this curve passing through $P(u_0,v_0)$ at $s = 0$, such that $C(s)$ has the tangent vector $\mathbf{c}'(s) = \mathbf{c}'$ exactly at P. Furthermore, the normal vectors of S can be restricted to the trace of C on S and referred as $N(s)$, so than the inner product of N and $\mathbf{c}'$ at P is $\langle N(s), \mathbf{c}'(s) \rangle = 0$.

Recalling the definition of the normal curvature from equation 2.44 and the definition of the gauss map from equation 2.45 the second fundamental form is defined as:

$$II_p(\mathbf{c}'(s)) = \left\langle N(s), \mathbf{c}''(s) \right\rangle = -\left\langle dN_p(s), \mathbf{c}'(s) \right\rangle = \kappa_n(p) \tag{2.48}$$

In other words, the magnitude of II_p at point P is for a unit vector $\mathbf{c}' \in T_p(S)$ equal to the normal curvature of a regular curve $C \subset S$ passing through P and having $\mathbf{c}'$ as its tangent

[4] It can be shown [Car93], page 18, that any arbitrary parametrization of a curve or surface in $\mathbb{R}^3$ can be transformed into an arc length parametrization. A parametric curve C can by parametrized by arc length s by the integral $C(s) = \int_{t_0}^{t} |C'(t)| dt$ with $t_0, t \in \mathbb{R}$. Thus the properties discussed here do not depend on the chosen parametric mapping but only on the local characteristics of the regular curve/surface.

at P. While it may sound confusing, II_p can be seen a the measure of the curvature of some curve in the surface at some point P.

The coefficients of II_p are usually denoted as

$$\begin{aligned} e &= -\langle N_u, \mathbf{x}_u \rangle = \langle N, \mathbf{x}_{uu} \rangle \\ f &= -\langle N_u, \mathbf{x}_u \rangle = \langle N_v, \mathbf{x}_v \rangle = \langle N, \mathbf{x}_{uv} \rangle = \langle N, \mathbf{x}_{vu} \rangle \\ g &= -\langle N_v, \mathbf{x}_v \rangle = \langle N, \mathbf{x}_{vv} \rangle \end{aligned} \tag{2.49}$$

Now, the equation 2.46 of the gauss map can be fully determined by the coefficients of the fundamental forms:

$$-\begin{bmatrix} e & f \\ f & g \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} E & F \\ F & G \end{bmatrix} .$$

The unknown coefficients matrix $\left[\begin{smallmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{smallmatrix}\right]$ is known in the literature as the Weingarten Matrix and can now be calculated by

$$\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} = -\begin{bmatrix} e & f \\ f & g \end{bmatrix} \begin{bmatrix} E & F \\ F & G \end{bmatrix}^{-1}$$

The interesting property of the Weingarten matrix is the fact that its eigenvalues deliver the maximal and minimal principal curvatures κ_1 and κ_2 and the eigenvectors the principal directions therefore.

Furthermore is also common to define the *gaussian* κ_K and *mean curvature* κ_H as a combination of the two *principal curvatures* κ_1 and κ_2:

$$\kappa_K = \kappa_1 \kappa_2 \tag{2.50}$$

and

$$\kappa_H = \frac{\kappa_1 + \kappa_2}{2}$$

The presented relations are enough to define a proper offset radius for a 'safe' displacement distance. For further discussion on the curvature and classical differential geometry of curves and surfaces refer to Do Carmo [Car93].

Avoiding Self-Intersections. Wallener *et al.* [WSM+01, Wal01] have shown that an offset surface E of a regular surface X of the form

$$E(u,v) = rN(u,v) + X(u,v)$$

will become singular exactly at a point $P(u,v)$ if and only if the magnitude of r at P is reciprocally equal to one of the principal curvatures κ_1 or κ_2. The singularity of the surface means that there in no unique inverse mapping $E^{-1} : S \subset \mathbb{R}^3 \to U \subset \mathbb{R}^2$ of the parametrization of E and that the Jacobian matrix $\mathbf{J}_p$ at P, as defined in section 3, equation 2.29 becomes non-invertible. In other words the offset surface E develops a cusp, where it is not differentiable. This violates conditions 2 and 3 of definition 3 in section 3, where a manifold surface has been defined. The proof can be done formally by constructing the particular offset surface $E : (X,r) \to X(p) + rN(p)$ and the gauss map $dN_p = -\kappa_i \mathbf{c}_i'$. Let the tangent $\mathbf{c}_i'$ be the first derivative of some curve C at P and also an eigenvector of the shape operator $-dN_p$ corresponding to the eigenvalue κ_i at P. This allows to express $-dN_p$ in its eigenspace as a linear combination as $dN_p = -\kappa_i \mathbf{c}_i'$. Now, the differential dE_p can be formed:

$$dE_p = dX_p + r\, dN_p + dr_p\, N(p)\ ,$$

which delivers with $dX_p = \mathbf{c}_i'$, $r = \frac{1}{\kappa_i}$, $dr_p = 0$ and $dN_p = -\kappa_i \mathbf{c}_i'$ following equation:

$$dE_p = \mathbf{c}_i' + \left(\frac{1}{\kappa_i}\right)\left(-\kappa_i \mathbf{c}_i'\right) = \mathbf{c}_i' - \frac{\kappa_i \mathbf{c}_i'}{\kappa_i} = 0 \quad q.e.d.$$

This derivative should deliver the tangent space of E at P, but because it is zero a contradiction has appeared. Note, that this formulation is true if and only if $\mathbf{c}_i'$ is an eigenvector of dN_p and implies that it is one of the principal directions at P and corresponds to a principal curvature κ_i.

Thus, in order do ensure the manifoldness it is desired to keep the offset distance d in interval $0 < r < \frac{1}{\kappa_i}$ with $i = 1,2$ or, if the displacement is performed in both surface normal directions in $-\frac{1}{\kappa_i} < r < \frac{1}{\kappa_i}$. This issue will be addressed again in section 4.1.2.

2.2.2.2. Discrete Curvature.

While it is interesting to observe how the analytical curvature of a surface can be calculated, the equations presented above have little direct application on polyhedral meshes. The computer graphics literature proposes several ways of the approximation of discrete curvatures, especially for the gaussian and mean measures. Two of them should be presented here. For an exhaustive discussion refer to Surazhsky *et al.* [SMS+03] and Dyn *et al.* [DHKL01].

The gaussian curvature can be computed on triangular meshes by

$$\kappa_K = \frac{1}{A_v}\left(2\pi - \sum_i^n \alpha_i\right) \tag{2.51}$$

with $\alpha_i = arccos\left(\frac{(e_i - v)(e_{i+1} - v)}{|e_i - v||\ e_{i+1} - v|}\right)$ and $(i\ mod\ n)$.

For the mean curvature the approximation can be obtained by

$$\kappa_H = \frac{1}{A_v}\left(\frac{1}{4}\sum \|\mathbf{v} - \mathbf{e}_i\|\,|\beta_i|\right) .$$

In both cases the $A_v = \frac{1}{3}A$ with A as the sum of the areas of all triangles incident to the vertex $\mathbf{v}$. The fraction $\frac{1}{3}$ defines the barycentric area of each triangle [DHKL01].

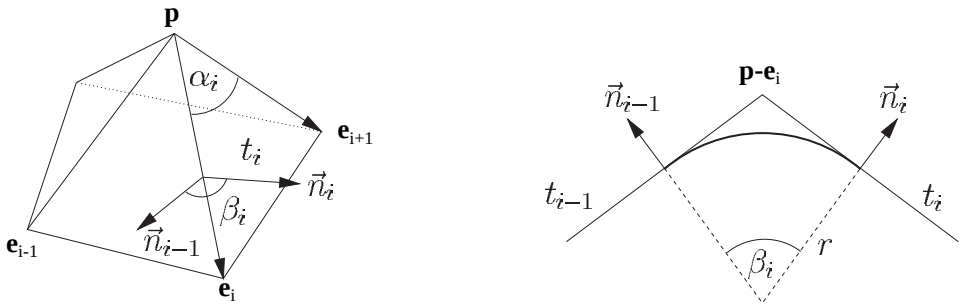


Figure 2.27.: *Discrete curvature of a polyhedral mesh. Figure courtesy from [DHKL01].*

By the means of this values it is rather easy to estimate local curvature on a discrete mesh and to compute a theoretically 'safe' radius of displacement in order to avoid local self-intersections.

2.2.3. Point of Reference for Displacement Offset

In the previous section the maximum allowed displacement distance was discussed in terms of the surface curvature. Nevertheless, the question of the appropriate offset distance in order to achieve desired visual results has not yet been addressed. So far, it has been stated that the displacement happens along the unit vertex normal $\mathbf{n}_v$, but here arises the discussion about a point of reference of the magnitude for the scaling. In this context it should be mentioned that there exist basically two different frames of characteristics of a surface: on the one hand the global characteristics (*extrinsic*), which depend on the environment of the surface as i.e. the space $\mathbb{R}^3$ and on the other hand the

local characteristics (*intrinsic*) that depend only on the surface properties. A member of the first ones is obviously the unite surface normal, which has a well defined length in the embedding euclidean space $\mathbb{R}^3$, where as an example of the the second ones the curvature can be stated, which is independent of the global space. See Figure 2.28 to compare the actuality.

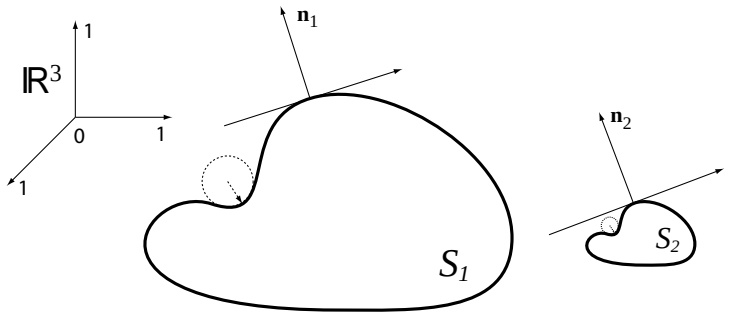


Figure 2.28.: *Global and local characteristics of a surface. While the unit normal vectors lengths $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ stays identic on both surfaces S_1 and S_2 and are independent of the parametrization, the curvature radii (symbolized by the dashed circles) depend locally on each surface.*

2.2.3.1. Referencing on Intrinsic Properties

It should be recalled that the aim of this work is to develop a method to create fine details across the surface in order to let it appear more naturally. Especially the generation of tree bark, leather or rough stones is attempted. For this purpose an intuitive approach to attach the offset magnitude would be therefore not the local surface curvature, but some object-derived measure, which might be even constant over the entire domain. It would allow to define the appropriate degree of ripples without highlighting flat against more curved regions (in fact, totally flat regions do not posses any curvature, thus some other reference or interpolation would be needed). Attaching this measure on the surface normal only on the other hand would produce a global space dependent offset which would not consider an objects dimensions (see next subsection 2.2.3.2).

Texture Coordinates. A suitable intrinsic reference for the offset is the surfaces' parametric space as defined for texture mapping. The offset can be than attached to i.e. an averaged distance D of the neighboring points in the u,v coordinates, which is:

$$D = \frac{1}{n}\sum_{i}^{n} |P(u,v) - P_i(u,v)| \ ,$$

where n is the valence and P the parametric position on the surface of some vertex $\mathbf{v}$. Although this measure is rather useful, it requires usually a continuous parametrization over the entire domain because sharp junctions would lead to falsified lengths.

As an interesting fact it should be noted that in fact arc length measure is related to the parametrization of a surface. It can be transformed into the natural parametrization of subdivision surfaces described in section 2.1.6.2. This case would be the perfect one, since global and local properties would have a well defined norm.

2.2.3.2. Referencing on Extrinsic Properties

While local surface properties allow the definition of suitable offsets along the surface in their own dimensions, global influence on the displacement can be performed too. As an representative example one can consider a displacement of objects of rather different sizes embedded in the same space and displaced by a constant value along the normal (see Figure 2.28). Naturally, this would lead to extremely different results for both objects – the bigger one will develop a rather moderate surface distortions, while the other one will change into a extreme morning star. Thus, this approach is said to be applied with caution in order to define new shapes.

Local Faces Area. A promising global reference might be the area of the surface measured in (x,y,z) coordinates of the points. Attaching the offset onto it would cause a displacement in the means of the surface range in global space on the one hand and a more or less constant stripe along the object on the other. A intuitive approach to obtain a rough approximation of the area along the surface can be achieved by measuring the area of polygons incident to a point. Please note, that it depends on the object dimensions, such that models of the same shape but different scale would employ different results.

Nonetheless, this measure is very well suited to determine the offset along the surface and deformations defined by its means can be controlled by adding a (user defined) scaling factor to the derived local offset. On the other hand this idea is also rather sensitive to the tessellation of an object, and thus it might develop undesired deformations if the sizes of adjacent polygons are not quite uniform. Fortunately, this issue does not concern subdivision surfaces which develop excellent tessellations.

2.2.4. Displacement Scaling

In the previous sections a suitable reference measure for surface offset has been discussed in terms of local and global embeddings. As a last parameter for influencing the distance, a global scaling function should be mentioned. For a simple displacement a

global scaling does not play any considerable role because it allows only just to scale all offsets by a constant value. This becomes significant in a procedural displacement, as in the method presented in this work. In such cases a global scaling function can be applied successively over several displacement steps of different resolutions of the same mesh. This sequence enables to influence each level by a well defined function $s(k)$ with k as the subdivision level. Generally, there can be two main types of a scaling functions defined [VPYB01]:

1. Proportional to the subdivision level: $s(k) = k$.
2. Inversely proportional to subdivision level $s(k) = \frac{1}{k}$.

While the first ones develop objects which usually grow or are 'blown up', the latter allows to define fractal like surface distortions. Additionally the function can take the offset referencing measure into account and thus, an interesting set of possible constellations can be created. This context will be further discussed in sections 4.2.4 and 5.

2.3. Displacement Maps

In the previous section the constrains for the displacement offset along the surface have been discussed. The particular source of the displacement has not yet been mentioned. As described, displacement along the surface can by applied by offset values obtained from a planar map $d(u,v)$ which follows the surface by its parametrization or usually particularly the texture coordinates. This method is well known from texture mapping, where for each vertex a color value is fetched from the map in order to attach it to a vertex. Colors between vertices can be linearly interpolated.

Different from texture mapping, the obtained values can be used to shift each vertex along its normal as described before. Thus, it is enough to supply a single scalar per vertex. Usually it is done in an additional texture or in the alpha channel of already supplied textures. In order to the way how textures are stored, the range for each scalar is defined in an interval $[0..1]$ as floating point numbers. Than the surface vertices can be distorted by the means defined in the underlying texture image. In most cases these images contain patterns known from real life surfaces as i.e. wood, stone, leather etc. structures. In height-field mapping, the patterns contain usually either some geodesic data or often stochastic Perlin-Noise distributed structures, which were defined procedurally.

The displacement textures used in the approach presented by this thesis are basically similar to the described ones. The main difference is that the patterns contained in the images are previously decomposed into their spectral components. The next sections will explain how this can be achieved.

2.3.1. Discrete Images

As described in the displacement surfaces section, the bivariate function which contains the information for surface perturbations can be defined as a discrete, two-dimensional image. The research area which deals with the analysis of discrete images is referred to as *image processing* [GW01].

An image will be expressed in mathematical terms as a function

$$I = \sum_{i=0}^{n} \sum_{j=0}^{m} p_{i,j} \tag{2.52}$$

where i, j denote coordinates of a *pixel* $p_{i,j}$ in a two-dimensional regular grid which can be also interpreted as an two-dimensional array. A *pixel* (picture element) is defined as a unique simplex of the image and it contains some color value. There are several approaches in computer graphics to define the color values, but in this work – without

going too far into the world of image processing [GW01] – the RGBA color space will be used without loss of the generality. This has been chosen due to two basic reasons: (1) the RGBA color mode is directly supported by the graphics hardware and (2) it can be also interpreted as a four-dimensional vector space. Thus, a pixel can be stated as a vector containing its four components

$$\mathbf{p}_{i,j} = \begin{bmatrix} r & g & b & a \end{bmatrix}^T$$

In traditional image processing, again due to hardware constrains it is common to use an 8 bit coding for each color component, such that the range of the possible colors is limited to 255 discrete steps. These can be mapped into a discrete interval [0..1]. In recent years, a new model for images has been introduced which allows dealing with much higher accuracy. It is known as the High Dynamic Range (HDR) [Aut06] mode and it provides images with up to 32 bit per channel.

Basically, the presented tools are powerful enough to define a discrete displacement map $d = I$ and to match the underling parametrization of a surface S such that to each vertex $\mathbf{v}_i$ with the corresponding texture coordinate $\mathbf{tc}_i = \mathbf{tc}(u,v)$ on the surface a pixel $\mathbf{p}_{u,v}$ of the map can be assigned.

This is how texture mapping works, where the color value for a vertex can be obtained from the image I at position u,v. This mechanism can also be 'misused' for displacement mapping, such that instead of querying for a color value, a single scalar for the offset of the vertex in its local tangent frame can be gained. In this context, a pixel can be viewed as a single value $p_{i,j} = s$.

2.3.2. Discrete Convolution

Going back to a interpretation of an image I as a discrete function, where each element $p_{i,j}$ can take a continuous value (or a set of values) of the interval $[0..1]$, it is possible to perform different operations on it. One very well known is the discrete Fourier transform, which allows the ability to state an image in terms of a set of linearly independent basis functions. This makes it possible to localize an image in both space and frequency domains. In terms of a Fourier transform, an image can be expressed as:

$$I(x,y) = \sum_i y_i g_i(x,y),$$

where y_i are the transform coefficients, which are computed from the signal by projecting onto a set of projection functions $h_i(x,y)$:

$$y_i = \sum_x \sum_y h_i(x,y) I(x,y).$$

The basis functions of the Fourier transform are sinusoids and cosinusoids of various spatial frequencies [HB95] in order to access all sub-bands separately. In order to perform the transform, a tool of choice for large domains has become the Fast Fourier Transform (FFT) algorithm.

For small domains it has turned out as useful to resort to the close relationship of the Fourier transform and the convolution operation. So it is possible to perform a sub-band decomposition of a linear, time-invariant signal without converting it into the frequency domain by convolving it by proper chosen kernel function. Since multiplication in the Fourier domain equals to convolution in the spatial domain, the spectral decomposition can be performed as a series of linear filter operations. For this reason the image function from equation 2.52 can be rewritten as a two-dimensional, discrete sum of shifted unit impulses $\delta(x,y)$. The function δ is also often referred as Dirac's delta and is defined for the bivariate case as

$$\delta(x,y) = \begin{cases} 1 & \text{if } x = 0 \wedge y = 0 \\ 0 & \text{if } x \neq 0 \vee y \neq 0 \,. \end{cases}$$

A two dimensional function can be defined as a convolution sum of the unit impulse $\delta(x,y)$ as:

$$I(x,y) = I \otimes \delta = \sum_n \sum_m I(n,m)\delta(x-n, y-m)$$

While by convolving a linear, time-invariant signal it is desired to define the convolution function (often referred as *convolution kernel* or *convolution filter*) h in terms of the unit impulse, one can perform a linear transformation $\mathcal{T}(I(x,y))$ of the image I into an unique function $K(x,y)$ in terms of the unit impulse as:

$$K(x,y) = \mathcal{T}(I(x,y)) = \mathcal{T}\left(\sum_n \sum_m I(n,m)\delta(x-n, y-m)\right) = (I \otimes h)(x,y) \,.$$

A kernel h can now be chosen in order to convolve the initial image I. For more details on digital image processing refer to Gonzales *et al.* [GW01].

2.3.3. Image Pyramids

The idea of decomposing images into 'pyramids' in order to analyze and compress them dates back to the year 1983 where Burt and Adelson [BA83], [AAB+84] introduced the Laplacian Pyramid. Their idea is rather simple: convolve the original image with a low-pass kernel and subtract the result from the source.

To do that, first a sequences of low-pass filtered images: $G_0, G_1, \ldots, G_n$ is constructed. This sequence can be generated by repeating the low-pass operation recursively and on the end the well known gaussian pyramid will be constructed. The final result of the laplacian pyramid can be reached by the consecutively subtraction of the particular low-bands at level n from the ones at level $n-1$. This produces a pyramid with steps of approximately one octave magnitude of the frequency spectrum. The sum of the decomposition steps of this pyramid results in the original image.

2.3.3.1. Gaussian Pyramid

First, the algorithm for the construction of the gaussian pyramid G_i will be formulated. It can be denoted for $0 < i < n$, with h as the gaussian filter kernel of the size 5×5 as follows:

$$G_i(x,y) = \sum_{m=1}^{5} \sum_{n=1}^{5} h(m,n) G_{i-1}(2x+m, 2y+n) \; . \tag{2.53}$$

If the image measures $2^n \times 2^n$ the pyramid will have n levels. In their original paper [BA83] they have shown that a convolution operation with a gaussian kernel leads to nearly exact low-passes of one octave magnitude. For the kernel function, some constrains have to be defined. Refer to Figure 2.29, where the weights for each ring of the kernel are denoted by a, b, and c moving from its center away.

First of all the generating kernel is chosen to be separable, which means that each axis x, y can be performed independently: $h(m,n) = \breve{h}(m)\breve{h}(n)$. Second, the one dimensional sub-kernels are symmetric such that $a = \breve{h}(0)$, $b = \breve{h}(-1) = \breve{h}(1)$ and $c = \breve{h}(-2) = \breve{h}(2)$. The third constraint is the normalization of the kernel by $a + 2b + 2c = 1$. Finally, it is demanded that each level i of the pyramid contributes the same total weight to level $i+1$ nodes. Thus, the values of $a + 2c = 2b$. Now, the value for the weight a can be chosen freely, while $b = \frac{1}{4}$ and $c = \frac{1}{4} - \frac{a}{2}$. In order to decompose a input displacement map, in this work a discrete 5×5 gaussian kernel of the form $\breve{h} = \frac{1}{20}\begin{bmatrix} 1 & 5 & 8 & 5 & 1 \end{bmatrix}$ has been used.

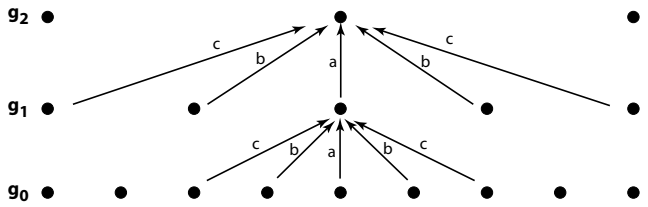


Figure 2.29.: *Representation of the process which generates a Gaussian pyramid.Note that node spacing doubles from level to level, while the same weighting pattern or 'generation kernel' is used to create all levels. Figure courtesy lean on [BA83].*

2.3.3.2. Laplacian Pyramid

Now, in order to separate the particular frequency bands, Burt and Adelson have shown that each octave equals nearly to the difference of the levels of previously generated low-pass pyramid. In order to perform a subtraction, the lower density levels have to be expanded to the same resolution as their precursors. This can be achieved by i.e. a linear interpolation (super-sampling) of each next level. Let $G_{i,j}$ denote a slice of the pyramid G on level i expanded j times. In this terms, the original image is $G_{0,0}$. Then the expansion operation can be formed as

$$G_{i,j}(x,y) = 4 \sum_{m=-2}^{2} \sum_{n=-2}^{2} G_{i,j-1}\left(\frac{2x+m}{2}, \frac{2y+n}{2}\right) . \tag{2.54}$$

In fact, the expand-operation is a reverse to the filtering operation from equation 2.53 and can also be applied recursively. Now, the difference of each two following levels i and $i+1$ can be taken as

$$L_i = G_i - G_{i+1,1} ,$$

and the sequence $L_0, L_i, \ldots, L_{n-1}$ is constructed. The last slice of the pyramid is defined as the last expanded gaussian step G_n which contains the lowest frequencies. Finally, since the pyramid L_i contains particular sub-frequencies of the original image G_0, it can be collapsed in order to retrieve G_0 as:

$$G_0 = \sum_{i=0}^{n} = L_i .$$

Images filtered in the described way contain nearly only the harmonic frequencies. An example of the first four bands of an input displacement map can be seen in Figure 4.1.

For the displacement approach developed in this work such sub-bands are supplied as a source of the surface perturbations for particular subdivision hierarchy resolutions. The details of this method will be discussed in chapter 4, section 4.1.2.

2.4. Summary

Recall that the attempted goal is multiresolution displacement on subdivision surfaces. The entire chapter has presented backgrounds which has been considered as important in order to implement the idea. For the reason of clearness this short section will summarize the presented foundations.

For the in section 2.1 presented Catmull-Clark scheme, the following can be summarized:

- The initial surfaces can be of arbitrary genus since the rules can be carried out on a mesh of arbitrary topology.
- After the first subdivision step all faces become quadrangles.
- The number of extraordinary vertices is fixed, and is equal to the number of extraordinary vertices in the mesh $\mathsf{M}^{(1)}$ produced after the first step.
- The surface approximates at its regular regions (away from irregular points) exactly the bivariate cubic B-spline and thus it is curvature smooth.
- Near the extraordinary points the surface does not possesses a closed form parametrization. Instead it consists of an infinite number of bi-cubic patches that converge to a limit point. The surface can be shown to have a well defined tangent plane at such a point, but the curvature is generally not well defined.
- After the fist step, in all following steps the face space complexity tends to $O(2^{2k})$ and the vertex space complexity to $O(2^k+1)^2$ for each of the quadrangles with k recursions.

For displacement surfaces which was the subject of section 2.2 it can be stated:

- General displacement mapping on plans or surfaces is rather trivial.
- In the case of curved surfaces it has to be taken in to account that singularities can be generated by inappropriate chosen offsets. To avoid this, the offset should be constrained by the radii of the principal curvatures.
- The offset for the displacement surface can be referenced on either intrinsic or extrinsic characteristics of the surface. The first one is independent of the global space but not of the surface parametrization instead. The second one is dependent

only on the global space and in the case of discrete meshes on the tessellation of the surface.

- In multiresolution displacement the consecutively added offsets can be scaled either proportionally or inverse proportionally to the resolution level. In the first case the surface will grow, in the second one it will develop fractal-like behavior which tends toward zero.

Finally for the displacement maps as introduced in section 2.3 it can be summarized:

- The supplied displacement maps are basically bivariate planar image functions and contain discrete height-values in the range $[0..1]$.
- By the means of discrete Fourier analysis or by convolution with proper kernels the image functions can be decomposed into their particular frequency bands such that each contains features corresponding to a harmonic frequency.

This summary should finish up the chapter of the theoretical foundations and lead to a brief overview of existing methods for subdivision surfaces evaluation and displacement techniques used in combination with the latter ones. After this, the idea of multiresolution displacement will be presented in detail in chapter 4.

3. Related Work

Mesh subdivision and displacement have been already discussed rather extensively, but independently from each other in chapter 2. This chapter will review interesting works done on fast evaluation and displacement of subdivision surfaces.

3.1. Evaluation Approaches of Subdivision Surfaces

Many researchers have focused on the problem of fast evaluation and real-time rendering of subdivision surfaces. In general, one can distinguish between four major strategies that differ very much form each other:

1. Exact analytic evaluation (Halstead and Stam [HKD93, Sta98] and more recently Zorin [ZK02]). The basics of this method has been presented in section 2.1.6.5.
2. The numerical method of forward differencing [PS96, BKS00].
3. Pre-evaluated and tabulated basis functions (developed by Bolz and Schröder [BS02, BS03]).
4. The most well known and widely used recursive evaluation (i.e. [DKT98, BLZ00], section 2.1.5). This is the method used in this work, since it is more flexible than precomputed basis functions and more stable, than the numeric method of forward differencing. In addition to that, it is naturally suited to introduce variations at each recursive evaluation level.

