Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics
Department of Geometry

Subdivision Surfaces

Master's Thesis

Bratislava
May 2002

Author: Peter Leškovský
Supervisor: RNDr. Pavel Chalmovianský

I honestly declare that I have written the submitted master's thesis by myself and I have used only the literature mentioned in the bibliography.

Peter Leškovský

Acknowledgments

I wish to thank to

- my supervisor Pavel Chalmovianský for his assistance in the preparation of my master's thesis
- my friend Zuzana Ružičková for drawing me some nice pictures

Abstract

There is much research to be done concerning the representation of surfaces derived from points that must be interpolated or approximated. Mostly known tensor-product B-splines work well for modelling surfaces based on rectilinear control nets but are not sufficient for more general surfaces. However the NURBS surfaces are high standard on any computer graphics systems, but these generating surfaces are based on exhaustive mathematical backgrounds.

The latest research, although first research were made in 1978 by E. Catmul and J. Clark [CC78] and D. Doo and M. Sabin [DS78], introduce new surface generating method that abstracts from the mathematical backgrounds and suggests efficient and easy to implement way of generating surfaces.

''Subdivision surfaces''

This work is concerning the basics behind the subdivision technique, introduces methods for producing highly smooth surfaces from polyhedra with boundaries of arbitrary topology and finally softly presents up-to-now-known schemes and some features in the meaning of "adaptive refinement". However the most emphasis is addressed to the effective implementation of many known schemes with the use of Microsoft's COM technology.

In the first sections you will find the basics about subdivision surfaces including the main geometric criteria that are required of the schemes. Thereafter we will explain the technology and the principles of implementing most known subdivision schemes. At the end we will present some results and consequential conclusions. All used subdivision techniques are described in detail in the appendix section.

As an addition to this master's thesis is the implementation of some schemes itself, together with the source code, the documentation and some gallery pictures, which should be found on attached CD[1].

[1]If you did not get the CD, try to check my web page http://kopernik.cc.fmph.uniba.sk/~leskovsky or write an e-mail to 7leskovsky@st.fmph.uniba.sk

Contents

1 Introduction **3**
- 1.1 Nowadays generating of surfaces . . . 3
- 1.2 Advantages of Subdivision Surfaces . . . 7
- 1.3 Disadvantages of Subdivision Surfaces . . . 7

2 Basics **9**
- 2.1 Fundamental principles . . . 9
- 2.2 Getting into Backgrounds . . . 12
  - 2.2.1 Cell Complexes . . . 12
  - 2.2.2 Topological Refinements . . . 14
  - 2.2.3 Functions on Cell Complexes . . . 16
  - 2.2.4 Subdivision scheme . . . 17
  - 2.2.5 Subdivision Matrix . . . 17
- 2.3 Analysis of Subdivision Surfaces . . . 19
- 2.4 Known Subdivision Schemes . . . 20
  - 2.4.1 The Subdivision Classification . . . 20

3 Implementation **22**
- 3.1 Input, output constraints . . . 22
- 3.2 The design . . . 23
  - 3.2.1 Client-Server layout . . . 23
  - 3.2.2 Interface . . . 24
  - 3.2.3 Our configuration . . . 24
  - 3.2.4 Used Technology . . . 26
- 3.3 Data Structures . . . 27
- 3.4 The implementation Methods . . . 28
  - 3.4.1 Vertices creation . . . 29
  - 3.4.2 Faces creation . . . 29
  - 3.4.3 Data association . . . 31
- 3.5 Notes on improvement . . . 31

4 Results **33**
- 4.1 Nice properties observed by subdivision schemes . . . 33

5 Conclusions **37**
5.1 The trends in constructing subdivision solids 37
5.2 Commercial software 37
5.3 Summary and future work 38

A Detailed subdivision schemes **39**
A.1 Loop 39
A.2 Catmull-Clark 40
A.3 Doo-Sabin 41
A.4 Butterfly 42
A.5 $\sqrt{3}$ 42

Chapter 1

Introduction

1.1 Nowadays generating of surfaces

At first we will take a look at some nowadays techniques of generating solid surfaces, techniques mostly used in CAGD[1], so we can better understand the features of subdivision surfaces.