The evaluation methods for curves and surfaces date back to the 60's where the first techniques for raster of non-parametric but implicit defined curves haave been developed. In the late 70's and early 80's algorithms for rendering of piecewise polynomial parametric curves have been introduced. Also the already mentioned methods of Chaikin [Cha74] and Lane-Riesenfeld [LR80] can be included in this category. The actual forward differencing method dates back to the year 1967 and was introduced by Pitteway (from [LSP87]).

3.1.1. Forward Differencing

The method of forward differencing takes advantage of the fact that a spline curve or surface, which can be evaluated recursively like presented in the example of DeCastejau algorithm in section 2.1.2.2 or the B-spline evaluation method of DeBoor [DeB72], can also be determined by linear transformations of one segment of it. By properly defining the coefficients of these transformations, which are basically either scaling, translating or rotating, the segment can be mapped to all other segments of the curve. Note that this can be done without the evaluation of other branches of the recursion. Lien *et al.* [LSP87] and Chang [CR89] presented an adaptive method (AFD) for cubic curves and surfaces. Pulli & Segal [PS96] have introduced a method for evaluating subdivision surfaces in hardware by forward differencing. Additionally, they introduced the idea of a 'sliding window', where only a small memory footprint is used to evaluate a piece of mesh depth-first before 'sliding' to a next piece. More recently Bischoff, Kobbelt and Seidel have extended this method in order to evaluate Loop-subdivision on modern hardware [BKS00].

The advantage of forward differencing is its performance since it is the fastest method, while on the other hand it is numerically quite sensitive and it can evaluate only subdivision based on splines and cannot deal with extraordinary vertices. These have to be evaluated recursively. Furthermore one does not have any control over the steps in between, and because this is exactly that which makes the approach of this thesis possible, forward differencing could not have been taken into account.

3.1.2. Pre-evaluated Basis Functions

This method was firstly introduced by Bolz and Schröder in the year 2002 [BS02]. It is an interesting approach which takes advantage of the derivation of subdivision from splines. In particular their method works only with Catmull-Clark scheme but is essentially extensible to other spline based methods too. The algorithm pre-computes a set of possible combinations of B-spline basis functions for each vertex valence and stores is in a table. These pre-computations are done by the usage of the general rules as presented in section 2.1.5. During the rendering, control meshes with topologies which match the ones stored in the table can be evaluated very fast to high resolutions by multiplying the control points positions with the basis functions (as in equation 2.17). This method has been extended for hardware implementation [BS03] as well. While it is very fast, its disadvantage is of course that it is quite inflexible, since only a limited set of possible valences can be computed. Additionally, it should be mentioned that this method is quite well suited for a parallel implementation.

3.1.3. Recursive Evaluation

The main goal of most subdivision surfaces evaluation techniques is the use of recursive refinement to obtain smooth surfaces out of arbitrary polygonal input. While the rules for the recursive subdivision of Catmull-Clark are basically known since their introduction in the year 1978 [CC78], they have been researched, analyzed and refined [BLZ00, DKT98] in the over two decades since. Refer to chapter 2, section 2.1 for details on this topic. While the recursive method is basically the slowest, it is on the other hand the most flexible and stable approach. In order to improve the evaluation by the recursive method, Müller and Havemann [MH00] have introduced a versatile data structure which allows for the depth-first evaluation of the subdivision, also using a small memory footprint, inspired by Pulli *et al.* [PS96]. Their method additionally allows the ability to adaptive tessellate the meshes depending on both, object and camera-view properties. The work of Müller and Havemann has been extended by Settgast *et al.* [SMFF04], where the sliding window strategy has been adapted for recursive subdivision also. More recently, Tatiana Surazhsky [Sur07] has refined this approach in order to gain more flexible adaptive tessellations.

This work takes advantage of the methods described above and additionally combines the 'sliding window' with a hardware implementation approach similar to Shiue *et al.* [SJP05]. Their method is also based on the recursive evaluation by the well known formulas (see section 2.1.5), but in difference to the others they split the mesh not into squarish patches of B-spline size but into fragments instead, where the neighborhood is spirally wrapped around each vertex. In the actual subdivision kernel, the neighborhood can be obtained in terms the spiral ordering. Additional tags for the vertices are stored in pre-computed look-up tables for each valence. In fact, this architecture is similar to the one developed for this work, while the main difference is that it stores the mesh in long 1D textures instead of the 2D regular patterns. Also the spirally defined neighborhood requires additional tags in contrast to the implicit defined regular quad ordering used in this thesis.

3.2. Displacement of Surfaces

General displacement mapping has been already described in detail in section 2.2. It is difficult to point an origin of this technique since it is that obvious. In computer graphics it had been firstly introduced to extend Jim Blinn's bump mapping [Bli78] in order to solve the flatness-problem on the silhouettes of bump-mapped objects. Cook considered them as a type of modeling and included them into his 'shading trees' collection [Coo84]. Nevertheless, most contributions of this technique can be found in the terrain rendering domain, where it is often referred as height field mapping.

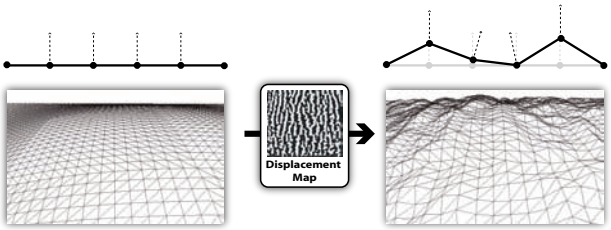


Figure 3.1.: *Mesh displacement along vertex normals.*

3.2.1. Displacement for Geometry Compression

On the other hand, it has also already been used in combination with subdivision surfaces by Lee *et al.* [LMH00] in order to achieve a new mesh representation. In their work they use a rather sparse domain which is subdivided into a highly tessellated smooth model and than the fine details are added by displacing the smooth model along its normals. The displacement fields must be previously computed from the original model, thus this method is not aimed at geometry synthesis but rather at compression in order to reduce the storage and processing time.

In a parallel work, also Guskov *et al.* [GVSS00] introduced 'normal meshes', which are a generalization of the previous approach. It allows expressing arbitrary geometric domains in terms of scalar fields, which store the entire details. Both methods are inspired by previous works, such as the work from Krishnamurthy and Levoy [KL96], which fits smooth surfaces to polygon meshes and the article of Forsay and Bartels [FB88], where they model fine features directly on B-spline patches by applying displacement. They have extended the method in a series of followed articles.

3.2.2. Displacement for Feature Editing

Displacement of subdivision surfaces has also been researched in order to manually model high detailed geometry. Khodakovsky and Schröder [KS99] have presented a method, where they definite arbitrary curves along subdivision surfaces and perform displacement along them in order to generate sharp or semi-sharp cracks and creases on smooth surfaces. This method has been extended by Biermann *et al.* [BMZB02] by enabling trimming curves on the surfaces in order to cut mesh pieces away.

3.2.3. Adaptive Tessellation of Displacement Surfaces

Another series of articles about displacement of general surfaces was coined by Gumbold, Hüttner, Dogget, Hirche, Strasser and others [GH99, DH00, BAD+01]. Their re-

search employs adaptive tessellation of highly curved regions which were obtained by surface displacement. In one of their articles they propose a hardware implementation for the method.

3.2.4. Procedural Displacement on Subdivision Surfaces

Velho *et al.* [VPYB01] introduced a way to synthesize shape features on subdivision surfaces using multiscale procedural displacement. Similar to the approach of this thesis they apply displacement to particular subdivision levels with variable scales. As a result they can synthesize several new classes of objects. They distinguish mainly two classes of objects, where on the one hand there are fractal like objects driven by offset functions inverse proportional to the object scale, on the other hand morphologic-like features with offset defined directly by the scale. Additionally they define two sub-classes of objects depending on the chosen point of reference of the displacement. This can be local (intrinsic, imposed by the surface itself), where the resulting object are 'grown' on the surface or global (imposed by the embedding), which generate volumetric-like shapes.

Also Tobler and Maierhofer *et al.* [TMW02, Mai02] have used fractal displacement to generate naturally looking tree bark. In their approach the displacement is performed on (user) specified and finite number of subdivision steps, bounded by local characteristics of the surface.

Both of these approaches apply the detail in a pre-processing step and thus do not benefit from any savings that can be achieved by procedural rendering. On the other hand these are the most related approaches to the one presented in this thesis.

3.3. Alternative Displacement Approaches

Additionally, some alternative methods for generation of fine surface deformations in order to enhance the reality of the appearance of synthetic objects should be mentioned. There is of course the classical bump mapping method from Jim Binn [Bli78] which works only by suppling perturbed normals onto smooth surfaces in order to compute the light reflections of an wrinkled surface. A more recent extension of this method is parallax mapping firstly introduced by Oliviera [OBM00]. In this method displacement of smooth surfaces is applied virtually only in the screen space. Due to the capabilities of modern graphics hardware it is possible to trace the ray from the camera through each pixel of the view port in real-time and simulate displacement near the surface by taking an offset into account in the light reflections calculations. This method is very

competitive in comparison to the presented one, nevertheless it was an intended decision to develop real displacement.

Furthermore, there are several other approaches which aim at enrichment of smooth surfaces. As an example one might mention the interesting method of shell maps [PBFJ05]. In this approach an offset surface is created and the space between the offset and the base surface is filled up with a tetrahedral mesh. Following that, the texture space is mapped into this tetrahedral structure in order to obtain many new interesting surface effects. But this approach moves too far away from the goals of this thesis presented in section 1.2. Thus it finalizes the overview of the related works, while the next chapter will go into the details of the implementation of procedural displacement on subdivision surfaces in real-time.

4. Multiresolution Displacement of Subdivision Surfaces

While previous chapters have given foundations for techniques used in this work, this chapter is dedicated to the actual main point of interest: *multiresolution displacement mapping on subdivision surfaces*. The first part will concern the term *multiresolution* and will explain the ideas developed for the realization. The second part will deal with the rather detailed description of the implementation on a programmable graphics card.

4.1. Idea Outline

In chapter 3 an overview of several approaches of displacement of subdivision surfaces have been given. The idea to apply offsets to the subdivision surface at different levels of detail is not new, however, the analogy between different frequency bands of a signal and the different subdivision levels is a novel approach.

4.1.1. Motivation

The motivation behind this idea is based on few main facts. First of all, there is an interesting close mathematical methodology behind the two approaches. Both subdivision surfaces as well as sub-bands of an image can be interpreted as a multiresolutionpresentation of a discrete signal [GSS99]. In both cases, the convolution operation can be seen as a key ingredient to derive the decomposition of each method into its particular components. Of course, there is no exact equivalence and some differences can be easily stated also. However, the similarities cannot be denied and are one of the reasons to match those fields with each other.

On the second hand, it is a well known certainty that the particular frequencies of an image often play a key role in its analysis, and further on, also synthesis. So as an interesting fact it is known, that a repeating pattern of a tillable texture depends mainly on the high frequencies, while the low frequency might be stretched over many small tiles. As a result, especially for human perception, a uniform but non-repeating field

with the same pattern can be generated. Thus, it is interesting to take this fact into account in order to create differently sized patterns on a surface.

The final reason for matching different sub-bands with different subdivision levels lies in the growing degrees of freedom of the surface characteristics. The consecutively applied displacement allows for the development of even concave surface ripples, which cannot be generated by classical methods. However, it has to be exercised with caution, since undesired effects like surface fold-overs might be generated. This chapter will explain the methods and the applied actions to ensure proper visual results used in this work.

Another issue which has significantly influenced the method described here has its origin in the particular implementation strategy on graphics hardware. Since the possibilities of custom programs on the hardware are by far not that limitless as in traditional software design, they provide a huge performance speedup on the other hand. This work was significantly concerned with such an adaption. For this reason different challenges have suddenly arisen and the implementation is often employed with resolving those problems. Section 4.2 will describe this issues in detail.

4.1.2. Multiresolution Displacement

Recalling the recursive subdivision surfaces method, it appears quite naturally to research the behavior of a surface while some 'additional' geometric information might be added into this sequential process. Since at each step one obtains compulsorily a fully parametrized and explicit mesh representation of the model, it is an easy task to perform displacement on its vertices. Doing it recursively induces a – perhaps uncontrolled – consecutive mesh distortion.

The interesting property of this combination is the fact that the resolution of the net becomes twice as dense with each step. For this reason consecutively finer details can be modeled on the net – or lets say modulated on the net – with growing density. While viewed from a far, only very coarse features are of interest, in the closeup fine ripples enhance the authenticity of the models' surface. Here arises the fact that usually if an object is inspected in a closeup view, only a limited portion of its entire shape is currently visible in the view port. So the idea behind this approach is to balance on-the-fly the actual amount of data provided to display by an adaptive generation of (1) the tessellation and (2) the fineness of detail of an object for maximizing the visual appearance but still maintaining the real-time performance.

To achieve multiresolution details it is necessary to examine given displacement texture for its coarse and fine characteristics and to separate them. As it is known from signals in the frequency domain, their globally prominent characteristics are carried by the low- and the fine traits by the high frequencies. For this purpose a sub-band decomposition

as described in section 2.3 is performed, which delivers a palette of different signal resolutions. The sum of all sub-bands returns the original image back again.

Now, applying the low-bands on a rather coarse subdivision surface resolution deforms the shape of the given object more significantly and modulates the prominent features of a displacement function. A sequential adding of more and more high frequency bands on the more and more dense net adds the desired wrinkles onto it. While the sub-bands are modulated on the surface by adding them along the surface normal on following subdivision meshes, in fact their sum is once again the original signal. But in difference to classical approaches, the resulting sample is enriched by its natural normal perturbation – which is by itself a consequence of the previous steps. One further issue related to this context is the magnitude of the offset at particular levels. Basically, there are several possibilities to define it, including the one to provide the control of it to the user. Because the aim of this work is as much of an automated procedure as possible, the offset is actually chosen by the means of the local properties of the surface. This issue will be subject of discussion in sections 4.1.2.3 and 4.1.2.4. Generally, the more dense the mesh representation, the smaller the offset that can be chosen and thus, the fine features arise less and less prominent. Additionally, this ensures that the wrinkles and ripples on the surface do not intersect each other. On the other hand, while it might be sometimes intended to generate such degenerations, the possibility to influence the offset parameters by the user is still given.

4.1.2.1. Sub-Band Displacement Maps

In the classic displacement approach the distance d of the shift is taken from an underlying texture. Since only scalar values are needed, one channel textures are enough. In this work displacement is applied similar to the classic method with the significant difference that it is done successively on several resolutions of the recursive mesh subdivision process.

Since most images supplied to displacement possess their main portion of the values in the middle of the color range, shifts along those would only lead to highlighting some extreme (as i.e. glaring) regions, while the most parts would stay undistorted. Applying such a displacement consecutively several times would result in an addition of the extremes, while details hidden in the middle gray range would stay basically unnoticed and overwhelmed. Figure 4.1, left most shows a classical displacement map and its histogram below. Note that the distribution of the values lies mainly in the middle of the range.

In order to avoid mesh growing, the offset which is normalized in range $d \in [0..1]$ can be linearly translated by $-\frac{1}{2}$ such that it affects the surface in both directions with respect to the orientation ($d \in \left[-\frac{1}{2}, \frac{1}{2}\right]$). Otherwise, the consecutively added shifts of the points

would lead to an expansion of the entire shape (blowing up) in the positive direction of the normals. By this transform the volume of the object can be preserved. Instead of just taking $-\frac{1}{2}$, it can be done more accurately by computing the mean value of the histograms for each input map separately and the shifts can be performed with respect to these. After that the zero level of a displacement map lies obviously in its gray values somewhere in the middle of $[0..1]$.

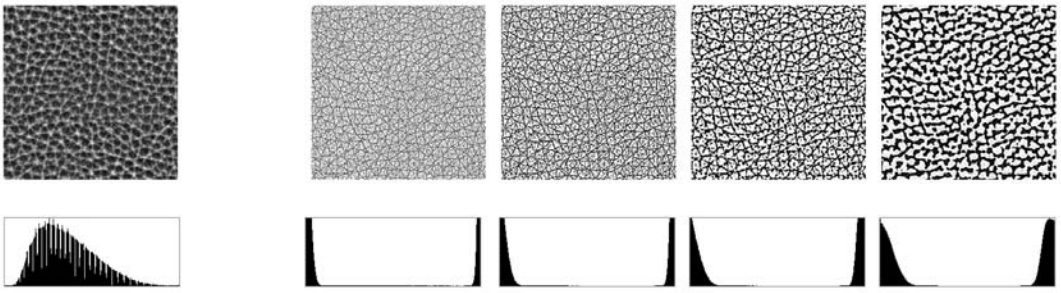


Figure 4.1.: *Left: Texture for dakota leather surface structure. Right: First four frequency bands of the displacement map in descending order extracted out of the texture. Below the associated histograms.*

In order to pluck the details of an input image and isolate them, a spectral decomposition can be performed. In section 2.3.3 the method of generating a laplacian pyramid of an image has been introduced. By performing such an analysis, a set of displacement maps with particular frequencies can be generated. with slices of one octave distance. Each level will be thus denoted as a sub-band.

The interesting properties of each sub-band are that its amplitude is again represented in the total range of $[0..1]$ and the details are clearly isolated. Figure 4.1, second left till right depicts four following sub-bands of the left most image. In the histograms below, one can see that the color values of each step lie either in the near of white or black regions of the color scale, which means that their contrast is rather high. The advantage is that the gray soup of the original signal has been decomposed and features of each size are put together to one sub-band with clear contrasts. Such displacements sources allow the addition of much more detail to the mesh than the classical ones. Figures B.4 and B.5 in appendix B depict a cylinder displaced six times with exactly the same scaling function. The difference is, that in Figure B.4 a decomposed map has been used while in Figure B.5 a simple one. As one can see, the first example is much more rich in details than the second one.

4.1.2.2. Resolution Matching

A further issue of this decomposition is the size of the features contained in each sub-band. While the high frequencies possess very fine details, the lower the band the bigger become the contained features. An interesting analogy to the subdivision procedure can be established in this context, since each texture-band differs from its precursor by exactly one power of two. This means, that an image of the size $(2^n) \times (2^n)$ can be decomposed in n different bands with growing fineness of contained features. Also the subdivision procedure refines a mesh in a power of two manner: each face is always decomposed into two in each direction, thus the density of the faces of one quad increases also by $(2^n) \times (2^n)$ and of the vertices of one quad by $(2^n + 1) \times (2^n + 1)$, where n is the subdivision level.

This analogy can be used to map particular sub-bands to corresponding sub-levels. Of course there is not much sense to use a texture with lower resolution than the mesh, because values for some vertices would be either missing or would have to be interpolated. In this implementation the maximal subdivision level in hardware has been set to 7 due to memory constraints. Including two previous software subdivision steps, an initial mesh might be subdivided up to ninth level. This would result in $(2^9 + 1)^2 = 263169$ vertices for a single input quad, which is an enormous sum and results in a vertex density which comes close to a common screen resolution. In fact there is obviously no need of displacement of the very low sampled meshes, since these should keep the main shape characteristics of an object. In practice it turned out, that displacement appears reasonable up from the third level. Than, usually up to 6 steps of subdivision with displacement appear meaningful. This results in $(2^8) \times (2^8)$ sized displacement maps if each input face is exactly covered by one tile. Because many objects are covered entirely by one texture coordinate layer only in range $[0..1]$, it turned out to be safe for most cases, to use images of the resolution 1024×1024.

The laplacian pyramid can be computed in the preprocessing and from the resulting sub-bands the six highest are taken for displacement, where the highest frequency is matched with the mesh on the sixth hardware step. This is sufficient since the very low bands do not contain any really interesting information of features in the image. For further discussion on the results of the displacement refer to section 5.

4.1.2.3. Offset Reference Approximation

In section 2.2.3 the question of a reference for the offset has been discussed. In order to provide such measures, two different approaches have been implemented. In both cases the offset is derived from the local neighborhood of every vertex. A global coordinates reference is derived in terms of roughly approximated local area measured in euclidean

space coordinates. In the other approach the supplied texture coordinates are taken in order to define a local offset reference by again approximation areas in parametric space.

In the consideration of the density of the mesh, it is not significantly important how exact the area is computed. It is of more concern that this measure is done uniformly for all vertex-types and that it in fact approximates the local neighborhood. This should ensure that the scaling happens in a natural way as given by the surface resolution.

Area. In case of quadrangles, the area can be computed exactly by:

$$A(v) = \frac{1}{2} ef \; sin\left(\frac{(\mathbf{e}_0 - \mathbf{e}_2)(\mathbf{e}_1 - \mathbf{e}_3)}{ef}\right)$$

where ef is the product of the diagonal lengths $ef = |\mathbf{e}_0 - \mathbf{e}_2|\,|\mathbf{e}_1 - \mathbf{e}_3|$. It could be be roughly approximated if one assumes that they are quite rectangular. Than the sinus term becomes just $sin(\frac{\pi}{2}) = 1$.

Nevertheless, the chosen solution goes further and approximates the area of a disc around each vertex by taking the average edge length to direct neighbors as its radius:

$$A(v) = \pi \left(\frac{1}{4}\sum_{i=0}^{3} |\mathbf{v} - \mathbf{e}_i|\right)^2 . \tag{4.1}$$

The resulting value approximates the desired local attributes adequate enough. Note that this formula can be applied on both parametric as well as euclidean coordinates. While in the first case the resulting measure is global space dependent (thus also to the objects dimensions with respect to the global basis), in the second one it depends on the supplied texture coordinates (refer to Figure 2.28 on page 56).

Both measures have their advantages and problems. The global one does not depend on user defined texture coordinates and performs in any case object. On the other hand it is sensitive to the tessellation and the objects' size. The parametric measure is independent of the latter, on the other hand it can lead to cracks if parametric space is not provided continuously (see Figure B.6 in Appendix B).

4.1.2.4. Offset Scaling Function

In the previous section an appropriate reference for global and local driven offset magnitudes have been derived. In this section the discussion will be extended to examine the procedural displacement generation. As mentioned, displacement maps applied on few (usually up to six) subdivision steps are taken from particular frequency bands of

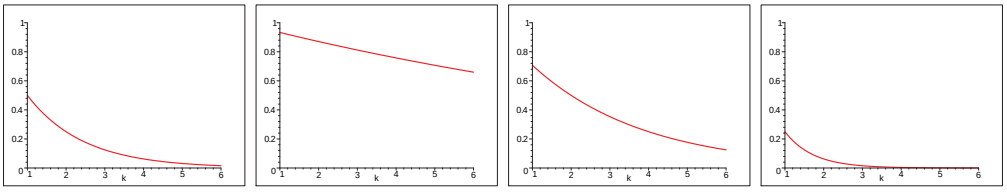


Figure 4.2.: *Scale function examples. The function $s(k)$ (equation 4.3) can be controlled by two parameters. Here A is fixed at $A = 1$. Most left: $B = 1$, second left: $B = \frac{1}{10}$. Second from the right: $B = \frac{1}{2}$, and most right: $B = 2$.*

an input signal. Thus, it is interesting to define a scaling function $s(k)$ which influences the magnitude of the displacement on certain levels k.

Recall that in terms of subdivision of regular vertices the area of incident faces of a vertex $\mathbf{v}$ is nearly quartered each subdivision step, such that it can be stated:

$$A(\mathbf{v}^k) \approx \frac{1}{2^2} A(\mathbf{v}^{k-1}) \approx \frac{1}{2^{2k}} A(\mathbf{v}^1) \,. \tag{4.2}$$

Note that this function is convergent and for $k \to \infty$ it will reach zero very fast. This is due to the fact that the area of faces vanish at the limit, such that there is an infinite number of continuous points. Scaling the offset by this means produces nice surfaces which are strongly distorted by lower frequencies of the displacement map while higher frequencies do not come into account and become smoothed out. This is because of the fast convergence of the function.

In fact, function in equation 4.2 can be used to derive a general scaling function. It is inspired by the area scaling and will be defined as an inverse proportional of the object density as

$$s(k) = \frac{1}{2^{Bk}} A \qquad A, B > 0 \text{ and } k \in \mathbb{N} \tag{4.3}$$

An interesting property of this function is that its limit is zero for $B < 1$ and A for $B \geq 1$. Furthermore, the limit sum $\sum_k^\infty s(k)$ of the curve taken at discrete k can be constrained by properly chosen parameters. In such a case the successive displacement applied even infinite number of times would finally converge to some bound defined by this sum (see Figure 4.3).

In order to change the characteristics of the function, such as flattening the curve in order to enhance the higher frequencies in higher subdivision steps, factor B in the exponent has been introduced. While k is implicitly given by the subdivision sequence, B allows one to influence it, which can be left to the user. Basically, it can also be just set to 1. The basis 2^k is given by the fact that it is the actual step from finer to coarser resolution

of the mesh in each dimension[1] (and also one step in the decomposed image). Thus, it can be seen as the frequency f of the subdivision and by taking its inverse proportional leads to fractal-similar scaling of the form $\frac{1}{f}$.

The parameter A controlls the total magnitude of the term. It can be set to a constant or be influenced by the natural area for uninterested or unexperienced users. The area can be measured either in global or parametric coordinates. On the other hand it turned out, that the specific control of its value results in a richer band of possible surface effects and this examinations will be the topic of section 5.2 in chapter 5.

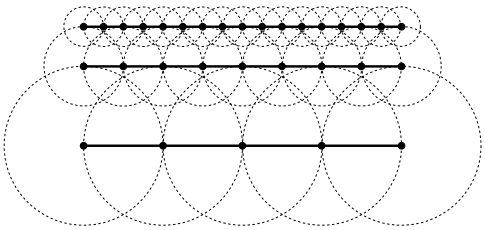


Figure 4.3.: *Displacement offset addition. If the sum of the scaling function is convergent to some constant, than the total magnitude of all displacement layers is bounded by it.*

4.1.2.5. Displacement Function

By taking the derived instruments into account the final offset D which is applied along the unit normal $\mathbf{n}$ of a vertex $\mathbf{v}$ can be defined over — basically infinite — steps of the subdivision. In practice there is only a finite set of displacement maps $d^k(u,v)$ because of the finite resolution of the input samples. Nevertheless, the final displacement offset can be stated by the following equation

$$D^k(v) = \min(s(k)\, d^k(v),\, R(v)) \;, \tag{4.4}$$

where $s(k)$ is the scaling function from equation 4.3 and R is the radius of the curvature. The minimum is taken in order to avoid that the offset exceeds the curvature radius independently of the scaling. The radius can be computed as reciprocal of the of the curvature (refer to equation 2.51) as

$$R(\mathbf{v}) = \kappa_K^{-1}(\mathbf{v}) \;.$$

Finally, the displacement is performed on each vertex according to equation 2.43 on page 47 by

[1] Note that the dimensions of $\mathbb{R}^3$ as well as of the tensor product $\mathbb{P} \times \mathbb{P}$ are linearly independent.

$$\hat{\mathbf{v}}^k = \mathbf{v}^k + D^k \, \mathbf{n}^k \, . \tag{4.5}$$

Implementation Limitations. In the practice it turned out that the computation of the local area has involved unexpected problems in the hardware implementation. This happened mainly due to the limitations of the graphics hardware such that the temporary registers count has been exceeded by adding the curvature computation to the routine. Due to this, in the implementation the scaling is appended only on the local area. Nonetheless, an additional shader program has been implemented in order at least test the constrained displacement. In the additional program the overall functionality of the presented method has been reduced to a minimum and the achieved result is presented in Figure B.12.

4.2. Implementation

In todays computer graphics it has become a challenge to implement powerful algorithms for the rather limited programmable graphics hardware. For geometry processing it also means that essential properties like mesh connectivity have to be stored or calculated in different manners.

In this section the implementation of the hardware-subdivision and displacement method will be described. First of all, a short introduction into the GPU programming and the used extensions will be given, followed by an overview of the software half-edge data structure underlying the shader core. Further on, the subdivision-kernel will be explained in detail followed by the descriptions of the displacement implementation. Finally, the adaptive subdivision approach developed in order to reduce the computational complexity will be addressed.

4.2.1. GPU Programming

Modern graphics architectures have become flexible as well as powerful. Once fixed-function pipelines capable of outputting only 8-bit-per-channel color values, modern GPUs include fully programmable processing units that support vectorized floating-point operations at full IEEE single precision. High level languages have emerged to support the new programmability of the vertex and pixel pipelines. Furthermore, additional levels of programmability are emerging with every major generation of GPU (roughly every 18 months). Examples of major architectural changes in the current generation (as of this writing) GPUs include vertex texture access and full branching support in the vertex and fragment pipeline. The currently incoming generation is expected to add 'geometry shaders', or programmable primitive assembly, bringing flexibility to an entirely new stage in the pipeline. In short, the raw speed, increased precision, and rapidly expanding programmability of the hardware make it an attractive platform for general-purpose computation. Courtesy of [OLG+04].

4.2.1.1. Render to Vertex Buffer

One of the extensions in the current Shader Model 3.0 (SM 3.0) specification is the possibility to fetch texture values in the vertex shader. This also includes access to textures which have been defined as render targets and in this way a data flow loop can be closed directly in video memory without usage of the CPU. A further extension, which is unfortunately not clearly specified yet, such that the graphics hardware vendors implement the usage of it differently in their drivers, is the possibility to redefine a texture directly as a vertex buffer in video memory. Refer to Figure 4.4, where the

classical fixed pipeline (left) is compared with the flexible pipeline from last generation (SM 2.0, middle) and with the current extensions of SM 3.0 (right).

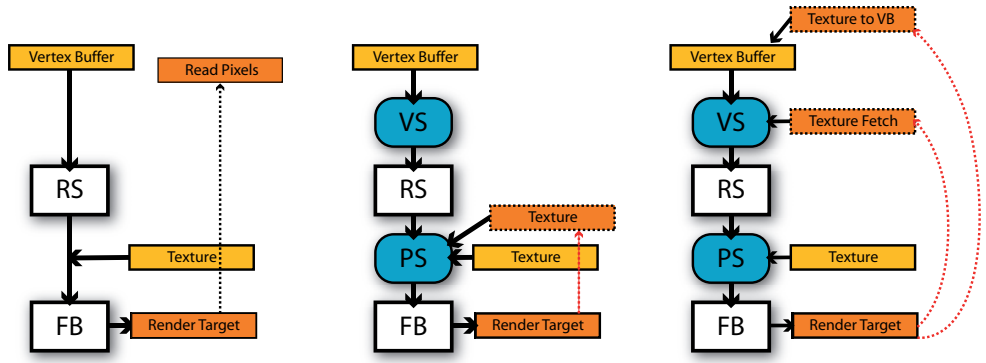


Figure 4.4.: *Render pipelines comparison. Left: classic fixed pipeline. Middle: programmable pipeline with the possibility to render to texture. Right: flexible pipeline with the possibility to access the render target in pixel and vertex shader.*

It should be considered that current hardware is able to render millions of triangles in absolute interactive frame rates. The actual bottleneck of the pipeline is the bus between system and video memory though which geometric data is delivered for rendering. In order to overcome this drawback, the approach developed for this thesis transfers a rather sparse geometric data into the video memory and computes the actually desired resolutions on-the-fly by the usage of the programmable pixel shader. In the shader a custom program performs the Catmull-Clark subdivision procedure and applies displacement if desired. The surface normals are also computed in the shader. All this happens by rendering new mesh resolutions consecutively into a fixed set of textures. Finally, the texture with the desired density is reinterpreted as a vertex buffer and passed for displaying. In order do define polygons in this vertex buffer, a fixed and previously calculated index buffer is also supplied. This index buffer provides a scheme how to join particular vertices into triangles for rendering and does not depend on the actual mesh topology. Moreover, since the pattern of every vertex buffer texture always has the same layout, the same index buffer can be reused.

4.2.1.2. Sliding Window

To efficiently perform the subdivision process on current hardware, it is necessary to regularize the input data structure of the algorithm. As mentioned in section 3, Pulli and Segal [PS96] have introduced a rendering method, which they called 'sliding window' in order to reduce the memory consumption of highly tessellated meshes. While their implementation differs in several ways from this one, in one essential issue it is lean on the same idea.

More precisely, in this approach only a fix-sized memory footprint is used to subdivide, displace and render basically unbounded large meshes. This memory portion, composed of a set of textures is the 'sliding window', which is successively moved along the initial mesh. Since the mesh can be decomposed in always exact the same, small and squared patches defined by the input quadrangles, the window can be consecutively filled with input data, all desired operations can be performed within it including output to the frame buffer and it can be deflated again, before moving to the next piece of the mesh. This sliding operation has to be performed for all – in best case all visible – mesh patches in one frame.

Since extraordinary vertices cannot be fitted into this scheme, it is allowed that each patch contain one of these at most, together with its 1-ring neighborhood (Figure 4.12) stored in an additional 1D texture. Because the number of extraordinary vertices is constant after the first subdivision step and they can be isolated by one more traditional subdivision step in software, the algorithm can be prepared to deal with this case.

The GPU 'sliding window' takes each of these patches and recursively subdivides and displaces it down to the desired level. Since there is no possibility to establish a 'physical' connection in memory between particular pieces, an overlapping neighborhood is included in order to maintain continuity across patches. For subdivision, as explained in section 2.1.5 only one ring of overlapping vertices is sufficient to ensure coherence (see Figure 4.8). The details of this issue will be presented in sections 4.2.2 and 4.2.3.

4.2.2. Data Structures

In order to facilitate subdivision on the shader, several conditions must be fulfilled by the provided data structures. Only a good management of the resources will allow to benefit from the advantages of the computational power of the GPU without creating a new bottleneck in the render pipeline. As mentioned before, the bus band width between the CPU and GPU is often such a bottleneck.

The challenges to the data structures are that on the one hand, to provide the whole mesh data fast enough to the GPU, on the other hand to encode all needed information into it. As described in previous chapter, a critical requirement for recursive subdivision algorithm is the knowledge of the neighborhood of each particular vertex.

To fulfill the stated condition several important changes in the mesh structures have to be taken. The subdivision procedure can be stated briefly in the following four basic steps, where the first one is performed once and off-line:

1. Split the mesh into small regular squared mesh pieces, such that each vertex is a regular one (valence of 4). This arrangement has an implicit connectivity information, which can be easily obtained.

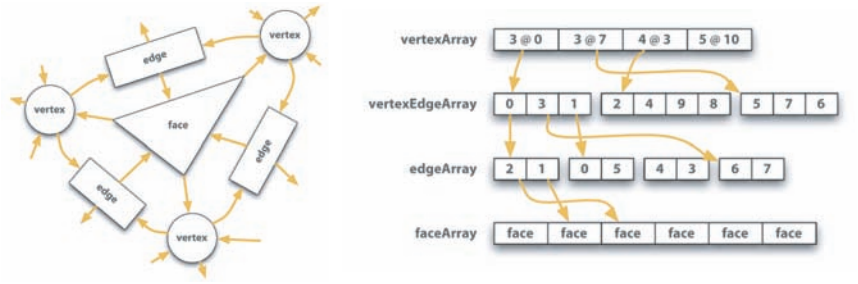


Figure 4.5.: *Half-edge data structure layout. Left: General structure of topological information in the half-edge data structure. Right: Memory layout of the topological information. Figure courtesy of [TM06].*

2. Subdivide each piece independently under the restriction to maintain exactly the same border as in neighbor pieces. Since a physical connection between the patches is not given, complete the lacking information through multiple data storage.
3. Allow at most one extraordinary vertex per piece and introduce a way to handle it.
4. Subdivide (and displace) all patches depth-first sequentiality to a certain level in one frame.

4.2.2.1. Software Half-Edge Data Structure

The mesh-partitioning operation has to be done in software once, before the actual real-time rendering can be performed. The software half-edge data structure contains exact neighborhood information, such that it is easy to obtain adjacent quads and their properties.

The underlying half-edge data structure is based on a number of index-arrays, which hold unique concatenations to each other. The structures are part of the rendering environment 'Aardvark', which is an internal development of the VRVis company. A more detailed description of the mesh handling has been documented by Tobler *et al.* [TM06]. The topological connections in the mesh are depicted in Figure 4.5. Through a set of interfaces on a higher level it is possible to access all needed information in the index arrays.

For the case of pure mesh subdivision, the mesh structure provides an interface to gain exactly a particular quadrangle by specifying its index. As a result of this query, one can obtain the indices of its four vertices, edges, texture coordinates etc. and also indicies to

its neighboring quads. These again, can be queried for their own properties (see Figure 4.6).

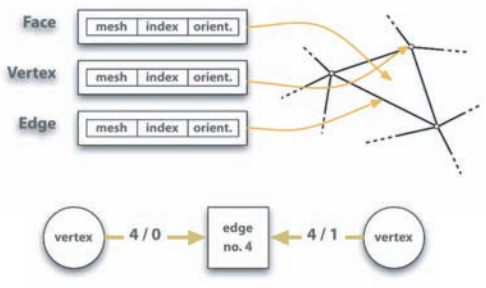


Figure 4.6.: *Higher level interfaces in the data structure. Top: Each element of the mesh can be encapsulated on the fly by a interface. Bottom: The orientation of the elements is also stored. Figure courtesy of [TM06].*

4.2.2.2. Mesh Patches Data Structure

The software structures can be used for traditional subdivision as well as for preparing the mesh for the hardware. By traversing sequentially the array of quadrangles, the mesh is split into overlapping regular patches, which are squared and have implicit neighborhood relations. This is perfectly suited for the Catmull-Clark scheme for quadrangle meshes. Each quad is encoded into a small, 4x4 pixel sized geometry texture including its neighbors. Though the adjacency information retrieved from the half-edge structure, all neighbors of each quad can be unambiguously determined and encoded into the same texture. For the case where the mesh piece contains an extraordinary vertex, an additional, one dimensional texture is used, which contains it and its neighboring vertices arranged spirally, as shown in Figure 4.12. For the cases of the actual borders and corners of the entire mesh two additional layouts are also provided (refer to Figure 4.8).

In fact, there is a lot of redundancy, since each of the patches overlaps with all its neighbors and only the middle 2 by 2 vertices are displayed. Compare Figure 4.8 which shows the layouts for the regular, border and corner patches. This results in an overhead of approx. 800% (for a exactly regular infinite mesh, since each patch is covered by its own quad and additionaly by the 8 neighboring quads) more stored vertices than in the original mesh, which finally still benefits in a performce improvement.

The reason for storing this big amount of redundant data lies in the nature of the subdivision surfaces scheme and the limitation given on graphics hardware. Since every patch is processed independently from the others and no connection to its neighbors is provided, and to compute proper position of the subsequent mesh vertices the local

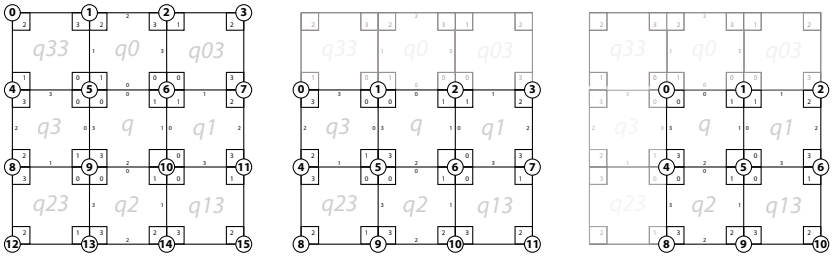


Figure 4.7.: *The three layouts for storing a mesh in a texture. Left for the entire patch, middle for a border case and right for a corner.Bordes and corners can be covered by this three schemes by rotation.*

neighborhood of the parent vertex is needed, one has to store it in each patch explicitly. In fact, only one vertex ring around the patch is sufficient to obtain proper geometric information, so the relative size of the redundant data decreases with each subdivision level. This can be seen in Figure 4.8 on the left hand side, where only the black marked vertices are necessary for the next subdivision step.

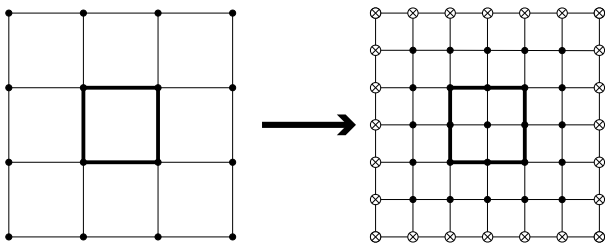


Figure 4.8.: *Subdivision step of a patch processed in the 'sliding window'. The bold marked vertices are actually the necessary neighborhood for subdivision. Vertices marked with $\otimes$ are not given any more. For final scene rendering, only the thick marked patch region is used.*

Geometry-Textures. This small mesh patches can be encoded as tiny geometry textures such that their format is adequate for the GPU processing. For each 4 by 4 vertices two four-channel HDR float textures (High Dynamic Range [Aut06]) are created: one containing the vertex position vector $\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z & tc_u \end{bmatrix}^T$ and the texture coordinate $\mathbf{tc}_u$ in the four *RGBA* channels and one containing the vertex' normal vector $\mathbf{n} = \begin{bmatrix} n_x & n_y & n_z & tc_v \end{bmatrix}^T$ and the $\mathbf{tc}_v$ texture coordinate respectively. The initial patch texture-layout can be seen in Figure 4.9. High Dynamic Range has been used to provide maximum numerical accuracy – in this case there are 32 bit per channel.

To locate the initial data in video memory several methods are possible:

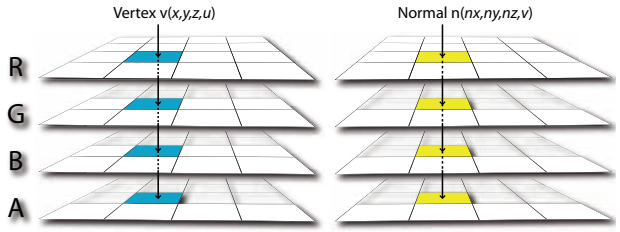


Figure 4.9.: *Layout of the geometry-textures. Each vertex and normal is stored as one pixel with its euclidean coordinates as RGB values. The alpha-channels in both textures are used to store the texture coordinates u, v.*

1. to store the patches in a an array an bind the particular one before subdivision
2. to store all patches in a texture atlas and provide proper texture coordinates
3. to store all patches in a 3D texture and provide proper depth-coordinate

In this work the whole patchwork is stored in an texture array and can be sequentially traversed during the rendering. Actually, this is not the optimal solution in consideration of the hardware architecture because it requires pipeline context switches after each patch is finished.

Alternatively, the texture-array may be kept as one three-dimensional texture, where the additional depth-coordinate could serve as an index. In praxis this solution turned out to be much more error-prone and difficult to implement. Especially the access to particular volume-texture slices was instable and the additional complexity of the shader program led to the limits of the variable registers of the graphics unit. Therefore, the more robust version with a software texture-array has been finally favored.

Additionally, the second and third solution suffer on the hardware texture size limitations (SM 3.0: 4096^2 for 2D and 1024^3 for 3D–texture), where in the first one basically unlimited number of patches is possible (limited through video memory). Their advantage is that no explicit texture binding is needed anymore, but only the specification of proper coordinates instead.

4.2.3. Subdivision Shader Architecture

In section 4.2.1 the strategy of 'sliding window' have been mentioned. In this approach only a small and fixed sized memory portion is used to render a whole object by consecutively filling, rendering and sliding it to the next position. It seems obvious, that this way of processing is quite suitable for an implementation in hardware [PS96]. In this work this strategy has been adapted for the implementation of the Catmull-Clark kernel. Particularly there is a fixed set of textures located in hardware memory on which

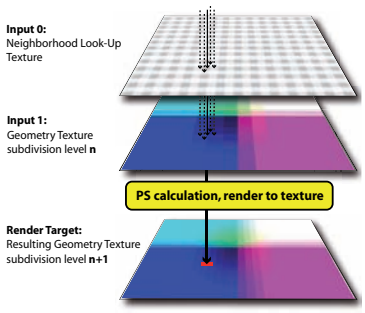


Figure 4.10.: *Pixel Shader. By means of the chess-board lookup texture, each type (face-, edge-, vertex-) of a point can be determined and a proper branch in the shader program can be chosen. The result is rendered into a next level texture (see Figure 4.17).*

the subdivision kernel is executed in the pixel shader stage of the GPU. Each next of the textures has a higher resolution in order to keep the upsampled vertex positions. The calculated results are rendered sequentially depth-first up to the desired level into next following texture, which can be either directly reinterpreted as a vertex buffer and provided to the final rendering pipeline without usage of the CPU, or again reused as a input for the next subdivision step. This pipeline is depicted in Figures 4.10 and 4.17.

4.2.3.1. Lookup Texture

The architecture of the shader is based on two new properties of current hardware: rendering to vertex buffer and dynamic branching in the pixel processing stage as described in section 4.2.1. The data structure allows one to uniquely identify each vertex and its neighborhood stored in the texture naturally due to its exactly regular pattern.

According to this pattern one can benefit from it, by suppling a reusable look-up textures set for the subdivision process. For each subdivision level such a texture with proper resolution can be generated off-line priorly to real-time processing. This texture is a squared chess-board-like grid, which matches exactly with the desired geometry-texture in the render target, thus the same set can be reused for all patches with the same layout. Since there are three different types of patches (Figure 4.7), three sets of look up textures are needed.

During the rendering [2] in the inner subdivision kernel (refer to Figures 4.16 and 4.17),the geometry information from each previous level is used as input data in such a way

[2]Note, that the term *rendering* is often used also for the subdivision and displacement operations, because the output data is obtained by rendering into an internal render target just the same way as rendering to frame buffer.

that the texture is point-interpolated over the look-up grid, as pictured in Figure 4.10. Because of the point-interpolation, the pixels from the previous level become nearly twice as large and cover a rectangular region on the current resolution like a stamp. By properly choosing a distance measure in the render targets parametric space one can access all upsampled texels from the previous level and obtain their values (coordinates) in order to compute the current level.

The lookup texture holds all needed information. First of all it holds in its alpha-channel the relations of the particular points in mesh. Figure 4.11 shows the repeating pattern of the vertices. Here each vertex type can be determined by an unique key.

Another channel (blue) holds the position of a potential extraordinary vertex and its neighborhood. If the shader program reaches the marked fragment, another branch of the algorithm is accessed and the positions of the vertex and its neighborhood are overwritten.

The green channel holds tags for the border of visible area of a patch. According to this region, borders to adjacent patches can be adjusted.

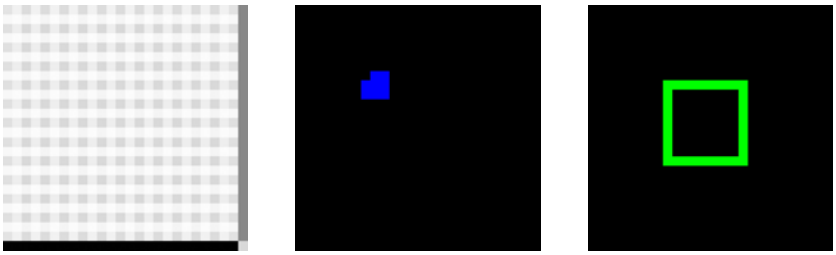


Figure 4.11.: *Lookup Table layout. Left: chess-like grid holds types of the points. Middle: blue channel marks the position of a potential irregular vertex. Right: green channel holds a mark of the visible region in order to adjust borders with neighboring patches.*

Information about a vertex type needed in the shader can be obtained at each fragment by picking the unique key stored in the lookup texture. By means of this key, a dynamic branch in the shader program is chosen to perform the particular operation on the vertex by taking its neighborhood into account. The result is rendered on the current position into the target pixel.

In current implementation, the shader distinguishes between nine different branches for the regular patch subdivision, depending on the properties of the vertex. This can be seen in the Listing below.

Listing 4.1: *Shader Subdivision Program*

```
1 function SUBDIVIDE
2         set vertex:=0
3         get unique key value from lookup texture texel
4         switch unique key
```

```
5                   case F1:
6                           vertex := compute face point
7                   case E1:
8                           vertex := compute vertical edge point
9                   case E2:
10                          vertex := compute horizontal edge point
11                  case E3:
12                          vertex := compute vertical border edge point
13                  case E4:
14                          vertex := compute horizontal border edge point
15                  case V1:
16                          vertex := compute vertex point
17                  case V2:
18                          vertex := compute vertical border vertex point
19                  case V3:
20                          vertex := compute horizontal border vertex point
21                  case V4:
22                          vertex := compute corner vertex point
23          end switch
24          return vertex
25  end function
```

All of the switch-cases can be accessed dynamically on the current subdivision hardware, such that unconcerned branches are not evaluated. For the computation of the particular vertices the general formulas as derived in section 2.1.5 have been implemented.

4.2.3.2. Extraordinary Vertices

Unfortunately there is no possibility to simply include a vertex of irregular valence into the regular pattern. To overcome this problem a different strategy has been chosen. Firstly, all extraordinary vertices have to be separated from each other by at least one ring of regular neighbors. This can be done very easily by just applying not less than two subdivision steps in software. Since after the first step no more new irregulars can pop up in the mesh anymore and the second step separates them, it is sure for all constellations. Thus, each patch contains at most one such irregular vertex and furthermore – by properly choosing the orientation of the patch – it is always located at exactly the same position. This fact can be used to encode that position into the look up texture, such that during processing of the fragments a hit on this position can be determined. Now, the shader program can jump into an other branch and recall the proper position of this vertex from a different source texture. It should be mentioned that not only the extraordinary by itself is affected but also its first order neighborhood. Thus the marks in the look up texture cover these vertices and they become overwritten as well. The situation around an extraordinary vertex is depicted in Figure 4.12.

But where does the proper position of the particular irregular vertex come from? This can be solved by extending the data structures, such that for each mesh-patch a one-dimensional texture for the extraordinary vertex with its neighbors arranged cyclically

around it, is stored additionally. This separation also happens in the software mesh split operation and the 1D textures are stored in texture memory for each patch. If the outer loop (see Figure 4.16) encounters a mesh patch with such a situation, an additional render pass is called in order to determine the proper position of this vertex and its first-order neighbors. This render pass can be performed exactly by the same means as the one described above: in the ping-pong manner, where after each subdivision step one texture serves as the source and another as the destination with both changing after each following iteration. Since the size of the texture does not depend on the subdivision level, but only on the vertex valence instead, for each valence there must only be two of those allocated.

The extra pass has to be performed priorly to the regular subdivision, such that the irregular vertex and its neighbors can be obtained from the 1D texture and become overwritten in the regular patch routine. Figure 4.12 shows the arrangement of the vertices in this case and Figure 4.11, right hand side (and also the color Figure B.24 in appendix B) the mark in the blue channel of the look up texture. Figure 4.17 depicts the entire shader architecture with its particular render passes.

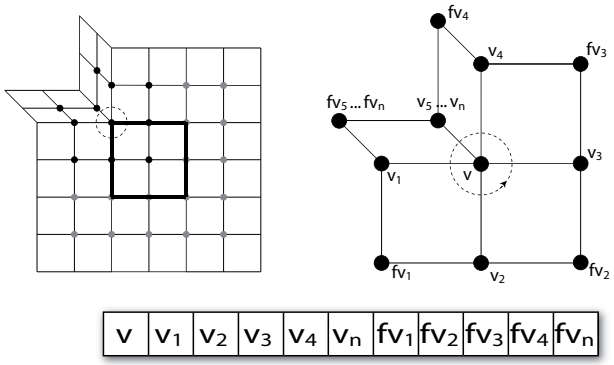


Figure 4.12.: *Handling an extraordinary vertex. Left: patch with extraordinary vertex. Right: Spirally ordered neighbors around a extraordinary vertex of valence n stored in 1-D array.*

Furthermore, note that due to hardware resources shortness the extraordinary algorithm cannot be implemented in a generic program. This has been resolved by providing a software routine which generates specific shader programs for each particular valence. This routine can be called once or each time in the preprocessing stage in order to create proper shaders. During the runtime these are executed as particular passes of the hardware and thus a context switch does not have to be performed.

4.2.3.3. Parametrization Continuity across Patches

Right now the subdivision of the quads of the mesh has been discussed. It should not be omitted that the parametrization of the net has to be subdivided also in order to obtain proper texture coordinates on each level. This is important for future texture mapping as well as for the displacement mapping in hardware, because the offset is taken from a texture by the means of the vertex' texture coordinates.

Due to the natural parametrization of subdivision as described in section 2.1.6 at each step texture coordinates can be computed by linearly interpolating its forefathers very easily. This can be done over adjacent patches in the same way as in the interior of each one. Problems may arise if the coordinates are defined over more adjoint $[0..1]$ intervals, which is usually quite common. In this cases the overlapped regions would obtain adulterated (u,v) values because the vertices on the common border possess the value 0 in the first patch and 1 in the second. The interpolated values in the first overlapped ring would not match to each other. This can be resolved by joining all intervals into a global one $[0..n]$ with $n \in \mathbb{N}$ over all patches. This is possible because of the texture repetition in the hardware (where values between $[0..1]$ are interpreted exactly the same as between $[n...n+1]$) and the interpolated values become identical in both adjacent mesh pieces.

4.2.4. Displacement Mapping

Previous sections have explained how subdivision surfaces can be implemented for graphics hardware. The chosen method ensures completely smooth subdivision across the mesh pieces and performs very fast and stable. Now the smooth surfaces should be enriched by the other essential ingredient of this work: displacement mapping on each particular subdivision level. As the source of the displacement, a set of individual frequencies of an input map are applied on the particular level.

The theoretically correct ideal displacement mapping as shown in Figure 2.22 requires the computation of the surface normal of the smooth surface as described in section 2.2. The computation of the limit normal of a fixed vertex is not an expensive task if the first-ring neighborhood is known. The resulting normals are the same on each subdivision level. Along these normals a shift of each vertex by an appropriate displacement offset can be performed. The offset is taken from the appropriate sub-band of the decomposition of the input displacement map and scaled by rules as presented in section 4.1.2.5. This kind of displacement leads to modified normal vectors in the next level of the subdivision process.

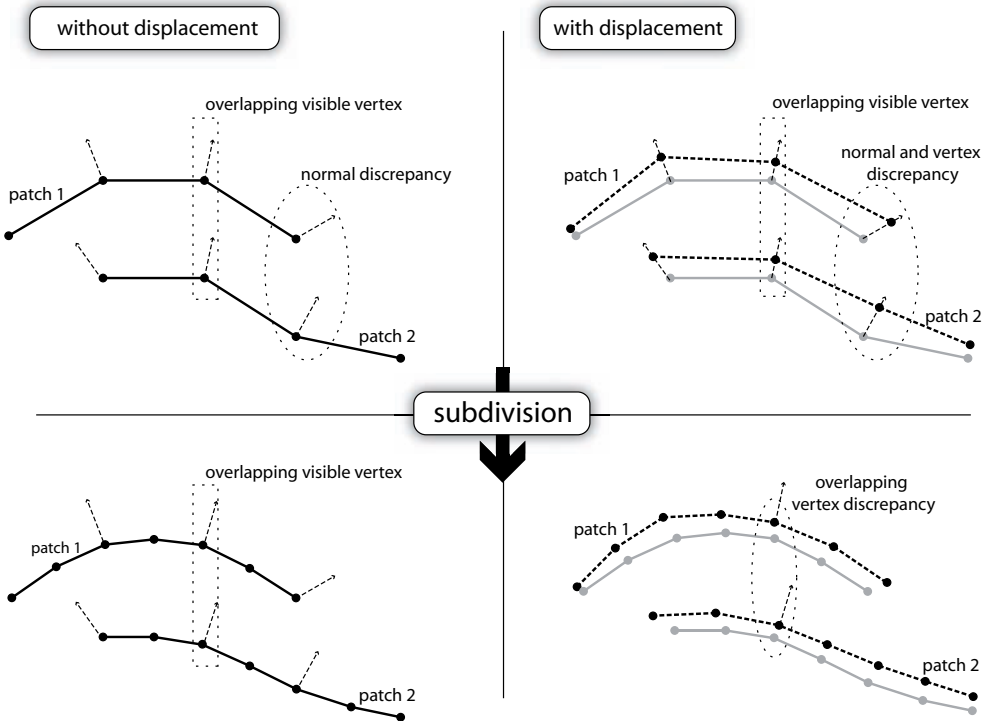


Figure 4.13.: *Subdivision (left) and displacement (right) applied on two overlapping patches. Right: since the overlap-normals of the patches are not consistent, the displaced vertices will be neither (see section 4.2.4.1).*

Unfortunately this is not as strait-forward in the GPU implementation because of the lack of matching first-ring neighborhoods of some vertices. This section will discuss this problems and solutions chosen in order to implement this technique on the GPU.