1. **Parametric surfaces**:
   Defined as
   $$Q : \Omega \in \mathbb{R}^2 \to \mathcal{S} \in \mathbb{R}^3$$
   $$Q(u,v) = [\, x(u,v)\, ,\ y(u,v)\, ,\ z(u,v)\,]\, , \quad u,v \in [\, 0,1\,]\, ,$$
   where x, y, z are real functions defined over plane $\Omega : [0,1] \times [0,1] \subseteq \mathbb{R}^2$.

   Although we can easily get the derivatives of these surfaces, compute intersections and we easily visualize these surfaces, the way of visualization is expensive and moreover there is bad local control of the shape of such generated surfaces.

2. **Implicit surfaces**:
   The set of surface points S is defined
   $$S = \{\, [\, x,y,z\,] \in \mathbb{R}^3 \mid f(\, x,y,z\,) = 0\, \}\, , \quad f : \mathbb{R}^3 \to \mathbb{R} \quad (\, continuous\,)$$
   (e.g. $S_{sphere} : f(\, x,y,z\,) = x^2 + y^2 + z^2 - r^2$ over the domain $\mathbb{R}^3$
   for some parameter r – the radius)

   So far simple objects, building up a "skeleton", are used to define the surface (this could be points, lines, circles, ...), and the surface is then created by "puffing up" the skeleton. By the inflating, several blending functions are used to ensure required fairness what leads to difficult implementations of sharp edges.

   You can imagine, that the shape control and evaluation is difficult since slightly changing function f can have drastic effects on the shape. In addition, such surfaces provide bad explicit parameterization for texturing.

[1]Computer Aided Graphics Design

A good example of this kind are the "MetaBalls" surfaces, where a group of hyper-ellipsoids is used in place of the continuous function f. At a certain distance these ellipsoids "melt" together to build a connected shape.

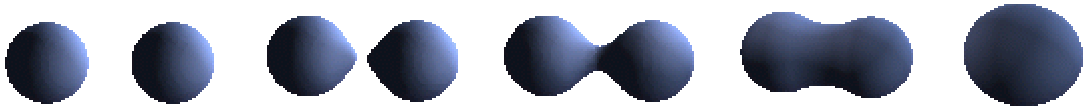


Figure 1.1: MetaBalls objects melting together.

MetaBalls are suitable for organic shaped objects, but become poor in geometric modelling.

New approach to implicit modelling suggests also so called "F-rep", where geometric objects are defined by a class of restricted functions which we are able to render fast (e.g. *min*, *max* functions, polynomials). The geometric object G is defined more generally:

$$G = \{ x \in \mathbb{R}^n \mid f(x) \geq 0 \}, \quad f : \mathbb{R}^n \to \mathbb{R} \quad (\,\mathit{continuous}\,)\,.$$

Although just three coordinates define the object in the modelling space, the higher coordinates can be used for a simple animation.

The biggest advantage of this spacial function class comes with their effective adaptation to set operations, space deformations, blending and morphing, since all these geometric modifications can be done just by using the basic calculus over the restricted function class (addition, multiplication, composition, ...). (For an overview see [Has02], or look for "Hyper-Fun" on the internet.)

3. **Surfaces defined over a grid of control points**:

   - Bézier patches

     For a given net of control points $P_{i,j}$ we define the Bézier patch as

     $$Q(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \; P_{i,j}\; B_i^n(u)\, B_j^m(v)\,, \quad u,v \in [\,0,1\,]\,,$$

     where B_i^k are the Bézier basis functions:

     $$B_i^k(t) = \binom{k}{i}\, t^i\, (1-t)^{k-i}\,, \quad t \in \mathbb{R}\,.$$

     These surfaces are easy to differentiate, easy to model and also the intersections can be computed easily. Nonetheless the variety of generated surfaces is not sufficient – the modelling of conic sections and other ellipsoid solids is unavailable.

Another problems arise when trying to join more patches into one surface with prescribed smoothness – strong constraints are required on the control points. The local control of generated surface is also not sufficient – by any repositioning of some control point the whole patch changes. As the last problem we mention the non-invariance with respect to perspective projection.