4.2.4.1. Displacement Normals Estimation

The consistency between two overlapping patches is easily ensured for the Catmull-Clark subdivision by maintaining the 1-ring-overlap as described before. However, the introduction of displacements at each subdivision level leads to an additional problem on the visible boundary of a patch.

If every normal is computed in its local tangent frame the positions of all 1-ring neighboring vertices have to be maintained identically. Note that the exact normal evaluation requires the first-ring neighborhood around a vertex as well (see section 2.1.6.5). However, the normal of the first invisible row of vertices of a patch differs from its counterpart in the neighbor-patch due to the lack of further information behind the first-ring. If displacement is performed, this discrepancy is also propagated to the first-ring vertices

behind the visible patch border and thus it produces a gap between adjacent patches in the next subdivision step (see Figure B.22 in Appendix B). The root of the problem is shown if Figure 4.13.

There are two possible remedies: it is possible to ensure consistency if the 2-ring neighborhood around each patch is stored and computed. Since this represents a significant computational and above all a huge storage overhead, this work resorts to the second possible solution: to use estimated normals for displacement before the subdivision.

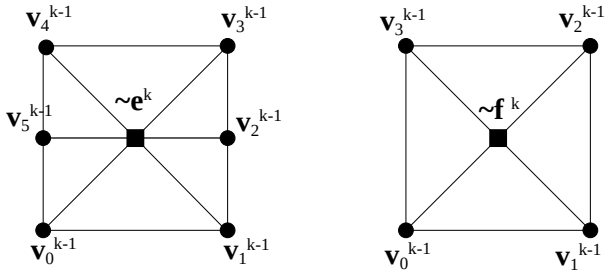


Figure 4.14.: *Normal estimations schemes. Left: an edge-point $\tilde{\mathbf{e}}^k$, right: face-point $\tilde{\mathbf{f}}^k$. Face point is bi-linearly interpolated by the 1-ring and edge-point is just set on the bisector. Than the normal is computed as normalized sum of the cross product normals of the resulting faces around.*

Of course, a rather good estimation of normals can be done only on vertex-points of the current subdivision level, because of the presence of the vertex and its 1-ring. If subdivision has not been performed, new face- and edge-vertices are not given yet. So the straight-forward solution is to temporarily estimate the missing edge- and face-points in a rather rough way. Each particular point $\tilde{\mathbf{p}}^k$ (which will be introduced in the just after following subdivision step as $\mathbf{p}^k$) is linearly interpolated by its surrounding points at level $k-1$. Than its normal is computed as a normalized sum of the cross product normals of the faces given by the interpolated point $\tilde{\mathbf{p}}^k$ and its neighbors $\mathbf{p}_i^{k-1}$ (see Figure 4.14).

Here one may ask, why to compute several normals of the same plane? There are two answers: (1) indeed not all quadrangles of the Catmull-Clark are totally planar and (2) it turnd out, that if the normals are computed by the means of just one face, small numerical differences again cause discrepancies across patches and gaps to appear in the mesh. The chosen solution closes the mesh totally 'watertight'.

It has an impact on the correctness of the displacement indeed, but in consideration of the density of the mesh it is negligible small and no noticeable drawback can be observed on the visual results. Finally, this solution provides a further benefit: because the estimation can be done for each vertex without knowledge of the properly placed

neighbor points, it can be performed just-in-time directly before the subdivision in the same shader program, which saves an expensive switch of render targets in order to compute proper normals. Exact limit normals can be computed after the last subdivision step and can be used for the illumination of the final scene.

4.2.4.2. Local Area Estimation

Besides consistency of the normals, the displacement offset has to be kept equally in adjacent patches too. This issue is very actual in context of displacement along perturbed vertex normals.

Similar to the normals-problem, by deriving a local offset magnitude from the incident faces of some particular vertex requires its equal neighborhood as well. Here again the problem arises: the neighborhood around the first invisible overlapped ring is no longer given away from the center of a patch. Refer to Figure 4.8, right hand side, where the with $\otimes$ imposed vertices cannot be kept equal anymore. Thus, the surrounding area is not the same as in the adjacent patch. Referencing the offset of those values would again cause small gaps between the patches, which would grow with any next subdivision step (Figure B.22 in Appendix B shows this example).

In order to resolve this problem local area can be estimated similarly as the surface normal. The actual, not yet existing vertices of next subdivision step are interpolated as presented in Figure 4.14. Than an average of the edge length of the vertices is taken in order to derive a local disc radius. The distances here can be taken either in global (x,y,z) or in local (u,v) coordinates. As discussed in section 4.1.2.3 it is not necessary to provide an exact area of the faces, but rather a proper measure over all vertices which can be held equally over adjacent patches.

Having estimated all needed local properties subdivision can be performed as usual. The displacement can be than applied just after the subdivision, particularly after the new vertex position has been computed. Since, as mentioned, estimated values from the previous step are used, this procedure can be done in the same shader program, without any context switch.

4.2.4.3. Displacement Map Continuity

In order to perform displacement on overlapping patches, one final issue should be taken into account. While a continuous texture coordinates layer can be kept over adjacent pieces, texture tiles can cause holes on common borders if the contained offset values d at the borders of the textures do not match each other. Such differences would not lead to gaps in coherent mesh because of a consistent index buffer. It would rather be noticeable as a kink on the surface. On patch borders it leads to cracks since there is no

connection of neighboring faces given, not even by the index buffer. This problem can be resolved by providing exactly tillable textures as displacement maps only. In such periodically repeating patterns each next border is a continuation of a previous tile and the offset can be held coherent in both patches.

4.2.5. Adaptive Subdivision and Displacement

While the last section has introduced the way of how displacement is performed, this section will concern the issue of reducing the scene complexity in order to improve the real-time performance.

Since in the decomposed mesh each patch is independent, so each can be subdivided and displaced to a certain level (see section 4.2.2). This constellation implies the possibility of a step-wise adaptive subdivision of the whole model, than different patches can be rendered at different resolutions depending on the current camera position. One can profit from this issue and allow an one-step difference between neighboring patches without loss of the geometric coherence of the mesh. Vertices of mesh patches at different levels do not lie on the same positions, however the last positions at the higher sampled mesh are known exactly.

So as a straight-forward solution, the border vertices between two differently sampled meshes can be forced to their positions at the lower subdivision level. Since in the higher-level mesh newly introduced edge-vertices do not have any forefathers, they can be placed exactly at the midpoints of the edges. This produces T–triangles and indeed, during the rendering small micro-holes in the surface can still appear at the T–junctions. This issue can be enforced by introducing zero–area–triangles at the T–junctions similar to those proposed by Losasso *et al.* [LH04].

The described procedure behaves very well on regular patch junctions, while it is difficult to implement in hardware for extraordinary vertices. Fortunately, the number of these is usually very limited in a mesh, such that the method does not allow different resolutions on adjacent patches sharing one extraordinary point in order to avoid those hard cases.

In fact - this solution is numerically safe and provides a completely 'watertight' mesh.

4.2.5.1. Resolution Estimation

Before subdivision, at the beginning of every frame, a fast routine estimates the visibility of each patch based on the current view position. This estimate is conservative, using a simple normal cone method for determining if the current patch is facing away from the camera, and can therefore be excluded from the GPU rendering pipeline. The normal is

computed for each patch at its original subdivision level and stored permanently. Than the angle ϕ between the patch normal vector and view port plane unite normal vector can be computed. According to this angle it can be decided if the the patch is facing to the camera or not. If $\phi > \frac{1}{2}\pi$ than the patch normal is facing away. In order to preserve the silhouette of the object, the critical angle can be set to a higher value (i.e. $\frac{3}{5}\pi$). This is implemented statically but it delivers sufficient results.

The routine also calculates the subdivision level for each visible patch based on the size of its screen space projection. A statically defined table is used to convert this projected size into the required subdivision level. In order to avoid holes, it is required that the subdivision level of neighboring patches may only differ by one level. This requirement is fulfilled by recursively correcting neighboring subdivision levels in case of a violation.

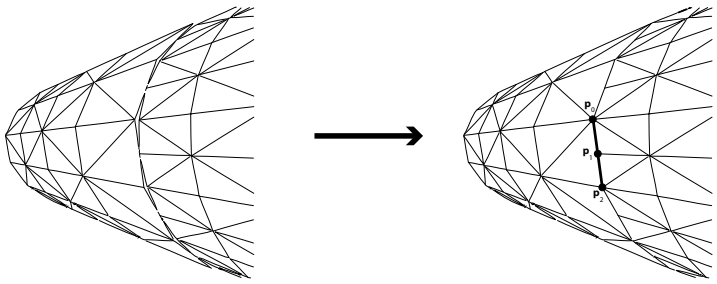


Figure 4.15.: *Closing a T-Joint between mesh patches. Left: mesh with two patches of different resolution. Right: the vertices of the higher sampled mesh are forced to the lower level. Points* $\boldsymbol{p}_0$,$\boldsymbol{p}_1$,$\boldsymbol{p}_2$ *from a zero area triangle.*

4.2.5.2. Patch Borders Adjustment

If during the subdivision the desired level of a particular patch is reached, depending on its neighbor patches conditions the border vertices have to be adjusted. This is again performed in the same shader program by simple skipping of the border vertex-points and interpolating the positions for edge-points in only the middle of each edge. To determine the affected points in the geometry texture, the green channel of the look-up texture holds unique marks. Vertices matching with these marks can be performed in a different branch of the shader program and the desired positions can be computed. Refer to Figure 4.15, where the gaps from the left mesh have been closed in the right mesh by this method.

This solution works rather satisfying for smooth subdivision, although small micro-holes of pixel-size can still appear during the rendering. The remedy therefore are the already mentioned zero area triangles. These triangles behave just like any regular faces,

although since they do not possess any area a normal computation is impossible. Fortunately this drawback is in fact not one, because (1) the normals can be computed for each vertex either by a sum of the cross products normals of its faces or by evaluating the limit normal. (2) As explained in section 4.2.1 the actual connectivity of the mesh does not depend on the triangulation in the final rendering, since these is determined by a precomputed index buffer. Just for this reason, the T-faces can be introduced very easily by including them into the index-buffer computation routine.

An further advantage of the T-faces is the fact that indeed during displacement it is hardly possible to keep the borders of differently sampled patches consistent. Here the T-triangles do a great job by totally closing any arising discontinuity.

4.2.6. Rendering

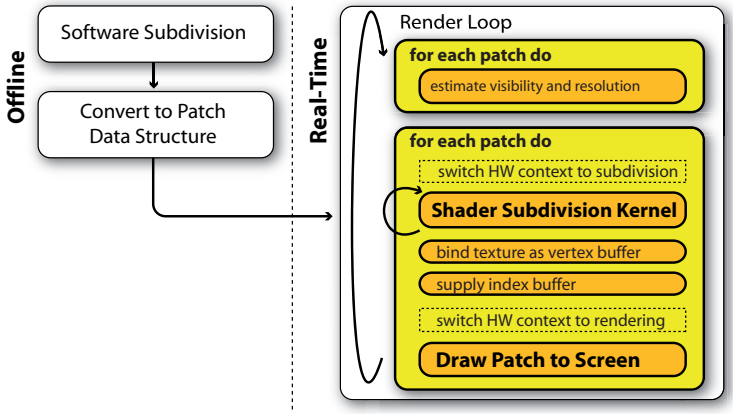


Figure 4.16.: *Data flow in the application. Left: preprocessing. Right: each patch is subdivided on the fly during the rendering. The inner shader kernel is depicted in Figure 4.17.*

The final rendering routine is a cycling function containing nested loops. Its basic structure can be obtained in Figure 4.16. In each frame the outer loop cycles over all patches and subdivides each depth-first to the desired resolution. Thus, there is one more inner loop in the subdivision kernel which is depicted in Figure 4.17. Once a patch is fully computed, the resulting geometry-texture is reinterpreted as a vertex-buffer and an one-for-all, off-line precomputed index-buffer is applied. Here the rendering pipeline must perform a context switch to change the render target to the frame buffer and load another shader program for visual output. In fact, this constellation is the actual bottleneck of the system. The frequent context switches of the shader cause the longest delays in the GPU pipeline, reducing the overall utilization of the graphics hardware.

Nevertheless, it results in a enormous subdivision speedup if compared to traditional software implementation based on half-edge data structures. Depending on the hardware, multiplicity of vertices are processed simultaneously by the GPU. Also, the fact that the computed data stored in the texture memory can be directly reinterpreted as vertices without passing it back to the system memory allows to proceed all this in real-time.

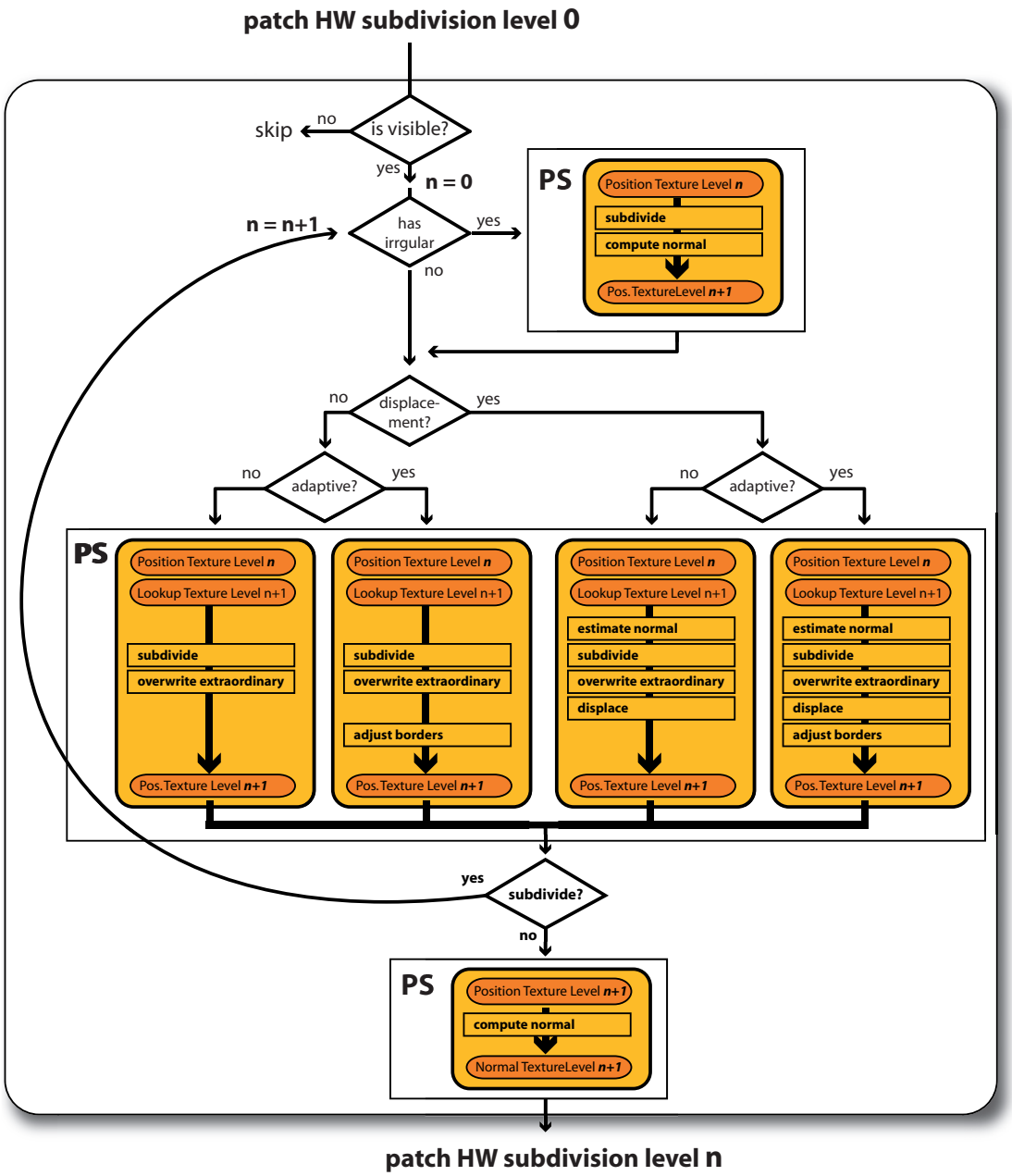


Figure 4.17.: *Data flow in the subdivision shader. Each input patch is supplied to the pipeline at level 0. A proper shader program is chosen interactively in order to either subdivide or displace the patch. If the patch contains an extraordinary vertex, its position is computed in an additional pass and overwritten. In case of adaptive subdivision, also border vertices are adjusted (see section 4.2.5). The procedure can be repeated while a certain subdivision level has not been reached. After last step normals are computed. Finally, the geometry-texture is reinterpreted as vertex-buffer directly in hardware memory and rendered to the scene. The outer loop over all patches is depicted in Figure 4.16.*

5. Results

In this chapter the implemented method of multiresolution subdivision will be examined in two main categories: performance and visual results. The first section will show runtime performance and a comparison to classic subdivision in software. The second section will discuss the parameter-settings of the scaling function in order to achieve good-looking simulations of real life surfaces.

5.1. Performance Analysis

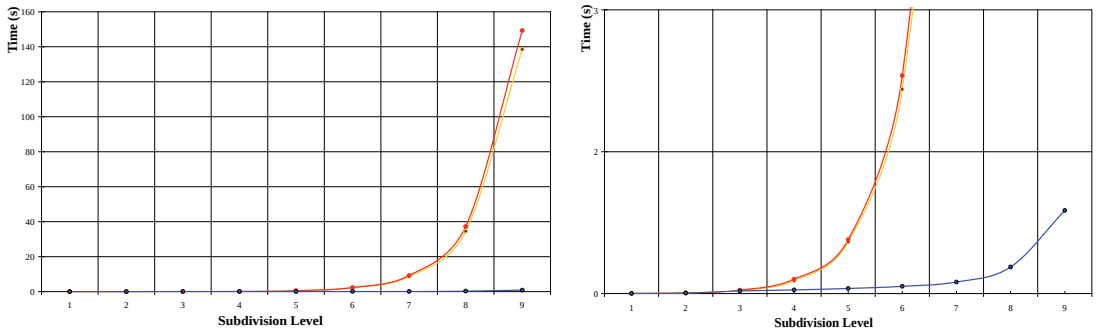
This section will present direct runtime comparisons on several tests. The model used for all of tests is a simple unit cube at original zero level. It is build out of 8 vertices and 6 faces.

All tests have been done on a Intel Core Duo 2.4 Ghz machine with 2GB main memory. The graphics is supported by an ATI Radeon X1900 GT GPU with 36 pixel pipelines, 513 Mhz core clock and 256 MB installed video memory, 768 MB texture memory and 660 Mhz memory clock.

All presented timings are given in seconds and show an average over 80 measured frames.

Hardware vs. Software. Figure 5.1 shows the comparison of the runtimes of traditional subdivision performed on half-edge data structures in software and the hardware accelerated solution up to level 9. Please note that the first two steps before hardware subdivision have been performed in software to ensure the isolation of extraordinary points and to achieve an appropriate size of the mesh patches. Thus, the mesh supplied to hardware is already composed of 96 faces. One can observe that hardware subdivision provides a huge performance improvement.

An interesting conclusion can be stated in terms of the ratio of the compared runtimes. It shows that the time complexity of the hardware does not follow the same increase as the one of the software. On the third step (which is the first one in hardware) hardware is only 1,7 times faster while on the ninth step it is almost 160 times faster than software. Indeed, the supposed reason is the parallelization of the hardware chip. Since it has

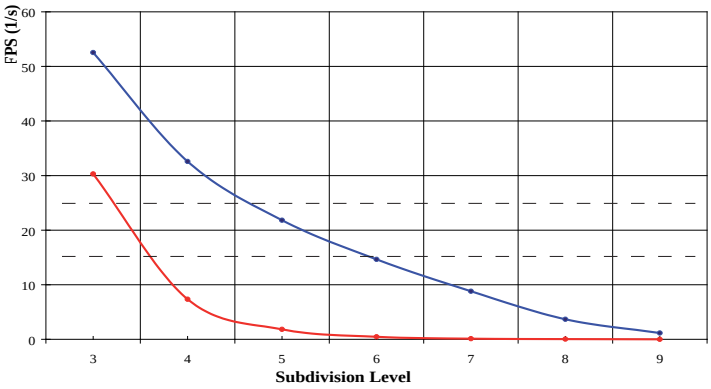


level	1	2	3	4	5	6	7	8	9
HW (blue)	0,002	0,006	0,019	0,031	0,046	0,068	0,114	0,273	0,871
SW_{full}(red)	0,002	0,006	0,028	0,116	0,418	1,735	6,945	27,993	112,040
SW_{sds}(yellow)	0,002	0,006	0,025	0,103	0,410	1,614	6,490	25,876	103,998
ratio (SW/HW)	-	-	1,7	4,4	11,9	31,6	76,1	126,6	159,0
outp. vertices	26	98	386	1538	6146	24578	98306	393218	1572866

Figure 5.1.: *Runtime comparison between traditional software subdivision (red) and the method in hardware (blue). Note, that the first two levels of the hardware subdivision has been performed in software. SW_{sds} refers to subdivision only, SW_{full} includes explicit mesh connectivity computation (time in seconds).*

36 pixel pipelines it can process as many vertices at the same time. Moreover, the low subdivision steps seem to be slower than the higher, which is probably caused by the overhead of the initialization of the pipeline in the first pass.Than, with each higher hardware step the discrepancy between both approaches increases. The open question which remains here is if the hardware could be even faster if it would be better utilized at all steps. The assumed answer is a yes, since it is not designed to process 8×8 pixel textures, where the data and instructions transfer causes a relatively large overhead if compared to the contained information. Processing of larger textures would reduce this overhead.

Figure 5.2 depicts theoretically possible frames per second on the given hardware. These values are obtained from the measurements shown in Figure 5.1 thus, additional overhead caused by rendering of visual effects has not been taken into account (albeit the context switches of the hardware has been measured). The table below collects the observed framerates obtained by the means of a hardware driver internal monitoring tool. These three rows of the table show the complete hardware subdivision time (HW_{full}) followed by the back face culling mode, where invisible patches are not supplied to the pipeline ($HW_{culling}$) and finally the adaptive tessellation on the three highest steps ($HW_{adaptive}$).



level	3	4	5	6	7	8	9
1/HW	52,548	32,576	21,811	14,642	8,800	3,666	1,148
1/SW	30,303	7,353	1,832	0,463	0,116	0,029	0,007
HW_{full}	38	25	19	15	9	4	1
$HW_{culling}$	48	31	24	18	12	6	2
$HW_{adaptive}$	-	-	-	-	13	8	2
outp. vertices	386	1538	6146	24578	98306	393218	1572866

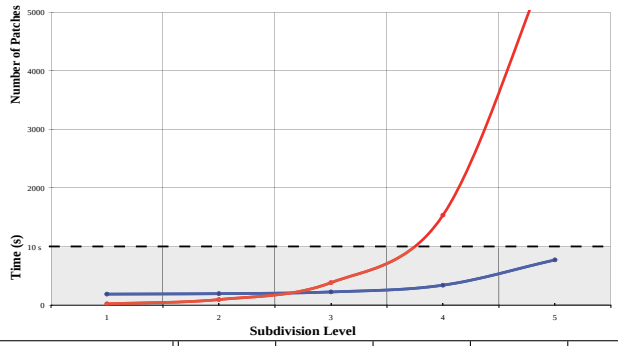
Figure 5.2.: *Theoretical frames per second of hardware (blue) and software (red) subdivision. Table below: actual framerates observed on the screen.*

Here one might wonder why the actual framerates captured on the screen provide better results as the theoretical best cases. The answer is that they were measured with a driver tool which does not update the captured output very accurate and the method used for scene illumination was a standard per pixel phong-shader which does not consume much time. Thus, its overhead is a very minor one.

Furthermore, one can observe in the table how invisible patches culling affects the performance. Indeed, it is an important ingredient and depending on the models shape and the camera position it can reduce the scene complexity very significantly. On the here tested example of a cube (which becomes quite spherical after subdivision) it removes almost always the same number of back-facing patches.

The extension of adaptive subdivision allows the ability to improve the performance even more. Its drawback is that it also reduced the visual appearance of the surface by lowering subdivision levels of some patches. In fact, it has not have been balanced enough in the current implementation yet, because the subdivision levels are determined by a static distance table. Nonetheless, it promises to be a good possibility to still improve the performance if its application would be appended on more dynamical scene properties.

According to the observed results it can be stated that subdivision up to the fifth level (HW level 3 with back face culling) is possible with absolutely interactive framerates. Also sixth and seventh step provide still acceptable results such that it can be spoken about real-time. The last row of the table in Figure 5.2 shows the actual number of vertices displayed on the screen. As one can obtain these are enormous.



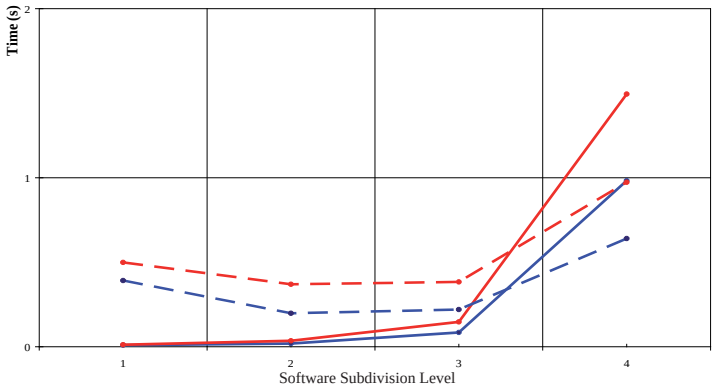
level	1	2	3	4	5
no. patchs	24	96	384	1536	6144
time	1,891	1,975	2,272	3,407	7,722
ratio: (time/patch)×10	0,788	0,205	0,059	0,022	0,012

Figure 5.3.: *Conversion time of a mesh into texture data structure depending on the number of input patches. Level k denotes number of patches by* 4^k *(time in seconds).*

Conversion Time. Figure 5.3 presents the time which is needed to convert a mesh into the texture-data structure depending on the number of patches. The conversion happens basically in a linear manner. Each face of the input mesh is processed sequentially by always the same routine. It turns out that large numbers of patches are performed faster than small ones. This issue should be again reasoned in a rather well system cache utilization and furthermore also in the extraordinary vertex conversion time, which is much more costly than the regular ones. In all five presented steps the number of these was always exactly eight.

One question for the future remains open at this point. It is if it will be possible to perform such conversion in real-time in order to append the hardware subdivision on selected regions of large, conventionally rendered meshes. This would allow to refine close-up views of arbitrary scenes on the fly in hardware.

Context switch Bottleneck. Finally, Figure 5.4 shows an interesting relation. The initial model has been supplied to the hardware subdivision in four different resolutions. The runtime has been than compared depending on the number of mesh patches.



SW level	1	2	3	4
HW steps: 1	0,009	0,019	0,084	0,982
HW steps: 2	0,012	0,035	0,147	1,495
no. patchs	24	96	384	1536
ratio: (HW1/patch)$\times 10^3$	0,391	0,198	0,220	0,639
ratio: (HW2/patch)$\times 10^3$	0,498	0,369	0,383	0,973

Figure 5.4.: *Hardware subdivision time depending on number of input patches. There has been one (blue) and two (red) HW-steps performed depending on steps of previous SW-SDS. Number of patches can be obtained by* 6×4^k *withs* $k =$*level. Dashed lines show the ratio of HW/patch. Time is given in seconds.*

The most relevant conclusion can be obtained by inspecting the ratio of the number of patches to the subdivision time. While this value is higher at the very first step than at the second and third ones, it grows rapidly up from the fourth step. The reason of the first value is surly a quite bad utilization of the system by providing only 24 patches. The amazing property can be seen in steps 2 and 3, where the time-per-patch ratio is almost linear such that the hardware needs the same time to subdivide 96 as well as 384 patches independently of the chosen hardware level (note that HW at step 1 deals with 8×8 patches and HW step 2 with 14×14). Up from the 4th software step this ratio increases which implies that the shader performance is lowered by the number of supplied patches, while their sizes remain the same.

Thus, it turns out that the hardware pipeline is heavily dependent on the number of supplied patches. This is indeed obvious due to the way of how patches are provided. The time consumed by permanent texture bindings takes a lot of the benefit of the hardware performance away. By moderately lowering the number of patches it does not play such a role and the rendering times are terrific. But with growing number of patches the performance decreases rapidly and the method tends to be absolutely not suitable for real-time.

Hence, this relation shows that by removing the bottleneck of providing the enormous geometric data to video memory a new – perhaps even worse – bottleneck has been created. Nevertheless, the method proves clearly the possibility of geometry processing on the GPU and provides a possibility in order to reduce the mentioned drawbacks by a modified rendering strategy.

5.2. Effects of Different Scaling Parameters

While last section has deald with the performance of the developed method, this one will focus on the visual results. In order to create a set of possible surface deformation, in section 4.1.2.4 an displacement offset scaling function has been introduced. It can be applied independently of the object dimensions on the multiresolution displacements in order to influence it on particular levels. Recalling the function from equation 4.3:

$$s_1(k) = \frac{1}{2^{Bk}}A \qquad A,B > 0 \text{ and } k \in \mathbb{N}\,,$$

the displacement can be controlled with parameters A and B where A can be also attached to the local properties of the mesh in order to reference the offset to local object properties. Furthermore, in empirical tests it has turned out that the function can be modified in a way such that the factor A can be additionally constrained by multiplying it with B, which results in much lower absolute scaling values:

$$s_2(k) = \frac{1}{2^{Bk}}(AB) \qquad A,B > 0 \text{ and } k \in \mathbb{N}\,.$$

Figure 5.5 compares this two functions. The benefit of this extension is that the magnitude of A is quite nice restricted and it does not have to be changed too much manually.

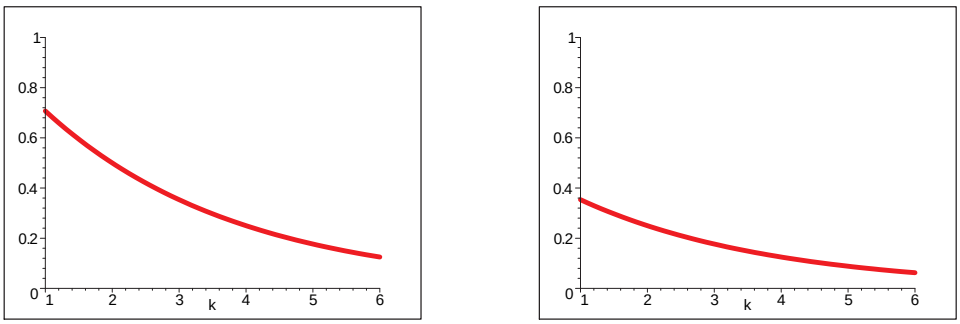


Figure 5.5.: *Modified scaling function. Left $s_1(k)$ and right $s_2(k)$, both with $A = 1$ and $B = \frac{1}{2}$.*

The visual response on the scaling has been investigated mostly on the 'Hand' model which is pictured in Figure B.19 at its original level. In most cases the displacement is derived from the 'dakota leather' map shown in Figure B.2 and scaled with the modified scaling function $s_2(k)$. The original map has been generated by a 'shape-of-shading' stereo-metric approach and has been decomposed into its sub-bands by the method presented in section 2.3.3.

Scaling. Appendix B depicts the effects of different parameters of the function $s_2(k)$ in colored Figures B.7 till B.11. In all examples the model has been subdivided up to the sixth hardware level. Below each setup a plot of the function is presented such that one can see, how particular resolutions have been influenced by the scaling function. Figure B.7 shows the 'Hand' with a rather low global scaling ($A = 0.2$) and compares it with different B settings. Intuitively, the second result $B = 0.5$ seems to appear most realistic. In the following test Figures the value of A has been increased stepwise. This enables one to see how A does control the global influence on all sub-bands. Thus, the total displacement can be controlled by the magnitude of A. In the test Figures this value has been set as a constant over the entire model.

The distribution of how much of each frequency participates on particular levels can be controlled by B. On can see that values smaller than 1 are flattening the curve of $s(k)$ and allow one to distribute the scaling to all bands with almost the same magnitude. This is in fact the opposite to the natural scaling which tends to vanish for higher steps k. The visual effect of this is such that the very fine features become much more prominent and develop strong wizened surfaces. On the other hand by increasing the parameter B the curve becomes more steeply and the high frequencies become eliminated. In contrast, the lower bands become more enhanced and the results employ quite smooth surfaces in the small but strong deformed in the large.

Figure B.13 shows displacements affected by a constant A in the comparison with A attached to local areas of each vertex. As one can see, the coarse deformation remain the same while the offset nestles to the shape attributes. It turns out that this kind of local reference for the offset provides best results.

Offset Referencing. Figure B.6 shows two spheres where the larger one is a copy of the smaller one scaled by factor 1.5. The top image shows both objects with displacement referenced on the global (x, y, z) coordinates with exactly the same scaling configuration. As one can see, the magnitude of the offset changes with the objects' size.

In contrast, the bottom image shows the mentioned spheres with displacement offset derived from the local texture coordinates. In this case the displacement does not change its magnitude.

Curvature Radius Constraint. Figure B.12 depicts an example of constraining the displacement offset by the means of local curvature. This instance has been generated by a modified version of the displacement routine. In order to passover the hardware limits for temporary registers, most other functionalities as the border adjustment as well as extraordinary vertex handling have been detached from the pipeline. The curvature has been estimated on vertex- and-edge points in similar way to the one for normals estimation presented in section 4.2.4. Face-points curvature has not been estimated and their radius constraint has been set statically set to infinite. Nonetheless in Figure B.12 one can notice how a local measure constrains the displacement and the strongly deformed surface develops concave features without locally intersecting itself.

The presented examples can be seen in color in Appendix B. The control of the displacement turned out as not too tedious and due to the real-time response on the given parameters, the proposed approach seems to provide at least a basis for possible control mechanisms.

6. Conclusions and Future Work

This final chapter summarizes the entire thesis and points out its achievements as well as the problems occurred during the development. In the end it will suggest extensions and solutions to open problems and propose possible applications of the developed technique.

The aim of this work was to develop naturally looking surface characteristics on smooth subdivision surfaces. To achieve this, the idea of applying multiresolution displacement has been favored.

6.1. Summary

In order to implement the undertaken goal, a wide spectrum of already existing methods and related topics have been researched. To understand what happens during the subdivision evaluation, the theory behind this method has been reviewed. Also displacement which seemed to be quite trivial, turned out to have interesting peculiarities especially if applied on multiresolution surfaces. In dozens of publications several interesting issues about the underlying mathematics could have been obtained. Section 2 is the result of this research.

Catmull-Clark Scheme. The first examined area was the one of subdivision surfaces. Section 2.1 tries to comprise all as important rated issues. In fact it covers only the tip of the iceberg, since this work is focused mainly on the Catmull-Clark subdivision scheme. While it is a very good example of the general approach and it delivers excellent results, it is by far not the only one. Nevertheless, on its instance some general attributes of subdivision have been shown in section 2.1.6, where the parametrization and the analysis of schemes have been discussed. On the other hand, the specific issues of the Catmull-Clark rules were helpful to become an idea how subdivision works. In particular it is its derivation from the bicubic B-splines as discussed in section 2.1.4. It has been shown that both representations are in fact equivalent and represent the same surface. This is at least true for regular mesh regions. In order to examine irregular vicinities for their continuity and convergence, an analytical method has been attempted. It turned out that these properties are by far not trivial and therefore it has taken a long time until proofs

have been provided. They require the eigenanalysis of the subdivision matrix and lead deep into mathematical roots of subdivision (refer to section 2.1.6.4). One practical benefit of this research is the possibility to compute exact positions, tangents and normals on the limit surface at given vertices. While it is an interesting method, it does not resolve the problem of the expensive recursive computations, since it does not allow for the evaluation on arbitrary positions. This task has not been treated in this thesis and existing solutions can be obtained in the literature proposed at the end of the section 2.1.6.5 (i.e. Jos Stam [Sta98] and Denis Zorin [ZK02]).

Displacement. Since the goal was to create natural surface effects, displacement has to be studied too. While this quite simple method can be generally explained in few words, in order to apply it progressively, further issues had to be taken into account. General displacement has been defined as a function which fetches an offset value at each vertex from an underlying displacement map by the means of the surfaces' parametric (u,v) coordinates and shifts the vertex by this value along its normal. Two terms have been mentioned here which had to be inspected more in detail: (1) displacement map and (2) parametrization.

Displacement Maps. The first one contains the actual features from the input sample. As mentioned, the approach attempted in this thesis was to distinguish the features by the means of their spatial size This can be done by the spectral decomposition of an image. Usually, spectral decompositions are performed by computing the discrete Fourier series coefficients of a given signal. Each of these represents a harmonic frequency.

Section 2.3.3 presents a well known albeit interesting algorithm for decomposing a discrete image into its spectral sub-bands in the spatial domain. It is an alternative approach in order to omit a discrete Fourier analysis. This is possible because of the equivalence of the convolution operation in the spatial domain to the multiplication in the frequency domain. Fourier coefficients can be computed as a linear combination of sinusoids, which can be also expressed as a convolution sum of shifts of the unit impulse. Due to this relation the analysis can also be done directly in the spatial domain by convolving an image by appropriately chosen kernel function. This saves the overhead of a discrete Fourier transform and its inverse and thats why it performs faster for small filters. For large filter kernels the discrete Fourier transform is the tool of choice.

The algorithm presented in section 2.3.3 is based on convolution and provides an approximated spectrum of a given input sample. It has been implemented for this thesis and it turned out that it delivers sufficient results in reasonable time. The resulting image pyramids consist of slices in one octave steps where each one contains the 'hidden' features of similar spatial sizes. The values in the slices have been matched to different mesh resolutions and are used as a source for the displacement offset.

Parametrization. The second item which was mentioned in order to define the displacement is the surfaces' parametrization. It has been studied in section 2.1.6 in order to explain how the displacement map is tied to the surface, how the normals are defined and how it is propagated to further submeshes. Additionally, the definition of a parametrized surface has allowed for the ability to derive the definition of the surface curvature in section 2.44.

Curvature. The issue of curvature has been researched for the reason of obtaining a measure for the maximal displacement offset in order to ensure that the resulting surface remains manifold. While basically it seems obvious that an offset curve does not develop self-intersections or cusps if the shift is smaller than the radius of the local curvature, the interesting task was to find a way to show it formally. To accomplish this, some essential foundations of the differential geometric properties of surfaces had to be studied. The results are collected in section 2.44 and give a short but comprehensive overview of the basic background matter. It introduces the definitions of the gauss map as well as of the first and second fundamental forms of a surface. From these provided essentials the maximal and minimal principal curvatures have been derived and it has been shown that an offset surface develops a cusp if the shift distance is equal to one of these curvatures radii. In order to use these cognitions in practice, section 2.51 introduces the discrete approximations for curvature measures.

Offset Reference. Defining the maximal offset in terms of the curvature does not automatically provide the expected results. In fact, curvature radius was meant to constrain the displacement but not as a general reference. Thus, in section 2.2.3 two basic possibilities for deriving the displacement magnitude have been discussed. On the one hand the displacement can be defined in terms of the global space by computing local surface properties in euclidean coordinates, on the order hand it can be defined by surface-only specific manner by computing the offset in terms of the (local) parametrization. Both of those have their pros and cons. Surfaces displaced in global coordinates are independent of any parametrization, but since the offset is attached to the size of the mesh faces, they do depend on their actual tessellation and moreover on their actual dimension according to the embedding space. This is because displacement is performed along the unit normal and an offset derived in the described way scales along the normal for each object size differently.

The second mentioned possibility allows the ability to define the measure of displacement in terms of the surfaces' parametrization. While this makes it independent from the embedding space, it becomes sensitive to the quality of the parameterization and especially to its continuity. Cracks in the parametric space lead to undesired and unpredictable displacement results because no standardized formula can be defined for such

cases. In fact, this leaves the responsibility of the magnitude and distribution of the displacement to the way of how an object has been parametrized. This can be either computed or even handcrafted by the user. The benefit of this approach is surely the separation from the global space.

Scaling Function. Independently of where from the offset is derived, since it could be defined evenly as a constant value, there is also the question of how much to displace on each level. The general assumption stated previously was that it should be scaled by the means of the current resolution because this also relies on the size of the features in the input sample.

Basically, there are two possibilities to scale the offset: (1) scaling proportional or (2) inverse proportional to the subdivision level. While the first would lead to a consecutive surface growing, the latter would usually converge to zero for infinite surface levels.

It is obvious that the desired effects can be achieved by the latter proposal, since it is related to the natural scaling as stated in the assumption. This function is the inverse proportional to the frequency $(\frac{1}{2^k})$, which is exactly the step between two resolution levels. The natural scaling works very well and delivers smooth models which do not usually develop any self intersections, on the other hand this approach is rather static and limits the variety of possible effects significantly. Moreover, this function converges rather fast toward zero with the effect that high frequencies become almost vanished. Therefore in section 2.2.4 the question of appropriate scaling functions has been elaborated. The proposed function has been inspired by the natural atrophy of the area of surfaces' polygons, which is approximately the square of the frequency term stated above. The derived function $s(k)$ employs two parameters by which means it is possible to control its convergence and furthermore, it can be attached to the local surface area in both – global and parametric space.

This work has tried to specify a controllable configuration in order to synthesize realistically looking surfaces. It turned out that the control of it is less than easy to do. In section 5.2 several results of the application of this scaling have been discussed.

6.2. Implementation Issues

Recursive subdivision is computationally an expensive process. While nowadays hardware capabilities are enormous, nothing of this will ever change the characteristics of the algorithm. Although there exist other algorithms to evaluate subdivision surfaces (see section 3), in order to apply successive displacement, only the recursive method is suitable. The initial assumption of this thesis was to develop a method which can subdivide in real-time and this up to very high levels. In a previous work, software sub-

division has been implemented by the author and thus the runtime properties have been known very well. It was clear that the attempted results cannot be reached by a software solution. For this reason a hardware implementation has been chosen to evaluate the multi-band displacement in a reasonable time. Another additional goal was to develop a method which might be competitive to the established parallax displacement mentioned in section 3.

It seemed to be possible because of the capability of recent graphics hardware, here especially the rendering to vertex buffer extension. Moreover, since graphics hardware is designed to process parallelized tasks, it also seemed that subdivision can profit from it. The recursive algorithm can be fully parallelized since each of the vertices on level $k+1$ can be computed absolutely independently out of level k.

In fact, it turned out that the assumptions basically were true. Results showed in Figure 5.1 demonstrate that the process done on the GPU executes very much faster than the software implementation. On the other hand results showed in Figure 5.4 shows rather clearly that the runtime of the shader program depends heavily on the number of patches supplied sequentially to the hardware pipeline.

It is evident that the reason for these limitations has its origins in the frequent context switches and texture bindings. In the current implementation the hardware cannot be utilized to its full performance, since it is not designed to process many small textures, but rather less bigger ones. This drawback might be advanced by designing another strategy for texture transfer to the subdivision shader, albeit it is a rather difficult task, since this would also require significant changes in the shader programs. These are in the current implementation on the absolute limits of the capabilities of the hardware instruction counts. More precisely, even the presented ingredients could not have been applied all at the same time. Subdivision with simultaneous derivation of the local area consumed all temporary registers of the graphics chip, such that the border adjustment routine could not has been performed in the same shader pass any more. The intended curvature constraints could not have been implemented at all in the chosen solution, such that an extra shader program had to be designed in order to achieve at least an exemplary result.

It should be mentioned that the implementation strategy has been chosen quite early and intuitively due to the lack of experience on GPU programming by the author. The architecture of the entire shader complex is very much influenced by traditional ways of software programming. In a review, it can be stated that the implemented way might be not the optimal one. Since the newly created bottleneck is mainly caused by the frequent context switches and therefore depending on the number of input patches (this can be seen in Figure 5.4), a remedy might be an architecture which minimizes these. This could be achieved for instance by providing the mesh to the GPU in much bigger textures and therefore less frequently. Unfortunately an implementation of it would

involve too significant changes of the data structures, such that it has not been included into the scope of this work anymore.

As an advantage of the presented method one can point out that this solution proves clearly that subdivision on the shader is possible. It shows that even on already outdated graphics hardware, a rather sophisticated geometry processing can be performed in a parallelized manner in absolutely convincing runtime. Also this implementation shows impressive results, albeit on rather limited input meshes.

Thus, despite the mentioned redesign of the architecture, the presented implementation can be still viewed as a starting point for further optimizations in order to improve the visual quality as well as the performance. Following additional extensions are conceivable:

- In the adaptive approach, the additional detail of finer subdivision levels could be introduced continuously, by scaling the displacements based on the viewer distance during the change from one subdivision level to the next.
- The borders that are forced to the coarser subdivision level in adaptive subdivision could be modified to a linear transition zone between differing subdivision levels.
- The shader programs should be optimized for optimal pipeline usage.

6.3. Conclusions

The initial assumption, as stated in section 1.2 that the chosen way will allow the creation of surfaces which are more rich in detail than created by conventional displacement mapping turned out as true. Multi-band displacement is indeed a very powerful possibility to model (in fact, in this context it can be spoken about displacement as a modeling method) a specific class of objects.

The characteristics of these objects are given by features hidden in a source image. By 'hidden' it is meant that these features are not directly visible albeit they are components of the image. There is nothing added, thus there is no synthesis on the input signal. The synthesis happens on the other subject of this work: the surface. It is an obvious task to synthesize new surfaces by simply adding detail to it by displacing its vertices. If it is added once it does not allow the variety of shapes possible in the presented approach. This is due to the fact that details of the source are added successively, which demands a multiresolution representation of the domain to be synthesized. Usually surfaces do not possesses these. However, this kind of representation is implicitly given by the recursive subdivision surfaces approach. This fact makes them ideally suitable for successive displacement.

In contrast, the features from the input image have to be obtained first. This is done by decomposing the image into its spectral components. Each of them contains features carried by a harmonic frequency and thus details of similar spatial size. These can be obtained and matched with sequential resolutions on the surface. Metaphysically it can be seen as gathering some properties of one domain by its analysis and transferring them to an other by a synthesis.

Now lets leave the area of philosophy and scratch the original idea again. The concept of the thesis as presented in section 4.1 was to add features of similar sizes from the input image to the surface on an appropriate resolution. Here 'appropriate' means that the spatial sizes of the details should correlate to the spatial sizes of the mesh polygons. Therefore the assumption has been made that the lower frequencies of an input image represent the more prominent features and the higher frequencies the finer ones. Also the magnitude of the distortion at each resolution is implied by its own means. This should impose large distortions on low tessellated meshes in order to enforce the basic shape of the object, followed by successive addition of smaller and smaller details along a finer but already strongly deformed surface. The expected results were naturally looking objects with fine structures on the surface. Of course all this had to happen in a continuous and controllable way and furthermore, real-time performance has also been focused on.

In the end, this thesis has presented a method for displacing subdivision surfaces on several resolutions by the means of spectrally decomposed patterns. In fact, to the authors knowledge, such an approach has not been attempted yet. As an additional goal, the entire subdivision and displacement procedure has been implemented on a programmable graphics accelerator in order to achieve real-time performance for very high tessellations.

Mathematically, the presented method can be seen as a extension to existing displacement approaches. In some way, this method can be also considered as a derivation from approaches presented in section 3.2.1 which aim at geometry compression by expressing a mesh in terms of a coarse base domain and a set of scalars which contain the surface details. This method has been proposed by Hoppe *et al.* [LMH00] and in a similar manner also by Gustkov *et al.* [GVSS00]. In contrast to their approaches, in the presented technique the geometric detail is not obtained by an analysis of any existing surface but generated artificially by the means of a supplied pattern samples instead. Furthermore it is also an inverse method to mesh smoothing approaches, which aim in removement of fine details [Tau00] in order to achieve smooth surfaces. While subdivision surfaces has been shown to be smooth, one more remark should be mentioned: if only a limited number of displacement steps has been applied followed by at least one additional pure subdivision step, the created offset surface does maintain again the continuity imposed

by the chosen subdivision scheme. This is C^2 (on regular regions) for the here used Catmull-Clark scheme.

While the techniques mentioned above do not aim directly at visual surface effects, one more similar work should be recalled. The procedural displacement on subdivision surfaces presented by Velho *et al.* [VPYB01] as the most related work. In contrast to their approach, this one does not uses procedures but extracts the details from patterns. For this reason it is a combination of analysis/synthesis. Furthermore, one should note that all the mentioned methods are not capable of real-time performance.

This work was very much concerned on this issue. More precisely, many limitations had to be taken into account in order to achieve such an implementation. The purpose of it was to provide the visual output as fast as possible.

These visual results can be examined in color in Appendix B. It is a small collection which tries to show the possibilities of the presented method. Almost all of the presented examples have been created by only very small changes to the control parameters A and B (see section 5.2). This eases the modeling process and additionally, since the results can be obtained in real-time, it turns out not to be to much tedious. Unfortunately, due to the mentioned mesh size limitations, only very few models could be used.

The presented results allows one to state that the developed technique is very well suited to model naturally looking surfaces of smooth models. Two different issues should be considered as significant extensions in order to compare this method to classical displacement:

1. Usually displacement is applied only once on the final tessellation, which limits this technique to an one-time shift of the surfaces' vertices. This does not allow the ability to achieve coarse deformations of the entire object with additional small features along the deformed surface. This issue has been extended in the presented work by the adaption of multiresolution displacement such that a vertex from the initial mesh is moved several times in different directions, each affected by the previous subdivision steps.
2. If displacement is applied on certain resolutions by always the same displacement map, the resulting surfaces do not develop the variety of differently sized features as in the presented technique (see Figure B.25). This is due to the reason that even if the underlying mesh is dense enough to allow the addition of rather fine features, these cannot be obtained in adequate contrast from a traditional displacement sample. Here the idea of separating the features was the key ingredient in order to impose very fine details.

Finally, it should be noted that all three subjects of this thesis can still be applied separately. The implemented subdivision kernel can be used without displacement and further on, the displacement is not limited to the presented spectral sub-band maps.

Also another sources for the offset are considerable. These should in best case contain traits of different spatial sizes in order to deform the geometric domain adequate to its particular resolutions.

According the performance of the subdivision kernel, in section 5.1 has been shown that the implemented method provides huge speed-up in comparison to the software solution. This benefit is caused not only by the rough graphics-chip speed but also by its parallelized pipelines. Finally, it has been also shown that the chosen architecture is very much dependent on the number of processed mesh patches which unfortunately limits the stated benefits. Nevertheless, the method proves clearly the possibility of geometry processing on the GPU and provides a possibility in order to reduce the mentioned drawbacks by a modified rendering strategy.

For the presented reasons the developed technique can find several possible applications in the practice. Few of them will be proposed in the next section.

6.4. Applications and Future Work

In section 1.2 it has been mentioned that the method developed in this thesis is a part of the basic research project GeomeTree [VG04] at the VRVis Company. The part which is covered by the presented work aims in visualization of very closeup views. The developed software has been fully integrated as a module into the internal rendering framework of the VRVis Company. It can be appended as one possible rendering front-end of a rendering pipeline.

Geometry Refinement Front-End. In the current development stadium the presented 'subdivision shader' can be used with any manifold meshes of arbitrary topology. It cannot be applied on specified parts of large meshes. In future it is intended to extend the 'subdivision shader' module in order make it possible to use the hardware accelerated front-end as a dynamic extension to general and large meshes. In this constellation it would be possible to render very large and statically stored meshes in the usual way and in the case of camera zoom-ins the hardware accelerated subdivision could be appended on demand in order to refine only the visible region of the entire mesh.

This would work in real-time if the mesh-to-texture conversion operation can be optimized in a way that allows real-time conversions only (see Figure 5.3).

Displacement Maps Extension. The presented hardware subdivision method is independent of the additional displacement extension. Also the latter is not constrained to the usage of the presented sub-bands displacement maps. Thus, in order to extend

the developed render technique, it is possible to broaden the variety of supplied displacement maps. During the evaluation of the current method, it turned out that beside the introduced multiresolution displacement maps, also classical as well as by 'shape-from-shading' and stereo-metrically calculated displacement maps deliver considerable results.

The idea of matching of particular frequency-bands with adequate mesh densities seems to be a rather good way to create realistically looking shapes which can be generated on recursive subdivision surfaces. In order to achieve further results the tool of image processing should to be extended. Images analyzed by a larger variety of different methods would enrich the variety of possibly surface effects and possible would allow to reconstruct more accurately existing patterns [TFA05]. Also shape reconstruction of entire 3D-object is considerable in the presented method which would be a extension to the already existing displaced subdivision surfaces [LMH00].

Parametrization Extension. A further extension which could be taken into account is the one of improving the parametrization mapping on the surface. In recent literature methods to synthesize textures directly on surfaces have been introduced [YHBZ01]. In order to detach the displacement from the planar texture, this approach could be considered to generate a displacement field along the surface.

Animation. While it has not been explicitly mentioned yet – obviously the developed method is also very well suited to render animated surfaces in real-time. Since at each frame the entire subdivision is performed from the beginning on, it is a quite small step to introduce time driven displacement sources. This extension would allow the ability to create a further class of surfaces which might animate growth as well as other visually very good-looking effects.

In consideration of the capabilities of the currently incoming generation of graphics accelerators, which include intentionally the ability of geometry processing, the list of possible applications of methods as presented in this thesis could be still continued. It is to envision that such extensions will come and will become state-of-the art very soon. While discussing these would go beyond the addressed topic, the last example of a possible application of the presented method should finalize the thesis.