- B-Spline surfaces

$$Q(u,v) = \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} P_{i_1,i_2}\, N_{i_1}^{d_1}(u)\, N_{i_2}^{d_2}(v)\,, \quad u,v \in [\,0,1\,]\,,$$

where P_{i_1,i_2} are the control points, $N_{i_j}^{d_j}$ are the B-Spline basis functions ($i_j \in \{0,1,\ldots,n_j\}$) defined over knot sequence $t_0, t_1, \ldots, t_{m_j}$ where $m_j = n_j + d_j + 1$:

$$N_i^0(t) = \begin{cases} 1 & t_j \le t < t_{i+1} \\ 0 & otherwise \end{cases}$$

$$N_i^k(t) = \frac{(t - t_i)}{(t_{i+d} - t_i)}\; N_i^{k-1}(t) + \frac{(t_{i+d+1} - t)}{(t_{i+d+1} - t_{i+1})}\; N_{i+1}^{k-1}(t)\,.$$

$$N_i^k(t) = \frac{(t - t_i)}{(t_{i+d} - t_i)}\; N_i^{k-1}(t)\,, \quad if \quad t_{i+1} = \cdots = t_{i+k+1}\,.$$

$$N_i^k(t) = \frac{(t_{i+d+1} - t)}{(t_{i+d+1} - t_{i+1})}\; N_{i+1}^{k-1}(t)\,, \quad if \quad t_i = \cdots = t_{i+k}\,.$$

$$N_i^k(t) = 0\,, \quad if \quad t_i = \cdots = t_{i+k+1}\,.$$

$$k = 1, 2, \ldots, d\,.$$

The control net enables local control of the surface while saving the smoothness of the surface. This surfaces are invariant with respect to affine transformations.

- NURBS[2] surfaces
  Defined over B-Spline basis functions as

$$Q(u,v) = \frac{\sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} w_{i_1,i_2}\, P_{i_1,i_2}\, N_{i_1}^{d_1}(u)\, N_{i_2}^{d_2}(v)}{\sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} w_{i_1,i_2}\, N_{i_1}^{d_1}(u)\, N_{i_2}^{d_2}(v)}\,, \quad u,v \in [\,0,1\,]\,,$$

[2]Non Uniform Rational B-Splines

where P_{i_1,i_2} are the control points and w_{i_1,i_2} are the control weights for each control point. When in the denominator the sum equals to null, $Q(u,v) := 0$ by definition.

The rational coefficients enable to construct free shaped curves, even conic sections efficiently. NURBS surfaces are invariant with respect to affine transformations and perspective projection[3].

The spline representation is particularly useful due to their intuitive shape control mechanism based on control vertices which locally attract the surface. Their main disadvantage is the complicated mathematical machinery, when connecting non-regular surfaces together, while observing desired grade of smoothness at "extraordinary" vertices[4]. In fact, spline based surfaces are mostly used for generating objects which are topologically equivalent with plane, torus or disk (with or without boundary).

The research of possible extensions moves in different directions: finding a way to generalize B-Splines functions or looking for a way to join patches in non-regular settings[5].

4. **Procedural representations**:

   Giving a mesh[6] and a simple rule of generating new surface points we produce finer approximation of required object. Such algorithms are used in fractal generation of surfaces or in generating of subdivision surfaces. The efficiency of subdivision algorithms, their easy implementation, flexibility and simplicity make them suitable for many interactive computer graphics applications. We will take a more detailed look at this methods of surface generating in the following pages.

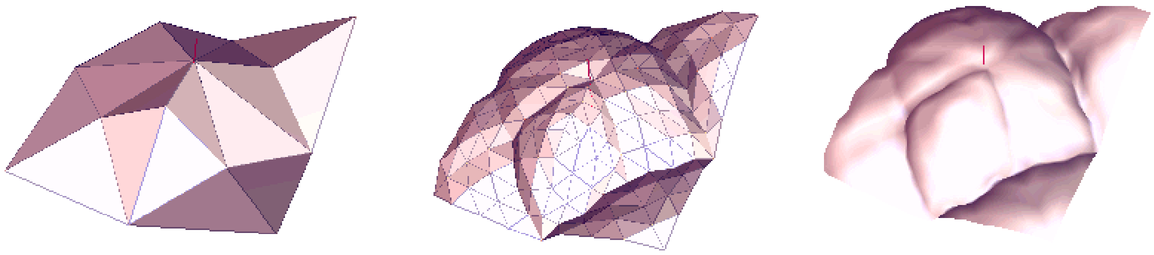


Figure 1.2: An example of fractal generated landscape. The "displaced midpoint insertion" method has been used.

[3] at first we project the control points and then we generate the curve by common algorithm

[4] vertices where more or less than 4 patches (for quadrilateral patches) or 6 patches (for triangular patches) join together

[5] See TURBS - Topologically Unrestricted Rational B-Splines [Rei95b]

[6] connected net of points in the modelling space building up a solid

1.2 Advantages of Subdivision Surfaces

The construction of surfaces through subdivision smartly addresses many issues that computer graphics adepts are faced with. Let us mention a few of them:

Arbitrary topology: It is desirable to build surfaces of arbitrary topology since in practise we cannot insist on vertices all having degree 4 (for quadrilateral meshes) or 6 (for triangular meshes)[7]. Subdivision generalizes classical spline patch approaches to arbitrary topology. They can handle objects of arbitrary topology quite well without loosing efficiency.

Scalability: For interactive applications, efficiency is of the primary importance. Because of its recursive structure, subdivision accommodates "multi-level" rendering, "adaptive refinement" and compression for the internet.

Surface features: Subdivision can control the shape and size of features, such as creases, grooves, sharp edges or even the fairness of a geometric object simply by manipulating the subdivision coefficients.

Multiresolution surfaces generation: Subdivision in its pure form is useful for generating smooth surfaces. However, applications such as special effects and animation require creation and manipulation of complex geometric models, which, like real world geometry, carry details at many scales. Multiresolution mesh editing proposes "level-of-detail" feature by including detail offsets at every level of subdivision.

Numerical stability: Subdivision representations have many nice properties required by finite element solvers. As a result they are suitable for many numerical simulation tasks in engineering and computer animation.

Code simplicity: While some deeper mathematical analysis are quite involving, the final implementation is simple and the runtime performance is efficient. There are also known formulas for computing normals and limit positions of the control points.

1.3 Disadvantages of Subdivision Surfaces

The main disadvantages of subdivision surfaces arise from **bad explicit parameterization** and therefore many physical or geometric properties like length, curvature or explicit analytical formulas for tangent vectors and surface normals are hard to determine. Several formulas for determining the end position of vertices, or for computing the normal and tangent vectors have been presented for each subdivision scheme, but all these formulas handle just the control points of the mesh at any subdivision level. Thus for random surface points some approximative methods have to be used.

There are still pending issues concerning non-regular controlling mesh. To gain a **$\mathbf{C^2}$-continuous surface** at all vertices meets difficulties at the extraordinary ones. Till

[7]the Euler characteristic for planar graphs tells us that for example the sphere cannot be build from quadrilateral (triangular) control mesh having all vertices of degree 4(6)

now, there exists no suitable scheme for $\mathcal{C}^2$-continuous surfaces[8]. While there is a possibility to obtain a scheme which is C^2-continuous, it has been shown, that such schemes either have large support or necessarily have zero curvature at extraordinary vertices.

Related problem is also creation of **smooth connections** of these surfaces. To achieve better boundary control, combined subdivision schemes and subdivision schemes with normal control have been presented.

By the implementation you have to watch the **large amount of data** arising by every iteration step, thus thinking of data compression is in place. Even by the huge meshes, created after several subdivision steps, **sufficient approximation** of the final surface is missing. To get better approximations with less control points variational schemes or an adaptive refinement rules must be used.

[8]We know schemes which produce $\mathcal{C}^2$-, or even $\mathcal{C}^3$-continuous surface almost everywhere, but at the extraordinary vertices they offer just $\mathcal{C}^1$-continuity

Chapter 2

Basics

Subdivision is a method for generating smooth surfaces, which first appeared as an extension of splines to arbitrary topology control nets. This method produces every time denser net of surface points, whose position is determined by the coarser net in domain of the final surface.