Bibliography

[AAB+84] Edward H. Adelson, C. H. Anderson, J. R. Bergen, Peter J. Burt, and J. M. Ogden. Pyramid methods in image processing. *RCA Engineer*, 29(6), November 1984. 62

[Aut06] ATI Research Inc. (Different Authors). Ati sdk documentation (march 2006). http://ati.amd.com/developer/radeonSDK.html, 2006. 60, 87

[BA83] Peter J. Burt and Edward H. Adelson. The laplacian pyramid as a compact image code. *IEEE Trans. on Communications*, (4), April 1983. 62, 63

[BAD+01] M. Bóo, M. Amor, M. Doggett, J. Hirche, and W. Strasser. Hardware support for adaptive subdivision surface rendering. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 33–40, New York, NY, USA, 2001. ACM Press. 70

[Bar84] Alan H. Barr. Global and local deformations of solid primitives. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 21–30, New York, NY, USA, 1984. ACM Press. 48

[BKS00] Stephan Bischoff, Leif P. Kobbelt, and Hans-Peter Seidel. Towards hardware implementation of loop subdivision. In Stephan N. Spencer, editor, *Proceedings of the 2000 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware (EGGH-00)*, pages 41–50, N. Y., August 21–22 2000. ACM Press. 67, 68

[Bli78] James F. Blinn. Simulation of wrinkled surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 286–292, New York, NY, USA, 1978. ACM Press. 1, 36, 69, 71

[BLZ00] Henning Biermann, Adi Levin, and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 113–120, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 5, 9, 30, 67, 69

[BMZ04] Ioana Boier-Martin and Denis Zorin. Differentiable parameterization of catmull-clark subdivision surfaces. In Roberto Scopigno and Denis Zorin, editors, *Eurographics Symposium on Geometry Processing*, pages 159–168, Nice, France, 2004. Eurographics Association. 44

[BMZB02] Henning Biermann, Ioana M. Martin, Denis Zorin, and Fausto Bernardini. Sharp features on multiresolution subdivision surfaces. *Graph. Models*, 64(2):61–77, 2002. 70

[BS88] A. A. Ball and D. J. T. Storry. Conditions for tangent plane continuity over recursively generated b-spline surfaces. *ACM Trans. Graph.*, 7(2):83–102, 1988. 43, 44

[BS02] Jeffrey Bolz and Peter Schröder. Rapid evaluation of catmull-clark subdivision surfaces. In *Proceedings of the Web3D 2002 Symposium (WEB3D-02)*, pages 11–18, New York, February 24–28 2002. ACM Press. 67, 68

[BS03] Jeffrey Bolz and Peter Schröder. Evaluation of subdivision surfaces on programmable graphics hardware. In *http://www.multires.caltech.edu/pubs/GPUSubD.pdf*, 2003. 67, 68

[Car93] Manfredo P. Do Carmo. *Differentialgeometrie von Curven und Flächen*. Vieweg Studium, Braunschweig/Wiesbaden, Germany, 1993. 34, 49, 50, 52, 53, 121

[CC78] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, September 1978. 5, 28, 30, 69

[Cha74] G. Chaikin. An algorithm for high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974. 22, 30, 33, 67

[Coo84] Robert L. Cook. Shade trees. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 223–231, New York, NY, USA, 1984. ACM Press. 69

[CR89] S.-L. Chang and M. S. R. Rocchetti. Rendering cubic curves and surfaces with integer adaptive forward differencing. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 157–166, New York, NY, USA, 1989. ACM Press. 68

[DeB72] C. DeBoor. On calculating with B-splines. *J. Approx. Theory*, 6:50–62, 1972. 15, 68

[DH00] Michael Doggett and Johannes Hirche. Adaptive view dependent tessellation of displacement maps. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 59–66, New York, NY, USA, 2000. ACM Press. 70

[DHKL01] Nira Dyn, Kai Hormann, Sun-Jeong Kim, and David Levin. Optimizing 3d triangulations using discrete curvature analysis. pages 135–146, 2001. 54, 55

[DKT98] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 85–94, New York, NY, USA, 1998. ACM Press. 5, 30, 31, 32, 33, 67, 69

[DS78] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10:356–360, September 1978. 5

[EC91] Gershon Elber and Elaine Cohen. Error bounded variable distance offset operator for free form curves and surfaces. *International Journal of Computatinal Geometry and Applications*, 1(1):67–78, 1991. 48, 49

[FB88] D. R. Forsey and R. H. Bartels. Hierarchical B-spline refinement. In *SIGGRAPH '88 (15th Annual Conference on Computer Graphics and Interactive Techniques, Atlanta, GA, August 1–5, 1988)*, pages 205–212, 1988. 70

[GH99] Stefan Gumhold and Tobias Hüttner. Multiresolution rendering with displacement mapping. In *Proceedings of the Siggraph Workshop on Graphics Hardware (EGGH-99)*, pages 55–66, N.Y., August 8–9 1999. ACM Press. 70

[GSS99] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In Alyn Rockwood, editor, *Proceedings of the Conference on Computer Graphics (Siggraph99)*, pages 325–334, N.Y., August8–13 1999. ACM Press. 73

[GVSS00] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 95–102, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 70, 117

[GW01] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. 59, 60, 61

[HB95] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238, New York, NY, USA, 1995. ACM Press. 61

[HKD93] Mark Halstead, Michael Kass, and Tony DeRose. Efficient, fair interpolation using catmull-clark surfaces. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 35–44, New York, NY, USA, 1993. ACM Press. 43, 44, 67

[KL96] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 313–324, New York, NY, USA, 1996. ACM Press. 70

[KS99] Andrei Khodakovsky and Peter Schröder. Fine level feature editing for subdivision surfaces. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 203–211, New York, NY, USA, 1999. ACM Press. 70

[Len03] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics, Second Edition*. Charles River Media, Inc., Rockland, MA, USA, 2003. 16, 121

[LH04] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph.*, 23(3):769–776, 2004. 97

[LMH00] Aaron Lee, Henry Moreton, and Hugues Hoppe. Displaced subdivision surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graph-*

ics and interactive techniques, pages 85–94, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 5, 70, 117, 120

[LR80] J. Lane and R. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. Pattern Analysis Machine Intell.*, 2(1):35–46, 1980. 21, 22, 27, 67

[LSP87] Sheue-Ling Lien, Michael Shantz, and Vaughan Pratt. Adaptive forward differencing for rendering curves and surfaces. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 111–118, New York, NY, USA, 1987. ACM Press. 67, 68

[Mai02] Stefan Maierhofer. *Rule-Based Mesh Growing and Generalized Subdivision Meshes*. PhD thesis, Technische Universität Wien, Technisch-Naturwissenschaftliche Fakultät, 2002. 6, 25, 71

[MH00] K. Müller and S. Havemann. Subdivision surface tesselation on the fly using a versatile mesh data structure. In Sabine Coquillart and Jr. Duke, David, editors, *Proceedings of the 21th European Conference on Computer Graphics (EG-00)*, volume 19, 3 of *Computer Graphics Forum*, pages 151–160, Cambridge, August 21–25 2000. Blackwell Publishers. 69

[MTM07] Przemyslaw Musialski, Robert F. Tobler, and Stefan Maierhofer. Smooth subdivision surfaces over multiple meshes. In *15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2007)*, January 2007. 2

[OBM00] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 359–368, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 1, 71

[OLG+04] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothey J. Purcell. A survey of general-purpose computation on graphics hardware. In Christophe Schlick and Werner Purgathofer, editors, *STAR Proceedings of Eurographics 2004*, pages 21–51, Grenoble, France, September 2004. Eurographics Association. 82

[Par01] Rick Parent. *Computer animation: algorithms and techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. 121

[PBFJ05] Serban D. Porumbescu, Brian Budge, Louis Feng, and Kenneth I. Joy. Shell maps. *ACM Trans. Graph*, 24(3):626–633, 2005. 72

[Ped94] Hans Kohling Pedersen. Displacement mapping using flow fields. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 279–286, New York, NY, USA, 1994. ACM Press. 48

[Pet00] Jörg Peters. Patching catmull-clark meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages

255–258, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 5

[PR98] Jörg Peters and Ulrich Reif. Analysis of algorithms generalizing *B*-spline subdivision. *SIAM Journal on Numerical Analysis*, 35(2):728–748, April 1998. 38, 42, 44

[PS96] Kari Pulli and Mark Segal. Fast rendering of subdivision surfaces. In *SIGGRAPH '96: ACM SIGGRAPH 96 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '96*, page 144, New York, NY, USA, 1996. ACM Press. 67, 68, 69, 83, 88

[Rei95] Ulrich Reif. A unified approach to subdivision algorithms near extraordinary vertices. *Computer Aided Geometric Design*, 12(2):153–174, 1995. 5, 29, 38, 42, 44

[Sha00] Brian Sharp. Subdivision surface theory. http://www.gamasutra.com/features/20000411/sharp_01.htm, 2000. 6, 7

[SJP05] Le-Jeng Shiue, Ian Jones, and Jörg Peters. A realtime gpu subdivision kernel. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1010–1015, New York, NY, USA, 2005. ACM Press. 69

[SMFF04] Volker Settgast, Kerstin Müller, Christoph Fünfzig, and Dieter W. Fellner. Adaptive tesselation of subdivision surfaces. *Computers & Graphics*, 28(1):73–78, 2004. 69

[SMS+03] Tatiana Surazhsky, Evgeni Magid, Octavian Soldea, Gershon Elber, and Ehud Rivlin. A comparison of gaussian and mean curvatures estimation methods on triangular meshes. In *ICRA*, pages 1021–1026. IEEE, 2003. 54

[Sta98] Jos Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 395–404, New York, NY, USA, 1998. ACM Press. 5, 44, 67, 112

[Str88] Gilbert Strang. *Linear Algebra and Its Applications*. Brooks/Cole, 3 edition, 1988. 39

[Sur07] Tatiana Surazhsky. Flexible adaptive tessellation of subdivision surfaces. In *Proceedings of 7th Korea-Israel Bi-National Conference Geometric Modelling and Computer Graphics*, pages 93–97, Seoul, Korea, 28–29 January 2007. 69

[SZD+98] Peter Schröder, Denis Zorin, Tony Derose, David R. Forsey, Leif Kobbelt, Michael Lounsbery, and Jörg Peters. Subdivision for modeling and animation - SIGRAPH '98 course notes. http://www.multires.caltech.edu/teaching/courses/subdivision/subdiv.zip, August 20 1998. 6, 10, 23, 30, 37, 39, 45

[SZSS98] Thomas W. Sederberg, Jianmin Zheng, David Sewell, and Malcolm Sabin. Non-uniform recursive subdivision surfaces. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages

387–394, New York, NY, USA, 1998. ACM Press. 5

[Tau00] G. Taubin. Geometric signal processing on polygonal meshes. In S. Coquillart and D. Duke, editors, *STAR Proceedings of Eurographics 2000*, Interlaken, Switzerland, August 2000. Eurographics Association. 117

[TFA05] Marshall F. Tappen, William T. Freeman, and Edward H. Adelson. Recovering intrinsic images from a single image. *IEEE Trans. Pattern Anal. Mach. Intell*, 27(9):1459–1472, 2005. 120

[TM06] Robert F. Tobler and Stefan Maierhofer. A mesh data structure for rendering and subdivision. In *14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2006)*, January 2006. 85, 86

[TMW02] Robert F. Tobler, Stefan Maierhofer, and Alexander Wilkie. A multiresolution mesh generation approach for procedural definition of complex geometry. In *SMI '02: Proceedings of the Shape Modeling International 2002 (SMI'02)*, page 35, Washington, DC, USA, 2002. IEEE Computer Society. 71

[VG04] Austria VRVis GmbH, Vienna. Geometree project. http://www.vrvis.at/rendering/research/geometree/, 2004. 2, 119

[VG07] E. Vouga and R. Goldman. Two blossoming proofs of the lane-riesenfeld algorithm. *Computing*, 79(2):153–162, 2007. 22

[VPYB01] Luiz Velho, Ken Perlin, Lexing Ying, and Henning Biermann. Procedural shape synthesis on subdivision surfaces. In *SIBGRAPI '01: Proceedings of the 14th Brazilian Symposium on Computer Graphics and Image Processing*, pages 146–153, Washington, DC, USA, 2001. IEEE Computer Society. 58, 71, 118

[Wal01] Johannes Wallner. Self-intersections and smoothness of general offset surfaces. http://citeseer.ist.psu.edu/179938.html; http://www.geometrie.tuwien.ac.at/wallner/parsf-neu.ps.gz, 2001. 48, 53

[Wik07] The Free Encyclopedia. Wikipedia. Curvature. http://en.wikipedia.org/wiki, 2007. 12, 49

[Wor07] Wolfram Math World. Wolfram math world. http://mathworld.wolfram.com, 2007. 19

[WSM+01] J. Wallner, T. Sakkalis, T. Maekawa, H. Pottmann, and G. Yu. Self-intersections of offset curves and surfaces. *Int. J. Shape Modeling*, 7:1–21, 2001. 48, 49, 53

[WW01] Joe Warren and Henrik Weimer. *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. 6, 20, 22, 38, 39, 44, 45, 121

[YHBZ01] Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and shape synthesis on surfaces. In Steven J. Gortler and Karol Myszkowski, editors, *Rendering Techniques*, pages 301–312. Springer, 2001. 120

[YZ01] Lexing Ying and Denis Zorin. Nonmanifold subdivision. In Thomas Ertl, Kenneth I. Joy, and Amitabh Varshney, editors, *IEEE Visualization*. IEEE Computer Society, 2001. 9, 37

[ZK02] Denis Zorin and Daniel Kristjansson. Evaluation of piecewise smooth subdivision surfaces. *The Visual Computer*, 18(5-6):299–315, 2002. 44, 67, 112

[Zor00] Denis Zorin. A method for analysis of C^1-continuity of subdivision surfaces. *SIAM Journal on Numerical Analysis*, 37(5):1677–1708, October 2000. 44