Although subdivision was introduced as a generalization of knot insertion algorithms for splines, it is much more general and allows considerable freedom in the choice of subdivision rules. These degrees of freedom can be used to obtain surfaces with specific properties (varying degree of smoothness, interpolation) or with features such as creases and cusps.

2.1 Fundamental principles

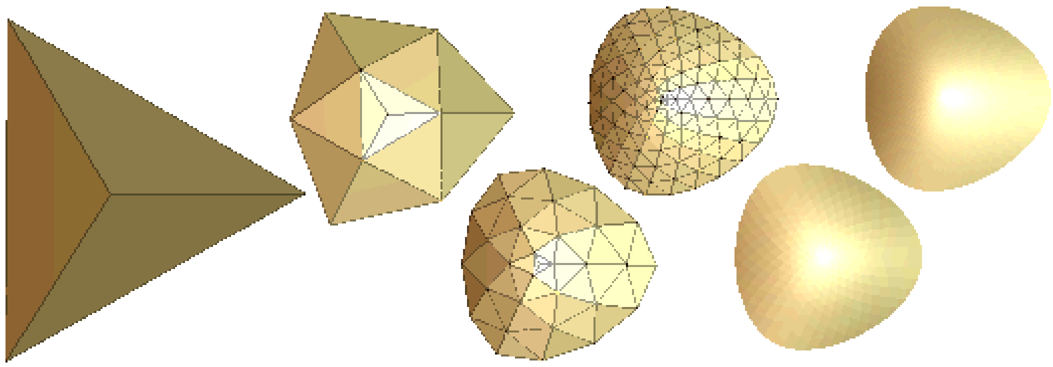


Figure 2.1: An example of Loop's subdivision scheme applied on pyramid (5 iteration steps are shown).

We can say, that subdivision method is based on iterated transformations. Let F be a function that maps one geometric shape $\mathcal{G}$ into another geometric shape. If $\mathcal{G}_0$ is an initial shape, we can define the infinite sequence of shapes by:

$$\mathcal{G}_{i+1} = F(\mathcal{G}_i)\,.$$

If F is "well-behaved" function, then there exists a limit shape $\mathcal{G}$ that is a fixed point of F,

$$\mathcal{G} = F(\mathcal{G}) \, .$$

Given an initial mesh, subdivision defines a smooth curve or surface as the limit of successively refined polyline or polyhedral meshes.

The refined meshes are obtained by adding new vertices to the mesh and connecting them according to the connection of the old vertices. The positions of new vertices are computed as functions of positions of the old vertices; the positions of old vertices in the refined mesh can be modified, or they may be even dropped.

One could imagine many ways to place the new points, but clearly, the shape and smoothness of the resulting surface depend on this choice. For the scheme to be practical, these rules should be as simple as possible, yet capable of generating a smooth limit surface. Mostly linear functions are used as the refinement rule for subdivision schemes.

Consider the following example:

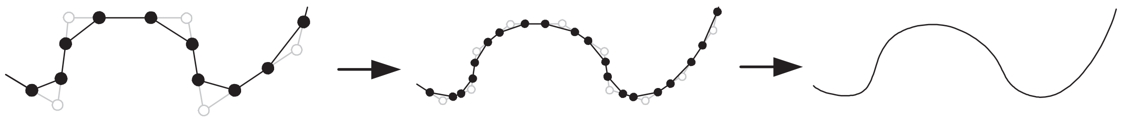


Figure 2.2: Chaikin's subdivision scheme for curves. The old vertices are shown as light gray hollow circles, the new are black and filled.

One of the simplest subdivision schemes is the Chaikin's [Cha74] corner cutting algorithm which prescribes the refinement of the initial polyline. The position of new points at level $i+1$ is computed as a linear combination of old points at level i as follows (see Figure 2.2):

$$p_{2k}^{i+1} = \frac{3}{4} p_k^i + \frac{1}{4} p_{k+1}^i \qquad p_{2k+1}^{i+1} = \frac{1}{4} p_k^i + \frac{3}{4} p_{k+1}^i \tag{2.1}$$

It is known that this scheme converges to a quadratic B-spline curve which is C^1 smooth.