A. Shader Source Code

Listing A.1: *Beispielcode*

```
1  //#############################################################################################
2  // Global variables
3  //#############################################################################################
4  Texture pos0,map,corner,nor0,dmap, nor1;
5  uniform float fValence = -1;
6  uniform float STEP;
7  uniform float4 Table;
8  #define STEP2 2*STEP
9
10 uniform int adjustArray[8];
11 uniform float customScale = 1.0;
12 uniform float distanceScale = 1.0;
13 uniform int dfunction = 1;
14
15 static int index = 1;
16
17 //#############################################################################################
18 // Texture samplers
19 //#############################################################################################
20 sampler Pos0 = sampler_state
21 {
22     Texture = <pos0>;
23 };
24 // ---------------------------------------------------------------------------------------
25 sampler Map = sampler_state
26 {
27     Texture = <map>;
28 };
29 // ---------------------------------------------------------------------------------------
30 sampler Nor0 = sampler_state
31 {
32     Texture = <nor0>;
33 };
34 // ---------------------------------------------------------------------------------------
35 sampler Nor1 = sampler_state
36 {
37     Texture = <nor1>;
38 };
39 // ---------------------------------------------------------------------------------------
40 sampler Corner = sampler_state
41 {
42     Texture = <corner>;
43 };
44 // ---------------------------------------------------------------------------------------
45 sampler DMap = sampler_state
46 {
47     Texture = <dmap>;
48 };
49
50
51 // ---------------------------------------------------------------------------------------
52
53 //#############################################################################################
54 // Vertex Shader
55 //#############################################################################################
56 // ---------------------------------------------------------------------------------------
57 struct VS_INPUT
58 {
59    float4 Pos : POSITION0;
60    float2 Tex0 : TEXCOORD0;
61 };
62 // ---------------------------------------------------------------------------------------
63 struct VS_OUTPUT
64 {
65    float4 Pos : POSITION0;
66    float2 Tex0 : TEXCOORD0;
```

```
67      float2 Tex1 : TEXCOORD1;
68  };
69
70  // ------------------------------------------------------------------------------
71  VS_OUTPUT vs1( VS_INPUT In )
72  {
73      VS_OUTPUT Out = (VS_OUTPUT) 0;
74      Out.Tex1 = In.Tex0;
75
76      In.Pos.xy = sign(In.Pos.xy);
77      Out.Pos = float4(In.Pos.xy, 0.0f, 1.0f);
78      //// get into range [0,1]
79      Out.Tex0 = (float2(Out.Pos.x, -Out.Pos.y) + 1.0f)*(0.5);
80      return Out;
81  }
82
83
84  // ------------------------------------------------------------------------------
85
86  //##############################################################################################
87  // Pixel shader
88  //##############################################################################################
89  // ------------------------------------------------------------------------------
90  struct PS_INPUT
91  {
92      float2 tc : TEXCOORD0;
93      float2 tc1 : TEXCOORD1;
94  };
95  // ------------------------------------------------------------------------------
96  struct PS_OUT
97  {
98          float4 color0 : COLOR0;
99
100 };
101 // ------------------------------------------------------------------------------
102 struct PS_OUT2
103 {
104         float4 color0 : COLOR0;
105         float4 color1 : COLOR1;
106 };
107
108 // ********************************************************************************
109 // global functions
110 // ********************************************************************************
111
112 // ------------------------------------------------------------------------------
113 float4 getEdgePoint_230(float2 tex)
114 {
115     float4  up = tex2D(Pos0, tex+float2(+STEP,0)); // upper vertex
116     float4  lp = tex2D(Pos0, tex+float2(-STEP,0)); // lower vertex
117     float4 ulp = tex2D(Pos0, tex+float2(-STEP,+STEP2)); // upper left
118     float4 llp = tex2D(Pos0, tex+float2(-STEP,-STEP2)); // lower left
119     float4 urp = tex2D(Pos0, tex+float2(+STEP,+STEP2)); // upper right
120     float4 lrp = tex2D(Pos0, tex+float2(+STEP,-STEP2)); // lower right
121
122     float4 v = (0.0625 * ( 6*up + 6*lp + ulp + llp + urp + lrp ));
123     v.w = 0.5 * (up.w + lp.w);
124     return v;
125 }
126
127 // ------------------------------------------------------------------------------
128 float4 getEdgePoint_240(float2 tex)
129 {
130     float4  up = tex2D(Pos0, tex+float2(0,+STEP)); // upper vertex
131     float4  lp = tex2D(Pos0, tex+float2(0,-STEP)); // lower vertex
132     float4 ulp = tex2D(Pos0, tex+float2(-STEP2,+STEP)); // upper left
133     float4 llp = tex2D(Pos0, tex+float2(-STEP2,-STEP)); // lower left
134     float4 urp = tex2D(Pos0, tex+float2(+STEP2,+STEP)); // upper right
135     float4 lrp = tex2D(Pos0, tex+float2(+STEP2,-STEP)); // lower right
136
137     float4 v = (0.0625 * ( 6*up + 6*lp + ulp + llp + urp + lrp ));
138     v.w = 0.5 * (up.w + lp.w);
139     return v;
140 }
141
142 // ------------------------------------------------------------------------------
143 float4 getVertexPoint_200(float4 v, float2 tex)
144 {
145     float4 ep10 = tex2D(Pos0, tex+float2(+STEP2,0)); // right edge point
146     float4 ep01 = tex2D(Pos0, tex+float2(-STEP2,0)); // left ep
147     float4 ep11 = tex2D(Pos0, tex+float2(0,+STEP2)); // upper ep
148     float4 ep00 = tex2D(Pos0, tex+float2(0,-STEP2)); // lower rp
149
```

```
150     float4 fp11 = ( v + ep10 + ep11 + tex2D(Pos0, tex+float2(+STEP2,+STEP2)) );
151     float4 fp10 = ( v + ep10 + ep00 + tex2D(Pos0, tex+float2(+STEP2,-STEP2)) );
152     float4 fp01 = ( v + ep01 + ep11 + tex2D(Pos0, tex+float2(-STEP2,+STEP2)) );
153     float4 fp00 = ( v + ep01 + ep00 + tex2D(Pos0, tex+float2(-STEP2,-STEP2)) );
154
155     return( 0.5 * v + 0.0625 * (ep00 + ep11 + ep01 + ep10) + 0.015625 * (fp00 + fp01 + fp10 + fp11) );
156 }
157
158
159 // --------------------------------------------------------------------------------
160 float3 overrideExtraordinary(float3 v, PS_INPUT In)
161 {
162         //float TableExt = tex2D(Map, In.tc).b;
163         float valStep = 2.0*fValence+1.0;
164         if(Table.b==200.0 && fValence!=-1) // vertex point
165         {
166                 v = tex2D(Corner, float2(0, 0));
167         }
168         // -------- face points
169         if(Table.b==250.0 && fValence!=-1) // proper face point # always total fValence-1
170         {
171                 v = tex2D(Corner, float2(((valStep-1.0)/valStep), 0.0));
172         }
173         if(Table.b==251.0 && fValence!=-1) // upper face point # allways fValence+1
174         {
175                 v = tex2D(Corner, float2(((1.0+fValence)/valStep), 0.0));
176         }
177         if(Table.b==252.0 && fValence!=-1) // left face point # always valStep-2
178         {
179                 v = tex2D(Corner, float2(((valStep-2.0)/valStep), 0.0));
180         }
181         // -------- edge points
182         if(Table.b==235.0 && fValence!=-1) // v edge # always # 1
183         {
184                 v = tex2D(Corner, float2((1.0/valStep), 0.0));
185         }
186         if(Table.b==236.0 && fValence!=-1) // v edge upper, always #2
187         {
188                 v = tex2D(Corner, float2((2.0/valStep), 0.0));
189         }
190         if(Table.b==245.0 && fValence!=-1) // h edge, always fValence
191         {
192                 v = tex2D(Corner, float2((fValence/valStep), 0.0));
193         }
194         if(Table.b==246.0 && fValence!=-1) // h edge left, always fValence-1
195         {
196                 v = tex2D(Corner, float2(((fValence-1.0)/valStep), 0.0));
197         }
198         return v;
199 }
200
201
202
203 // --------------------------------------------------------------------------------
204 float3x3 subdivideVertexTexNormal(PS_INPUT In)
205 {
206         float4 v;
207         float3 n;
208         float2 t;
209
210         if(Table.a==200.0)
211         {
212                 v = tex2D(Pos0,In.tc);
213             t.x = v.w;
214                 t.y = tex2D( Nor0, In.tc ).w;
215                 v = getVertexPoint_200(v, In.tc);
216                 n = tex2D( Nor1, In.tc).xyz;
217         }
218         if(Table.a==215.0)
219         {
220                 v = tex2D(Pos0, In.tc);
221                 n = tex2D( Nor1, In.tc).xyz;
222                 t.x = v.w;
223                 t.y = tex2D( Nor0, In.tc ).w;
224         }
225         if(Table.a==205.0) //vertex border horizontal
226         {
227                 v = tex2D(Pos0,In.tc);
228                 t.x = v.w;
229                 t.y = tex2D( Nor0, In.tc ).w;
230                 float4 ep1 = tex2D(Pos0, In.tc+float2(+STEP2,0));
231                 float4 ep2 = tex2D(Pos0, In.tc+float2(-STEP2,0));
232                 v = 0.75*v + 0.125*(ep1+ep2);
```

```
233                 n = tex2D( Nor1, In.tc).xyz;
234         }
235         if(Table.a==210.0) //vertex, border, vert
236         {
237                 v = tex2D(Pos0,In.tc);
238                 t.x = v.w;
239                 t.y = tex2D( Nor0, In.tc ).w;
240                 float4 ep1 = tex2D(Pos0, In.tc+float2(0,+STEP2));
241                 float4 ep2 = tex2D(Pos0, In.tc+float2(0,-STEP2));
242                 v = 0.75*v + 0.125*(ep1+ep2);
243                 n = tex2D( Nor1, In.tc).xyz;
244         }
245         if(Table.a==230.0) //h-ep
246         {
247                 v = getEdgePoint_230(In.tc);
248                 n = 0.5 * (tex2D( Nor1, In.tc+float2(-STEP,0)).xyz + tex2D( Nor1, In.tc+float2(+STEP,0)).xyz);
249                 t.x = v.w;
250                 t.y = 0.5 * ( tex2D(Nor0, In.tc+float2(+STEP,0)).w + tex2D(Nor0, In.tc+float2(-STEP,0)).w );
251         }
252         if(Table.a==240.0) //v-ep
253         {
254                 v = getEdgePoint_240(In.tc);
255                 n = 0.5 * (tex2D( Nor1, In.tc+float2(0,-STEP)).xyz + tex2D( Nor1, In.tc+float2(0,+STEP)).xyz);
256                 t.x = v.w;
257                 t.y = 0.5 * ( tex2D(Nor0, In.tc+float2(0,+STEP)).w + tex2D(Nor0, In.tc+float2(0,-STEP)).w );
258         }
259         if(Table.a==235.0)
260         {
261                 v = 0.5 * (tex2D(Pos0,In.tc+float2(-STEP,0)) + tex2D(Pos0,In.tc+float2(+STEP,0)));
262                 n = 0.5 * (tex2D( Nor1, In.tc+float2(-STEP,0)).xyz + tex2D( Nor1, In.tc+float2(+STEP,0)).xyz);
263                 t.x = v.w;
264                 t.y = 0.5 * ( tex2D(Nor0, In.tc+float2(+STEP,0)).w + tex2D(Nor0, In.tc+float2(-STEP,0)).w );
265         }
266         if(Table.a==245.0)
267         {
268                 v = 0.5 * (tex2D(Pos0,In.tc+float2(0,-STEP)) + tex2D(Pos0,In.tc+float2(0,+STEP)));
269                 n = 0.5 * (tex2D( Nor1, In.tc+float2(0,-STEP)).xyz + tex2D( Nor1, In.tc+float2(0,+STEP)).xyz);
270                 t.x = v.w;
271                 t.y = 0.5 * ( tex2D(Nor0, In.tc+float2(0,+STEP)).w + tex2D(Nor0, In.tc+float2(0,-STEP)).w );
272         }
273         if(Table.a==250.0)
274         {
275                 v = 0.25 * (
276                           tex2D(Pos0, In.tc+float2(-STEP,-STEP)) + tex2D(Pos0, In.tc+float2(-STEP,+STEP))
277                         + tex2D(Pos0, In.tc+float2(+STEP,-STEP)) + tex2D(Pos0, In.tc+float2(+STEP,+STEP)));
278
279                 n = 0.25 * (
280                           tex2D( Nor1, In.tc+float2(-STEP,-STEP)).xyz + tex2D( Nor1, In.tc+float2(+STEP,-STEP)).xyz
281                    + tex2D( Nor1, In.tc+float2(-STEP,+STEP)).xyz + tex2D( Nor1, In.tc+float2(+STEP,+STEP)).xyz);
282                 t.x = v.w;
283                 t.y = 0.25 * (
284                           tex2D(Nor0, In.tc+float2(+STEP,+STEP)).w + tex2D(Nor0, In.tc+float2(+STEP,-STEP)).w
285                         + tex2D(Nor0, In.tc+float2(-STEP,+STEP)).w + tex2D(Nor0, In.tc+float2(-STEP,-STEP)).w );
286         }
287                 float3x3 vnt = {
288                 v.x,  v.y,  v.z,
289                 n.x,  n.y,  n.z,
290                 t.x,  t.y,  0,  };
291
292         return vnt;
293 }
294
295
296 // --------------------------------------------------------------------------------
297 float3x3 subdivideVertexTex(PS_INPUT In)
298 {
299         float4 v;
300         float2 t;
301
302         if(Table.a==200.0)
303         {
304                 v = tex2D(Pos0,In.tc);
305             t.x = v.w;
306                 t.y = tex2D( Nor0, In.tc ).w;
307                 v = getVertexPoint_200(v, In.tc);
308         }
309         if(Table.a==215.0)
310         {
311                 v = tex2D(Pos0, In.tc);
312                 t.x = v.w;
313                 t.y = tex2D( Nor0, In.tc ).w;
314         }
315         if(Table.a==205.0) //vertex border horizontal
```

```
316         {
317                 v = tex2D(Pos0,In.tc);
318                 t.x = v.w;
319                 t.y = tex2D( Nor0, In.tc ).w;
320                 float4 ep1 = tex2D(Pos0, In.tc+float2(+STEP2,0));
321                 float4 ep2 = tex2D(Pos0, In.tc+float2(-STEP2,0));
322                 v = 0.75*v + 0.125*(ep1+ep2);
323         }
324         if(Table.a==210.0) //vertex, border, vert
325         {
326                 v = tex2D(Pos0,In.tc);
327                 t.x = v.w;
328                 t.y = tex2D( Nor0, In.tc ).w;
329                 float4 ep1 = tex2D(Pos0, In.tc+float2(0,+STEP2));
330                 float4 ep2 = tex2D(Pos0, In.tc+float2(0,-STEP2));
331                 v = 0.75*v + 0.125*(ep1+ep2);
332         }
333         if(Table.a==230.0) //h-ep
334         {
335                 v = getEdgePoint_230(In.tc);
336                 t.x = v.w;
337                 t.y = 0.5 * ( tex2D(Nor0, In.tc+float2(+STEP,0)).w + tex2D(Nor0, In.tc+float2(-STEP,0)).w );
338         }
339         if(Table.a==240.0) //v-ep
340         {
341                 v = getEdgePoint_240(In.tc);
342                 t.x = v.w;
343                 t.y = 0.5 * ( tex2D(Nor0, In.tc+float2(0,+STEP)).w + tex2D(Nor0, In.tc+float2(0,-STEP)).w );
344         }
345         if(Table.a==235.0)
346         {
347                 v = 0.5 * (tex2D(Pos0,In.tc+float2(-STEP,0)) + tex2D(Pos0,In.tc+float2(+STEP,0)));
348                 t.x = v.w;
349                 t.y = 0.5 * ( tex2D(Nor0, In.tc+float2(+STEP,0)).w + tex2D(Nor0, In.tc+float2(-STEP,0)).w );
350         }
351         if(Table.a==245.0)
352         {
353                 v = 0.5 * (tex2D(Pos0,In.tc+float2(0,-STEP)) + tex2D(Pos0,In.tc+float2(0,+STEP)));
354                 t.x = v.w;
355                 t.y = 0.5 * ( tex2D(Nor0, In.tc+float2(0,+STEP)).w + tex2D(Nor0, In.tc+float2(0,-STEP)).w );
356         }
357         if(Table.a==250.0)
358         {
359                 v = 0.25 * (
360                           tex2D(Pos0, In.tc+float2(-STEP,-STEP)) + tex2D(Pos0, In.tc+float2(-STEP,+STEP))
361                         + tex2D(Pos0, In.tc+float2(+STEP,-STEP)) + tex2D(Pos0, In.tc+float2(+STEP,+STEP)));
362                 t.x = v.w;
363                 t.y = 0.25 * (
364                           tex2D(Nor0, In.tc+float2(+STEP,+STEP)).w + tex2D(Nor0, In.tc+float2(+STEP,-STEP)).w
365                         + tex2D(Nor0, In.tc+float2(-STEP,+STEP)).w + tex2D(Nor0, In.tc+float2(-STEP,-STEP)).w );
366         }
367                 float3x3 vnt = {
368                 v.x,  v.y,  v.z,
369                 0,  0,  0,
370                 t.x,  t.y,  0,  };
371
372         return vnt;
373 }
374 // ------------------------------------------------------------------------------
375 float3 adjustBorders_1(float3 v, PS_INPUT In)
376 {
377         bool passThru = 0;
378         passThru  = (adjustArray[0]==1)*(Table.g==1.0 || Table.g==4.0  || Table.g==6.0);
379         passThru += (adjustArray[1]==1)*(Table.g==5.0 || Table.g==6.0  || Table.g==12.0);
380         passThru += (adjustArray[2]==1)*(Table.g==7.0 || Table.g==12.0 || Table.g==10.0);
381         passThru += (adjustArray[3]==1)*(Table.g==3.0 || Table.g==10.0 || Table.g==4.0);
382         passThru += (adjustArray[4]==1)*(Table.g==4.0);
383         passThru += (adjustArray[5]==1)*(Table.g==6.0);
384         passThru += (adjustArray[6]==1)*(Table.g==12.0);
385         passThru += (adjustArray[7]==1)*(Table.g==10.0);
386
387         if(Table.a==200.0 && passThru)
388         {
389                 v.xyz = tex2D(Pos0, In.tc).xyz;
390         }
391         if(Table.a==215.0 && passThru)
392         {
393                 v.xyz = tex2D(Pos0, In.tc).xyz;
394         }
395         if(Table.a==205.0 && passThru)
396         {
397                 v.xyz = tex2D(Pos0, In.tc).xyz;
398         }
```

```
399         if(Table.a==210.0 && passThru)
400         {
401                 v.xyz = tex2D(Pos0, In.tc).xyz;
402         }
403         if(Table.a==230.0 && passThru)
404         {
405                 v.xyz = 0.5* (tex2D(Pos0,In.tc+float2(-STEP,0)) + tex2D(Pos0,In.tc+float2(+STEP,0))).xyz;
406         }
407         if(Table.a==240.0 && passThru)
408         {
409                 v.xyz = 0.5* (tex2D(Pos0,In.tc+float2(0,-STEP)) + tex2D(Pos0,In.tc+float2(0,+STEP))).xyz;
410         }
411         if(Table.a==235.0 && passThru)
412         {
413                 v.xyz = 0.5* (tex2D(Pos0,In.tc+float2(-STEP,0)) + tex2D(Pos0,In.tc+float2(+STEP,0))).xyz;
414         }
415         if(Table.a==245.0 && passThru)
416         {
417                 v.xyz = 0.5* (tex2D(Pos0,In.tc+float2(0,-STEP)) + tex2D(Pos0,In.tc+float2(0,+STEP))).xyz;
418         }
419         return v;
420 }
421
422
423 // ------------------------------------------------------------------------------
424 float3 adjustBorders_2(float3 v, PS_INPUT In)
425 {
426         bool passThru = 0;
427         passThru  = (adjustArray[0]==1)*(Table.g==1.0 || Table.g==4.0  || Table.g==6.0);
428         passThru += (adjustArray[1]==1)*(Table.g==5.0 || Table.g==6.0  || Table.g==12.0);
429         passThru += (adjustArray[2]==1)*(Table.g==7.0 || Table.g==12.0 || Table.g==10.0);
430         passThru += (adjustArray[3]==1)*(Table.g==3.0 || Table.g==10.0 || Table.g==4.0);
431         passThru += (adjustArray[4]==1)*(Table.g==4.0);
432         passThru += (adjustArray[5]==1)*(Table.g==6.0);
433         passThru += (adjustArray[6]==1)*(Table.g==12.0);
434         passThru += (adjustArray[7]==1)*(Table.g==10.0);
435
436         if(Table.a==200.0 && passThru)
437         {
438                 v.xyz = tex2D(Pos0, In.tc).xyz;
439         }
440         if(Table.a==215.0 && passThru)
441         {
442                 v.xyz = tex2D(Pos0, In.tc).xyz;
443         }
444         if(Table.a==205.0 && passThru)
445         {
446                 v.xyz = tex2D(Pos0, In.tc).xyz;
447         }
448         if(Table.a==210.0 && passThru)
449         {
450                 v.xyz = tex2D(Pos0, In.tc).xyz;
451         }
452         return v;
453 }
454
455
456
457 // ------------------------------------------------------------------------------
458 float3 displacement(float3 v, float3 n, float2 t)
459 {
460         float4 off = tex2D(DMap, t);
461         float scale = (((off.x+off.y+off.z)/3.0)-0.5);//*Table.r;
462         v += (dfunction==1) * float4(n.xyz,1)*scale*distanceScale*customScale*(STEP); //0.04;//*STEP2;
463         v += (dfunction==2) * float4(n.xyz,1)*scale*distanceScale*customScale*0.1;//(STEP*STEP); ;//0.04;//*STEP2;
464         v += (dfunction==3) * float4(n.xyz,1)*scale*distanceScale*customScale*(STEP*STEP); //0.04;//*STEP2;
465         v += (dfunction==4) * float4(n.xyz,1)*scale*distanceScale*customScale*sqrt(STEP); //0.04;//*STEP2;
466
467         return v;
468 }
469
470
471 // ********************************************************************************
472 // main, pass111: subdivision
473 // ********************************************************************************
474 PS_OUT2 ps_main111(PS_INPUT In)
475 {
476         Table = tex2D(Map, In.tc);
477         float3 v = 0;
478         float2 t = 0;
479
480         float3x3 vnt = subdivideVertexTex(In);
481         v = vnt[0];
```

```
482          t = vnt[2].xy;
483
484      v = overrideExtraordinary(v,In);
485
486          PS_OUT2 Out;
487          Out.color0.xyz = v;
488          Out.color1.xyz = 0;
489          Out.color0.w = t.x;
490          Out.color1.w = t.y;
491          return Out;
492  }
493
494
495  // ************************************************************************************
496  // main, pass112: subdivision with border adjustment
497  // ************************************************************************************
498  PS_OUT2 ps_main112(PS_INPUT In)
499  {
500          Table = tex2D(Map, In.tc);
501          float3 v = 0;
502          float2 t = 0;
503
504          float3x3 vnt = subdivideVertexTex(In);
505          v = vnt[0];
506          t = vnt[2].xy;
507
508          v = adjustBorders_1(v,In);
509
510      v = overrideExtraordinary(v,In);
511
512          PS_OUT2 Out;
513          Out.color0.xyz = v;
514          Out.color1.xyz = 0;
515          Out.color0.w = t.x;
516          Out.color1.w = t.y;
517          return Out;
518  }
519
520
521  // ************************************************************************************
522  // main, pass121: subdivision + displacement
523  // ************************************************************************************
524  PS_OUT2 ps_main121(PS_INPUT In)
525  {
526          Table = tex2D(Map, In.tc);
527          float3 v = 0;
528          float3 n = 0;
529          float2 t = 0;
530
531          float3x3 vnt = subdivideVertexTexNormal(In);
532          v = vnt[0];
533          n = vnt[1];
534          t = vnt[2].xy;
535
536          v = displacement(v, n, t);
537
538      v = overrideExtraordinary(v,In);
539
540          PS_OUT2 Out;
541
542          Out.color0.xyz = v;
543          Out.color1.xyz = 0;
544          Out.color0.w = t.x;
545          Out.color1.w = t.y;
546          return Out;
547  }
548
549  // ************************************************************************************
550  // main, pass122: subdivision + displacement with border adjustment
551  // ************************************************************************************
552  PS_OUT2 ps_main122(PS_INPUT In)
553  {
554          Table = tex2D(Map, In.tc);
555          float3 v = 0;
556          float3 n = 0;
557          float2 t = 0;
558
559          float3x3 vnt = subdivideVertexTexNormal(In);
560          v = vnt[0];
561          n = vnt[1];
562          t = vnt[2].xy;
563
564          v = displacement(v, n, t);
```

```
565
566         v = adjustBorders_2(v,In).xyz;
567
568         v = overrideExtraordinary(v,In).xyz;
569
570         PS_OUT2 Out;
571         Out.color0.xyz = v;
572         Out.color1.xyz = 0;
573         Out.color0.w = t.x;
574         Out.color1.w = t.y;
575         return Out;
576 }
577
578
579
580
581
582 // ********************************************************************************
583 // main, pass2: normal calculation
584 // ********************************************************************************
585 PS_OUT2 ps_main22(PS_INPUT In)
586 {
587         float3 n = 0;
588         PS_OUT2 Out;
589
590         float3 v1,v2,v3,v4;
591         float4 v = tex2D(Pos0, In.tc);
592         float2 t = float2( tex2D(Pos0, In.tc).w, tex2D(Nor0, In.tc).w );
593
594         //calculate normal
595         v1 = tex2D(Pos0, In.tc+float2(+STEP,0)).xyz - v.xyz;
596         v3 = tex2D(Pos0, In.tc+float2(-STEP,0)).xyz - v.xyz;
597         v2 = tex2D(Pos0, In.tc+float2(0,+STEP)).xyz - v.xyz;
598         v4 = tex2D(Pos0, In.tc+float2(0,-STEP)).xyz - v.xyz;
599
600         n = cross(v1,v2);
601         n += cross(v2,v3);
602         n += cross(v3,v4);
603         n += cross(v4,v1);
604
605         Out.color0.xyz = v.xyz;
606         Out.color1.xyz = normalize(n);
607         Out.color0.w = t.x;//tex2D(Pos0, In.tc+float2(-0,+0)).w;// /255.0;//In.tc1.x;
608         Out.color1.w = t.y;//tex2D(Nor0, In.tc+float2(-HALF,+0)).w;// /255.0;//In.tc1.y;
609
610         return Out;
611 }
612
613
614 // ********************************************************************************
615 // main, pass3 : extraodrinary handling
616 // ********************************************************************************
617 float4 ps_main3(float2 tex : TEXCOORD0) : COLOR0 {
618 float mystep= (1.0/ (6.0));
619 float4 v = 0;
620 float map = tex2D(Corner, tex).a;
621 float3 p = tex2D(Corner, float2(0.0 ,0.0)).xyz;
622 float3 p1 = tex2D(Corner, float2(1*mystep,0.0)).xyz;
623 float3 p2 = tex2D(Corner, float2(2*mystep,0.0)).xyz;
624 float3 p3 = tex2D(Corner, float2(3*mystep,0.0)).xyz;
625 //float3 vp12 = tex2D(Corner, float2(4*mystep,0.0)).xyz;
626 //float3 vp23 = tex2D(Corner, float2(5*mystep,0.0)).xyz;
627 //float3 vp31 = tex2D(Corner, float2(6*mystep,0.0)).xyz;
628 float3 fp12 = 0.25 * (p + p1 + p2 + tex2D(Corner, float2(4*mystep,0.0)).xyz);
629 float3 fp23 = 0.25 * (p + p2 + p3 + tex2D(Corner, float2(5*mystep,0.0)).xyz);
630 float3 fp31 = 0.25 * (p + p3 + p1 + tex2D(Corner, float2(6*mystep,0.0)).xyz);
631 if(map==1.0) v = float4( 0.25 * (p + p1 + fp31 + fp12) , 1.0);
632 if(map==2.0) v = float4( 0.25 * (p + p2 + fp12 + fp23) , 2.0);
633 if(map==3.0) v = float4( 0.25 * (p + p3 + fp23 + fp31) , 3.0);
634 if(map==4.0) v = float4( fp12 , 4.0);
635 if(map==5.0) v = float4( fp23 , 5.0);
636 if(map==6.0) v = float4( fp31 , 6.0);
637 if(map==0.0) v = float4(0.3333333 * p + 0.1111111 * ( 0  + p1 +  fp12 + p2 +  fp23 + p3 +  fp31), 0.0 );
638 return v;}
639
640 float4 ps_main5(float2 tex : TEXCOORD0) : COLOR0 {
641 float mystep= (1.0/ (10.0));
642 float4 v = 0;
643 float map = tex2D(Corner, tex).a;
644 float3 p = tex2D(Corner, float2(0.0 ,0.0)).xyz;
645 float3 p1 = tex2D(Corner, float2(1*mystep,0.0)).xyz;
646 float3 p2 = tex2D(Corner, float2(2*mystep,0.0)).xyz;
647 float3 p3 = tex2D(Corner, float2(3*mystep,0.0)).xyz;
```

```
float3 p4 = tex2D(Corner, float2(4*mystep,0.0)).xyz;
float3 p5 = tex2D(Corner, float2(5*mystep,0.0)).xyz;
float3 vp12 = tex2D(Corner, float2(6*mystep,0.0)).xyz;
float3 vp23 = tex2D(Corner, float2(7*mystep,0.0)).xyz;
float3 vp34 = tex2D(Corner, float2(8*mystep,0.0)).xyz;
float3 vp45 = tex2D(Corner, float2(9*mystep,0.0)).xyz;
float3 vp51 = tex2D(Corner, float2(10*mystep,0.0)).xyz;
float3 fp12 = 0.25 * (p + p1 + p2 + vp12);
float3 fp23 = 0.25 * (p + p2 + p3 + vp23);
float3 fp34 = 0.25 * (p + p3 + p4 + vp34);
float3 fp45 = 0.25 * (p + p4 + p5 + vp45);
float3 fp51 = 0.25 * (p + p5 + p1 + vp51);
if(map==1.0) v = float4( 0.25 * (p + p1 + fp51 + fp12) , 1.0);
if(map==2.0) v = float4( 0.25 * (p + p2 + fp12 + fp23) , 2.0);
if(map==3.0) v = float4( 0.25 * (p + p3 + fp23 + fp34) , 3.0);
if(map==4.0) v = float4( 0.25 * (p + p4 + fp34 + fp45) , 4.0);
if(map==5.0) v = float4( 0.25 * (p + p5 + fp45 + fp51) , 5.0);
if(map==6.0) v = float4( fp12 , 6.0);
if(map==7.0) v = float4( fp23 , 7.0);
if(map==8.0) v = float4( fp34 , 8.0);
if(map==9.0) v = float4( fp45 , 9.0);
if(map==10.0) v = float4( fp51 , 10.0);
if(map==0.0) v = float4(0.6 * p + 0.04 * ( 0  + p1 +  fp12 + p2 +  fp23 + p3 +  fp34 + p4 +  fp45 + p5 +
fp51), 0.0 );
return v;}



//#####################################################################################
// Techniques
//#####################################################################################
technique Subdivide
{
        // #0 ----------------------------------------
        pass Pass111 // subdiv
    {
        VertexShader = compile vs_3_0 vs1();
        PixelShader =  compile ps_3_0 ps_main111();
    }
        // #1 ----------------------------------------
        pass Pass121 // subdiv + borders
    {
        VertexShader = compile vs_3_0 vs1();
        PixelShader =  compile ps_3_0 ps_main112();
    }
        // #2 ----------------------------------------
        pass Pass112 //subdiv + displacement
    {
        VertexShader = compile vs_3_0 vs1();
        PixelShader =  compile ps_3_0 ps_main121();
    }
        // #3 ----------------------------------------
        pass Pass122 // subdiv + displacement + borders
    {
        VertexShader = compile vs_3_0 vs1();
        PixelShader =  compile ps_3_0 ps_main122();
    }
        // #4 ----------------------------------------
        pass Pass2 // normals
    {
        VertexShader = compile vs_3_0 vs1();
        PixelShader =  compile ps_3_0 ps_main22();
    }

        // extraordinary vertices
        // #5 ----------------------------------------
        pass Pass4 //3
        {
                VertexShader = compile vs_3_0 vs1();
        PixelShader =  compile ps_3_0 ps_main3();
        }
        // #6 ----------------------------------------
        pass Pass5
        {
                VertexShader = compile vs_3_0 vs1();
        PixelShader =  compile ps_3_0 ps_main5();
        }
        // #7 ----------------------------------------
        pass Pass6
        {
                VertexShader = compile vs_3_0 vs1();
```

```
730                 PixelShader =  compile ps_3_0 ps_main6();
731         }
732         // #8 --------------------------------------
733         pass Pass7
734         {
735                 VertexShader = compile vs_3_0 vs1();
736         PixelShader =  compile ps_3_0 ps_main7();
737         }
738         // #9 --------------------------------------
739         pass Pass8
740         {
741                 VertexShader = compile vs_3_0 vs1();
742         PixelShader =  compile ps_3_0 ps_main8();
743         }
744 }
745
746
747 // ------------------------------------------------------------------------------
```

B. Color Plates

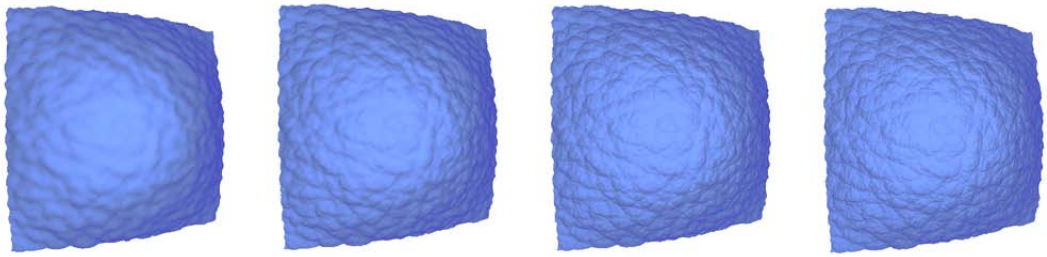


Figure B.1.: *A patch with displacement taken form the leather function (Figure B.2). From left to right: 6 up to 9 subdivision steps with increasing details resolution.*

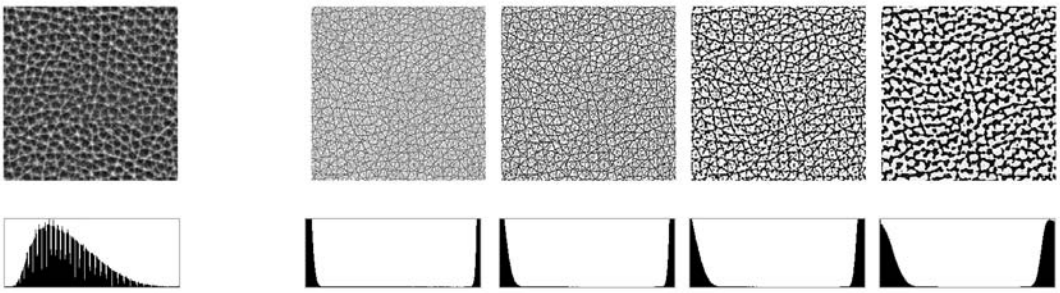


Figure B.2.: *Left: Texture for dakota leather surface structure. Right: First four frequency bands of the displacement map in descending order extracted out of the texture.*

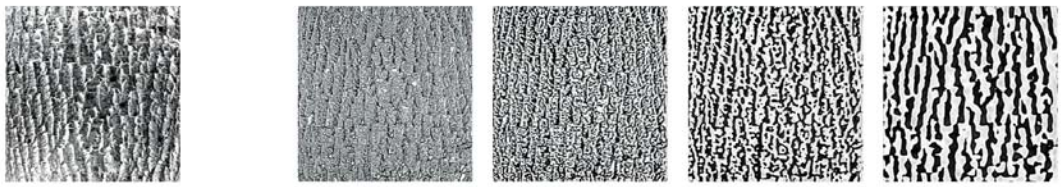


Figure B.3.: *Left: Texture for tree bark surface structure.. Right: First four frequency bands of the displacement map in descending order extracted out of the texture.*

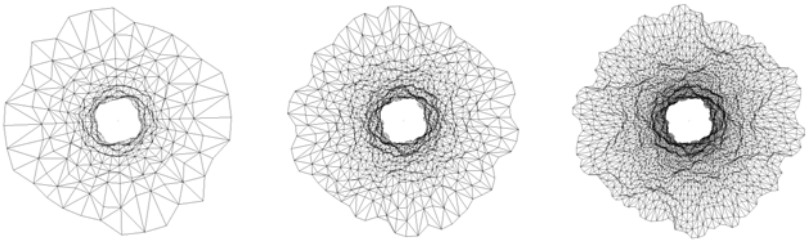


Figure B.4.: *A cylinder model subdivided in hardware with MR-displacement applied.*

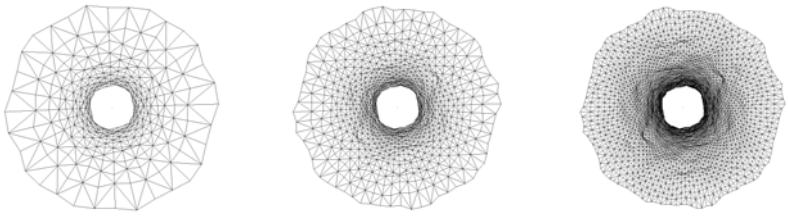


Figure B.5.: *A cylinder model subdivided in hardware with non-filtered displacement applied.*

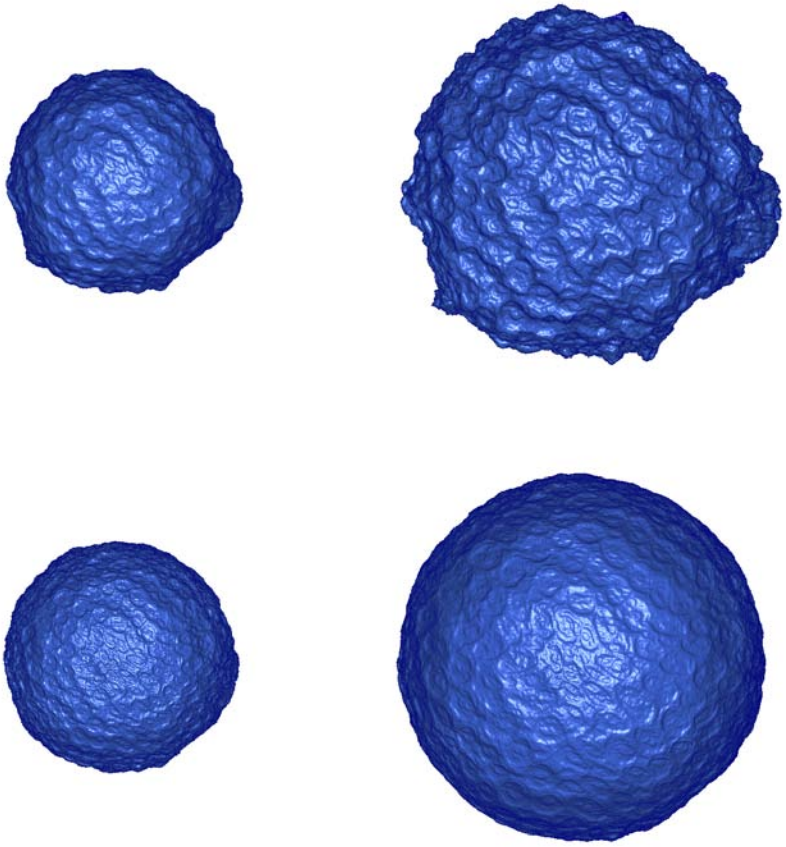


Figure B.6.: *Global versus local offset reference. Top: Euclidean, bottom: parametric local area has been used to reference the offset.*

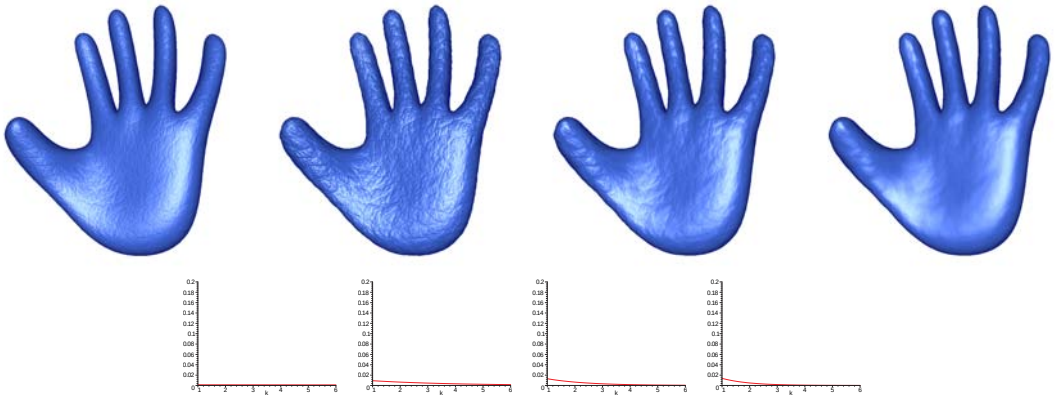


Figure B.7.: *Hand model with* $A = 0.2$*, left to right and* $B = 0.05$*,* $B = 0.5$*,* $B = 1.0$*,* $B = 1.5$

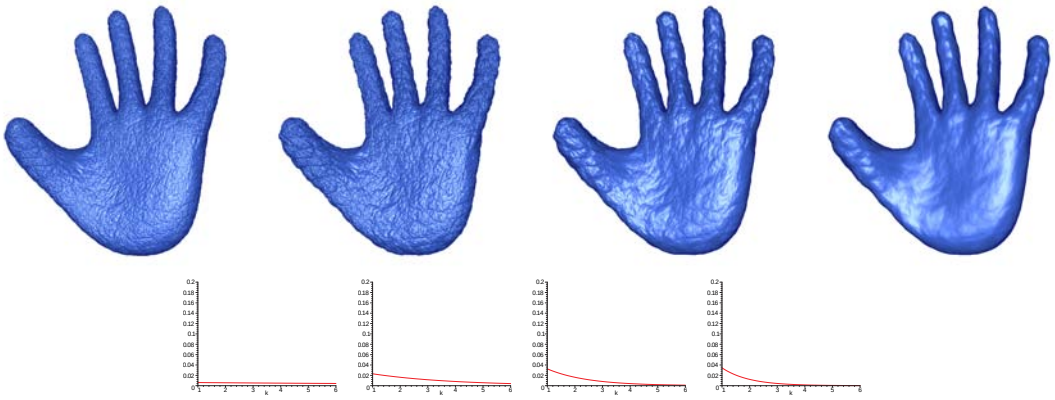


Figure B.8.: *Hand model with* $A = 0.5$*, left to right and* $B = 0.1$*,* $B = 0.5$*,* $B = 1.0$*,* $B = 1.5$