Let us have a better look at this example – at the subdivision property, proposed by uniform B-splines[1].
Take the B-spline basis functions of second order defined over uniformly spaced knot sequence $[\ldots\ 0, 1, 2, 3, \ \ldots]$ (denoted by $[\ldots\ t_0, t_1,\ t_2, t_3, \ \ldots])$

$$\mathbf{N}_i^2(t) = \begin{cases} \frac{1}{2}(t-i)^2 & t_i \le t < t_{i+1} \\ -\frac{3}{2} + 3(t-i) - (t-i)^2 & t_{i+1} \le t < t_{i+2} \\ \frac{1}{2}(3-t-i)^2 & t_{i+2} \le t < t_{i+3} \\ 0 & otherwise \end{cases} \quad , i \in \mathbb{Z}$$

[1]For better understanding how to get these equations one is referred to [War95] pp. 11–17, or [ZSD+00] pp. 22–29

The goal is to define the same linear space over refined knot sequence $[\ldots\ 0, 1/2, 1, 3/2, 2, 5/2, \ldots]$ and get the transformation matrix from one space to the other. We take the basis functions $N_i^2(2t)$, $i \in \mathbb{Z}/2$ defined over the refined knot sequence.

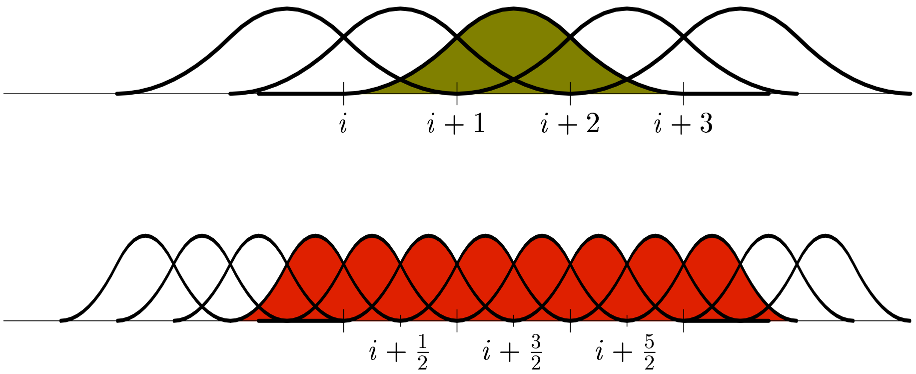


Figure 2.3: The B-spline basis functions defined over uniformly spaced knot sequence and defined over refined knot sequence.

One can prove, that the old basis function satisfy a refinement equation[2]

$$\mathbf{N}^2(t) = \frac{1}{4}\mathbf{N}^2(2t) + \frac{3}{4}\mathbf{N}^2(2t-1) + \frac{3}{4}\mathbf{N}^2(2t-2) + \frac{1}{4}\mathbf{N}^2(2t-3)\,,$$

so in terms of dilates we get

$$\mathbf{N}_i^2(t) = \frac{1}{4}\mathbf{N}_{2i}^2(2t) + \frac{3}{4}\mathbf{N}_{2i+1}^2(2t) + \frac{3}{4}\mathbf{N}_{2i+2}^2(2t) + \frac{1}{4}\mathbf{N}_{2i+3}^2(2t)\,.$$

If we denote $\mathbf{N}^2(t)$ as a vector consisting of all translates $\mathbf{N}_i^2(t)$ of the B-spline, and using a matrix $\mathcal{S}$ to encode the refinement equations, we can rewrite the last equation to

$$\mathbf{N}^2(t) = \mathbf{N}^2(2t)\,\mathcal{S}\,,$$

where a finite portion of $\mathcal{S}$ is:

$$\mathcal{S}_{-4\ldots3,-4\ldots3} \quad = \quad \frac{1}{4}\begin{pmatrix} 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 \end{pmatrix}$$