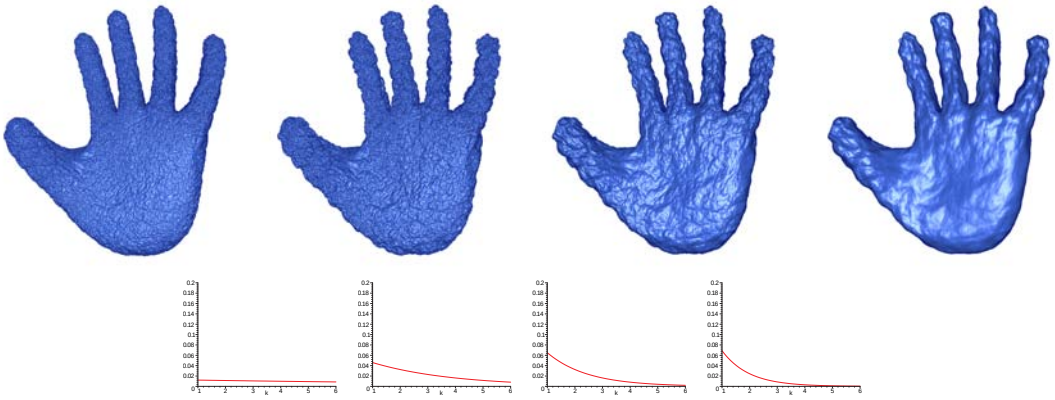


Figure B.9.: *Hand model with* $A = 1$*, left to right and* $B = 0.1$*,* $B = 0.5$*,* $B = 1.0$*,* $B = 1.5$

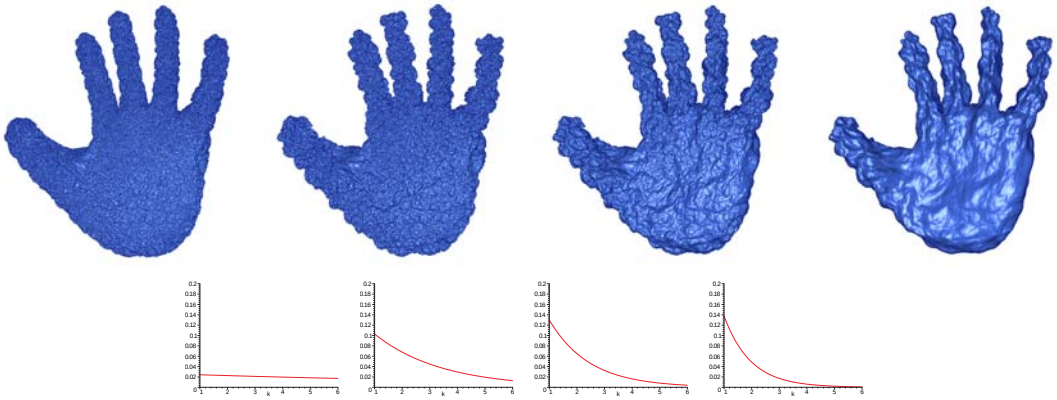


Figure B.10.: *Hand model with* $A = 2$*, left to right and* $B = 0.1$*,* $B = 0.6$*,* $B = 1.0$*,* $B = 1.5$

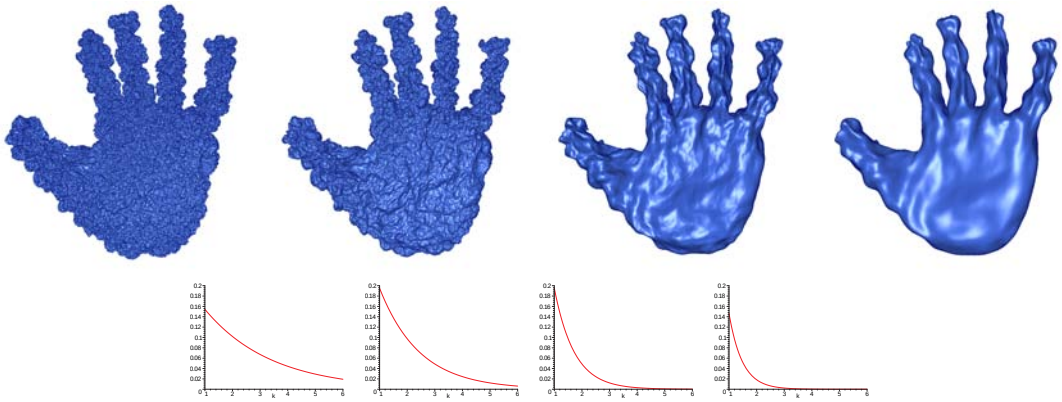


Figure B.11.: *Hand model with* $A = 3$*, left to right and* $B = 0.6$*,* $B = 1.0$*,* $B = 2.0$*,* $B = 3.0$

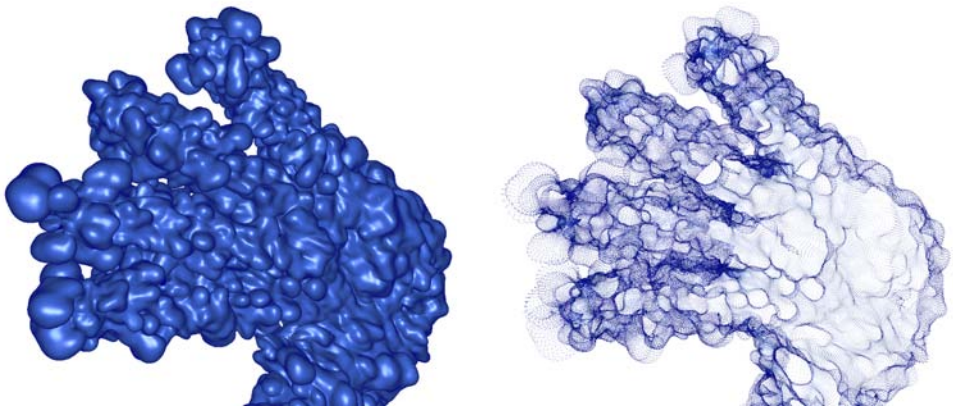


Figure B.12.: *Local self-intersection preserving displacement bounded by surface curvature.*

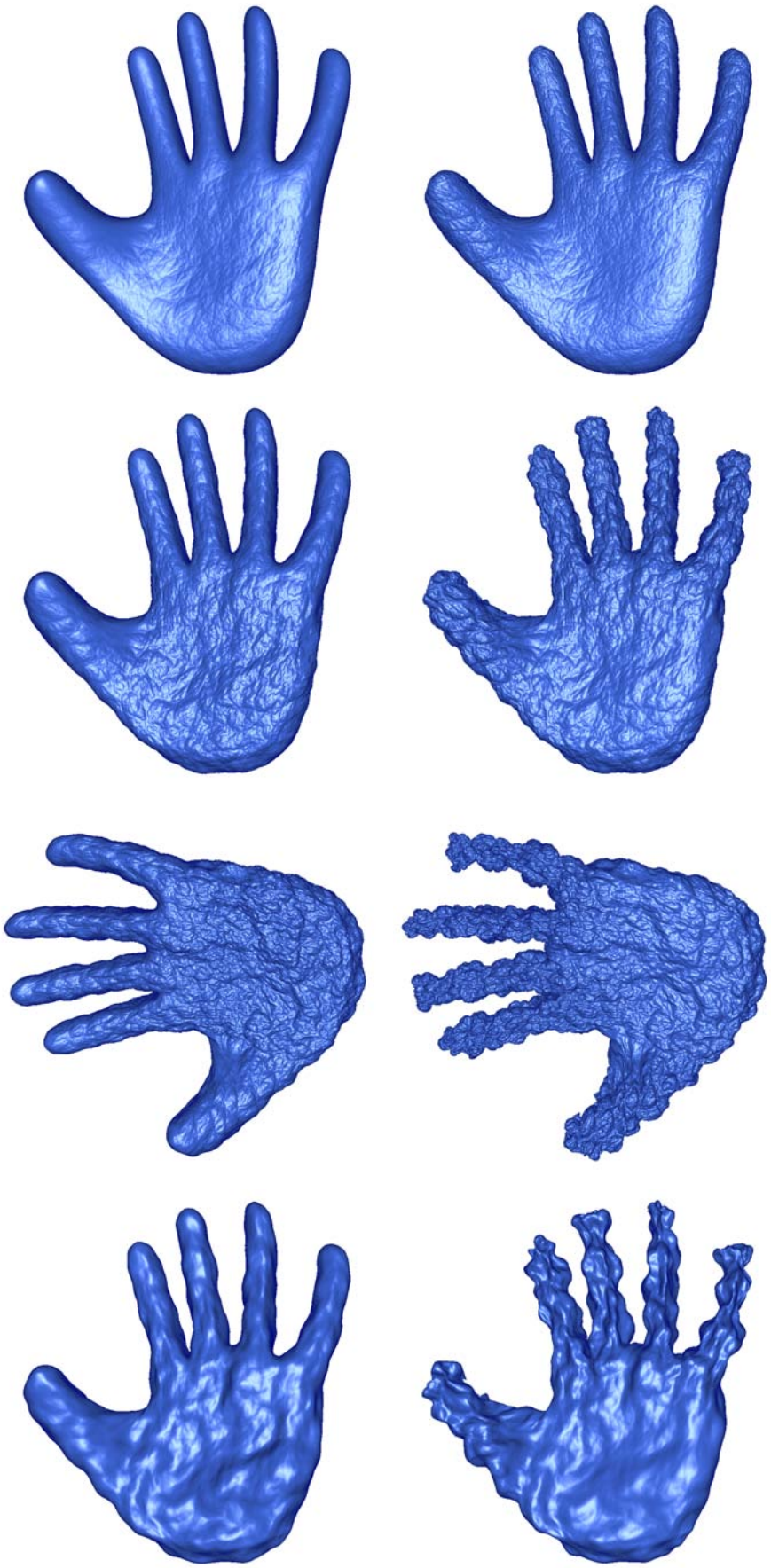


Figure B.13.: *Comparison of the scaling attached to the local area (left column) and a static value which is lean on the average face area of the initial mesh (right column).*

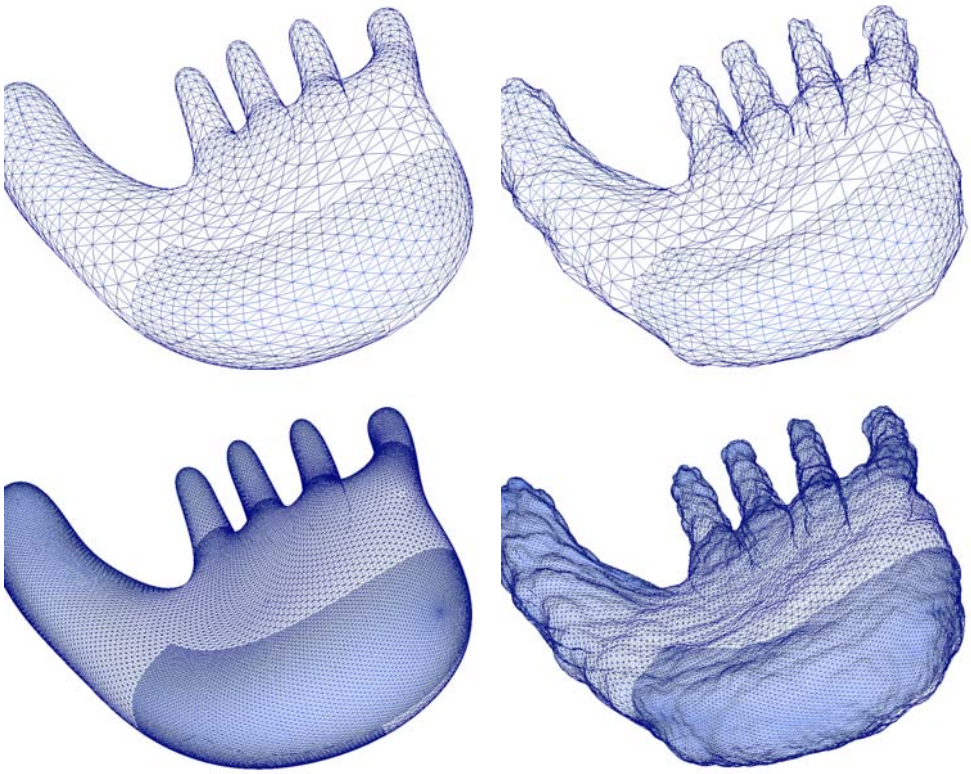


Figure B.14.: *Adaptive Subdivision.*

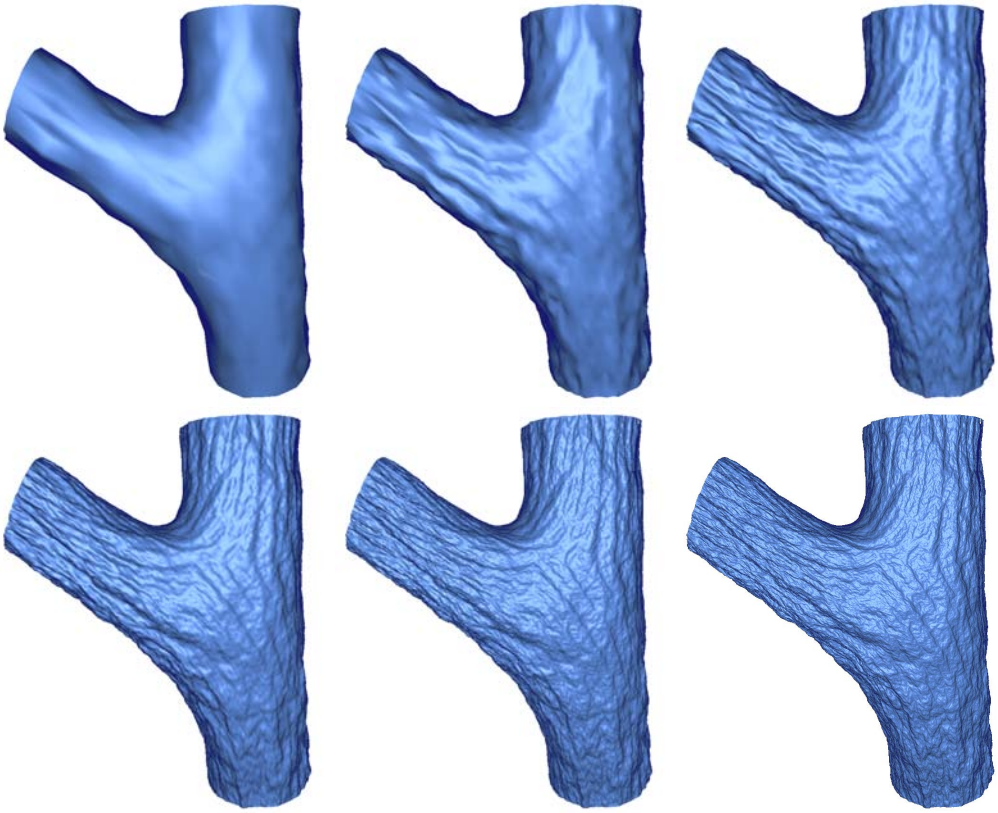


Figure B.15.: *Tree Bark. Recursive subdivision in the GPU after two initial software subdivision steps.*

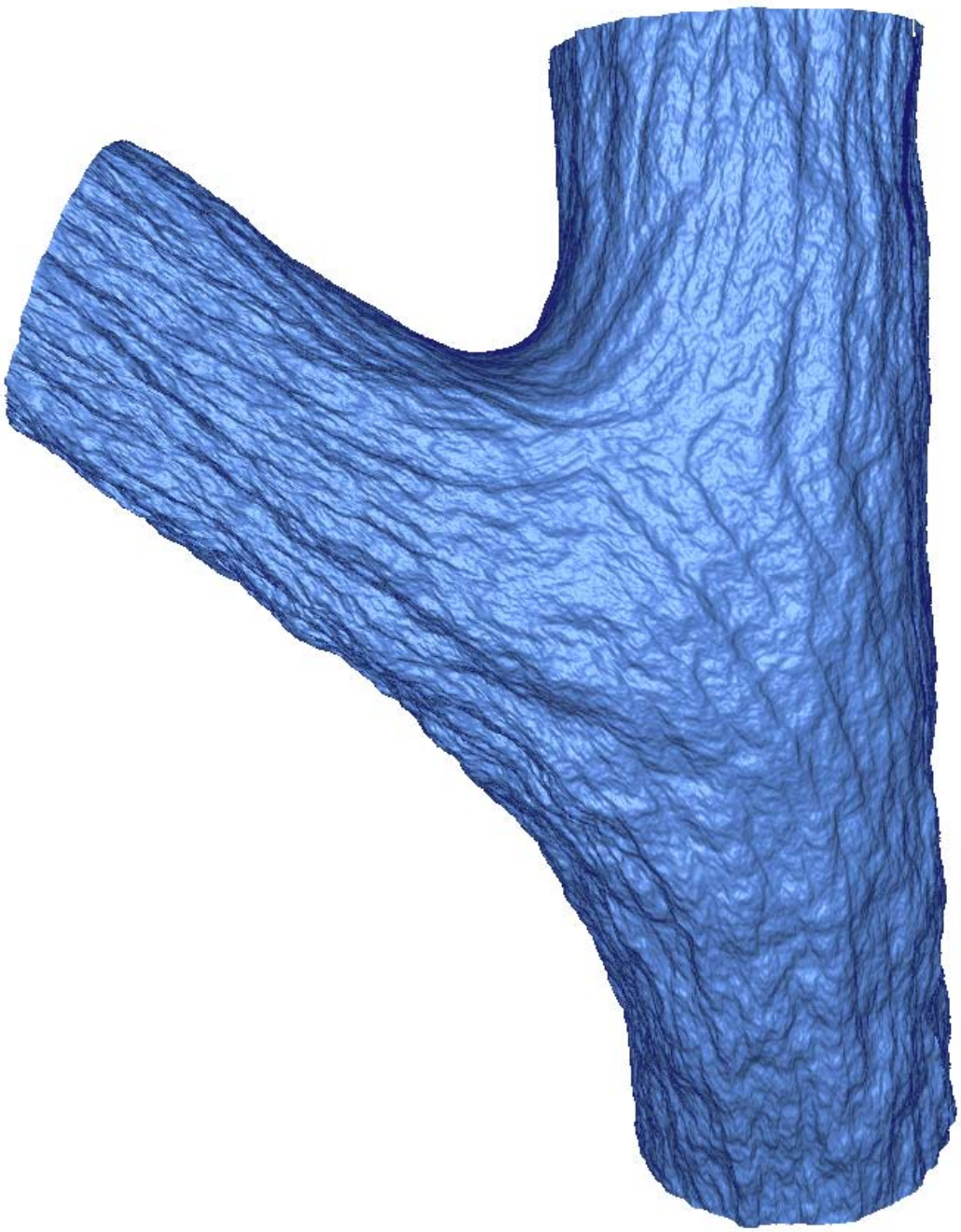


Figure B.16.: *Tree Bark. Software SD: 2, hardware SD: 6.*

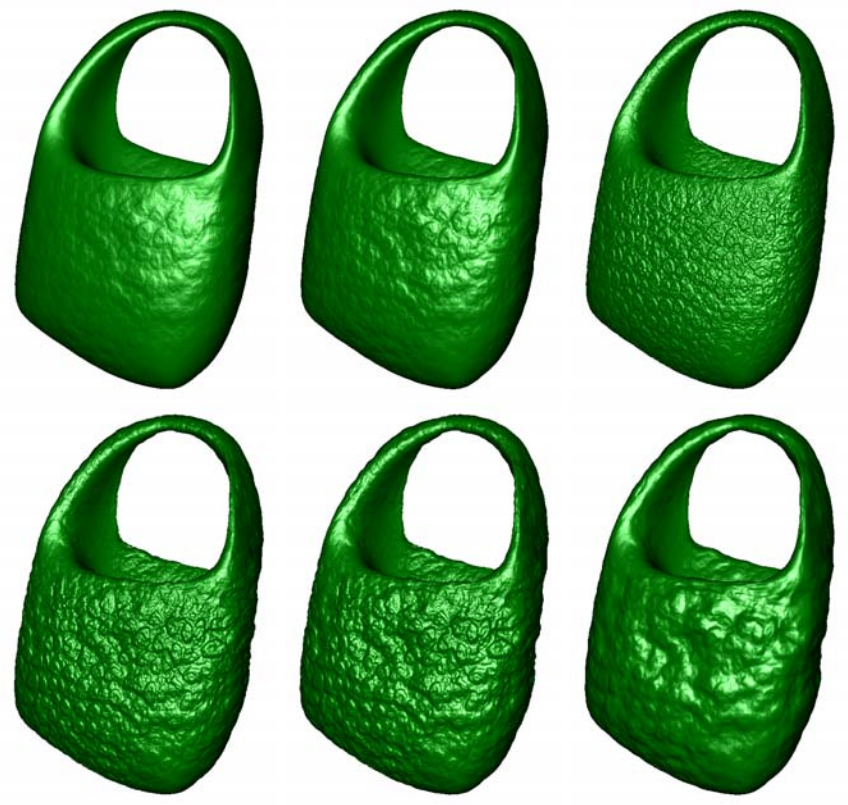


Figure B.17.: *Lady Bag with alien skin.*

Figure B.18.: *Lady Bag with dakota leather.*

Figure B.19.: *Initial control mesh of'The Hand'.*

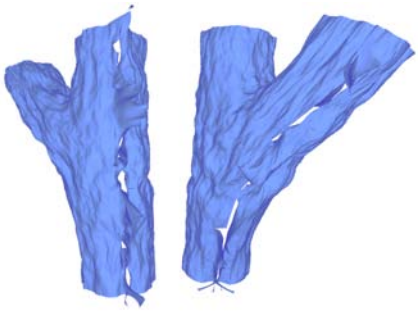


Figure B.20.: *Discontinuities in parametric space.*

Figure B.21.: *Lady Bag with Alligator Leather displacement map.*

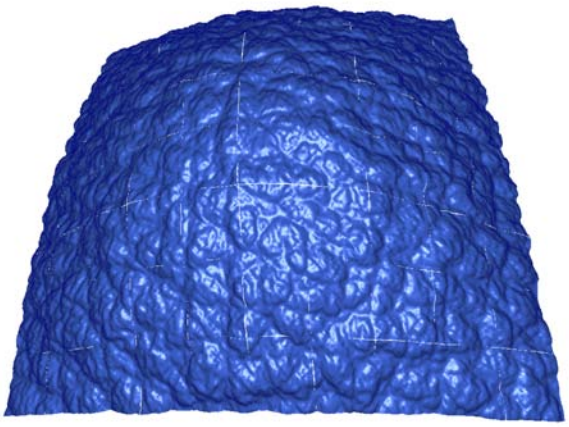


Figure B.22.: *Gaps in mesh if displaced along wrong normals.*

Figure B.23.: *The Hand without extraordinary vertices support.*

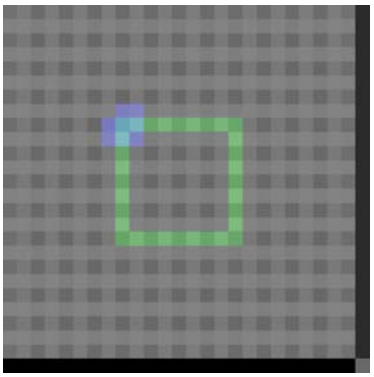


Figure B.24.: *Look up Table.*

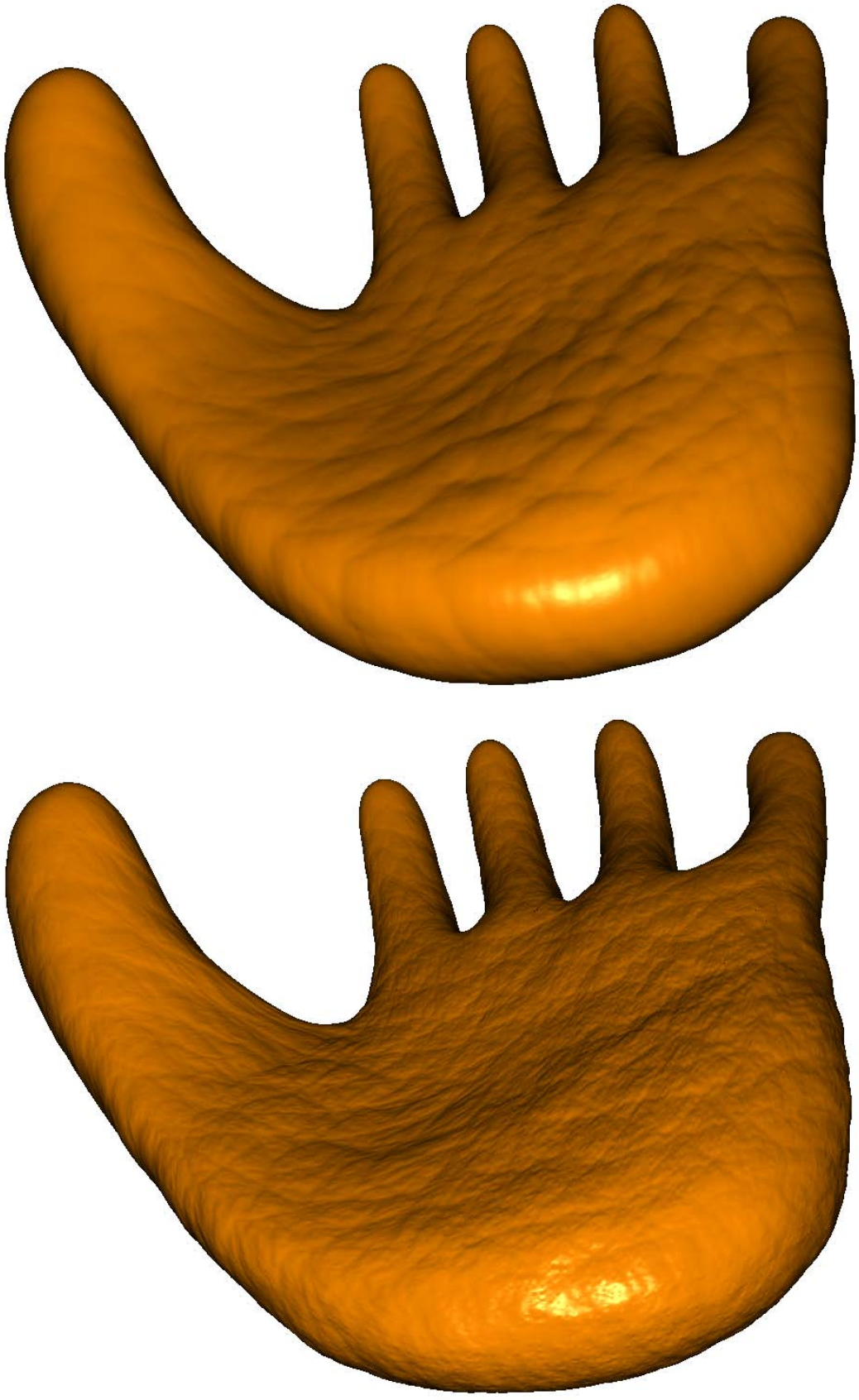


Figure B.25.: *Top: classic map, bottom: MR displacement map.*