Now consider $\mathbf{p} = [\ldots, p_{-2}, p_{-1}, p_0, p_1, p_2, \ldots]^{\mathrm{T}}$, the vector of control points of a given curve. The B-spline curve itself can be denoted as

$$spline(t) = \mathbf{N}^2(t)\,\mathbf{p} = \mathbf{N}^2(2t)\,\mathcal{S}\mathbf{p}\,.$$

[2]the universal refinement equation for B-splines of degree n is given by $\mathbf{N}^n(t) = \frac{1}{2^n}\sum_{k=0}^{n+1}\binom{n+1}{k}\mathbf{N}^n(2t-k)$, which can be found in [ZSD+00][War95]

By repeating this process

$$\begin{aligned} spline(t) &= \mathbf{N}^2(t)\,\mathbf{p}^0 \\ &= \mathbf{N}^2(2t)\,\mathbf{p}^1 \quad = \mathbf{N}^2(2t)\,\mathcal{S}\mathbf{p}^0 \\ &\;\;\vdots \\ &= \mathbf{N}^2(2^j t)\,\mathbf{p}^j \quad = \mathbf{N}^2(2^j t)\,\mathcal{S}^j\mathbf{p}^0\,, \end{aligned}$$

the relationship between the control points at different levels arise

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

Now we can see, that the rows of $\mathcal{S}$ specify the positions of new control points p_i^{j+1} as a linear function of the old control points p_i^j. Using the former matrix $\mathcal{S}$ the points are placed as specified in Chaikin's algorithm (see Equation 2.1).

2.2 Getting into Backgrounds

In the following section we will describe the basic mathematical notions and also give an intuitive view of the subdivision process. More detailed look can be found in [Zor98] [Rei99] and [Cha01].

At the beginning we will give an introduction to cell complexes and at the end we just briefly mention the general properties of subdivision surfaces and more difficult criteria and techniques used to their investigation.

2.2.1 Cell Complexes

The cell complex theory is based upon special ordered *n-tuples* to enable an enumeration of this structure.

Let $\mathcal{A}$ be a set. By $\mathcal{A}^n$ we denote the set of all possible selections of n elements from the elements of the set $\mathcal{A}$ without repetition. The elements of $\mathcal{A}^n$ will be denoted as $(a_0, \ldots, a_{n-1})$ and called **n-*tuple variations***. Let $\mathcal{A}^{[n]}$ be the set of all *cyclically* and *reverse unordered n-tuple variations*. The elements of $\mathcal{A}^{[n]}$ will be written $[a_0, \ldots, a_{n-1}]$ where $a_i \in \mathcal{A}, i = 1 \ldots n-1$. The set $\mathcal{A}^{[n]}$ is in fact the set $\mathcal{A}^n$ factored by relation of equivalence $\simeq$:

$$(a_0, \ldots, a_{n-2}, a_{n-1}) \simeq (a_{n-1}, a_0, \ldots, a_{n-2})$$

$$(a_0, a_1, \ldots, a_{n-2}, a_{n-1}) \simeq (a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$$

Thus $\mathcal{A}^{[n]} = \mathcal{A}^n/_{\simeq}$. We say, that an element $[b_0, \ldots, b_{k-1}], b_i \in \mathcal{A}$ is an ***subelement*** of $[a_0, \ldots, a_{n-1}], a_i \in \mathcal{A}$, $k \leq n$, and denote $[b_0, \ldots, b_{k-1}] \subset [a_0, \ldots, a_{n-1}]$, iff there exist elements $c_0, \ldots, c_{n-k-1} \in \mathcal{A}$ such that $(a_0, \ldots, a_{n-1}) \simeq (b_0, \ldots, b_{k-1}, c_0, \ldots, c_{n-k-1})$. In this case we also say that $[b_0, \ldots, b_{k-1}]$ is ***incident*** with $[a_0, \ldots, a_{n-1}]$ and vice versa.

Definition 2.2.1. *An **abstract 2-dimensional cell complex** $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ is a triple of sets:*

$\mathcal{V} \neq \emptyset$ is a countable set of vertices – also called the 0-faces,

$\mathcal{E} \subset \mathcal{V}^{[2]}$ *is a set of edges – called 1-faces,*

$\mathcal{F} \subset \bigcup_{i=3}^{\infty} \mathcal{V}^{[i]}$ *is a set of faces – 2-faces,*

satisfying the following conditions:

1. *if $u \in \mathcal{V}$, then there exists $v \in \mathcal{V}$ such that $[u, v] \in \mathcal{E}$ (no isolated vertices),*

2. *if $[v_0, v_1, \ldots, v_{k-1}] \in \mathcal{F}$, then, $[v_0, v_1], [v_1, v_2], \ldots, [v_{n-2}, v_{n-1}], [v_{n-1}, v_0] \in \mathcal{E}$ (all edges of a face are in $\mathcal{E}$),*

3. *if $[u, v] \in \mathcal{E}$, then there is $F \in \mathcal{F}$ such that $[u, v] \subset F$ (no "dangling" edges),*

4. *if $[u, v] \in \mathcal{E}$, there are no more then two faces $F_1, F_2 \in \mathcal{F}$ such that $[u, v] \subset F_1$ and $[u, v] \subset F_2$ (no more than two faces share an edge),*

5. *if $v \in \mathcal{V}$ and set $link(v) = \{u \in \mathcal{V} \,|\, \exists F \in \mathcal{F} \wedge v \in F \wedge u \in F \wedge u \neq v\}$ is finite with k elements $u_0, \ldots, u_{k-1}$, than there exists such permutation $(u_0, \ldots, u_{k-1})$ of $link(v)$ that $[u_i, u_{i+1}]$ is edge of some face $F \in \mathcal{F}$ with $v \in F$ for all $i = 0, \ldots, k-2$. If there exists also such face for $[u_0, u_{k-1}]$ than vertex v is called* inner*, if not, the vertex v is called* boundary*. (No articulation vertices).*

Definition 2.2.2. *Let $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ be a cell complex. The cell complex $\mathcal{C}_{dual} = (\mathcal{V}', \mathcal{E}', \mathcal{F}')$ is called **abstract dual cell complex** to complex $\mathcal{C}$ iff:*

1. *$\mathcal{V}' = \mathcal{F}$,*

2. *$[u', v'] \in \mathcal{E}'$ iff $\exists [u, v] \in \mathcal{E}, [u, v] \subset u' \wedge [u, v] \subset v'$ (recall that u' and v' are faces in $\mathcal{C}$),*

3. *for each $v \in \mathcal{V}$ there is corresponding face $F_v \in \mathcal{F}'$ such that $v' \in F_v$ iff $v \subset v'$ (all incident faces to the vertex v build up new face).*

The cell complex represents a special, by faces enhanced, not really planar (just piecewise planar) graph. The dual cell complex reminds therefore a dual graph.

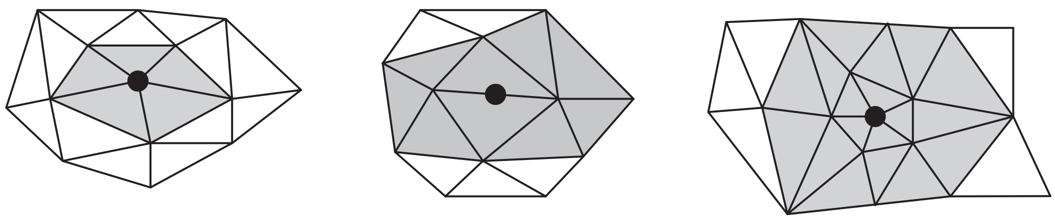


Figure 2.4: The 1-ring (left, center) and 2-ring (right) neighborhoods of a vertex. The 1-neighborhood of a vertex at an edge is meant to be the 1-neighborhood of the edge (thus of the end vertices of that edge).

Definition 2.2.3. *A* ***subcomplex*** $\mathcal{C}' = (\mathcal{V}', \mathcal{E}', \mathcal{F}')$ *of a complex* $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ *is a complex, that satisfies* $\mathcal{V}' \subseteq \mathcal{V}$, $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{F}' \subseteq \mathcal{F}$.

Definition 2.2.4. *Let* $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ *be a complex. A* ***1-neighborhood*** $N_1(V)$ *of a set of vertices* $V \subset \mathcal{V}$ *is the subcomplex of* $\mathcal{C}$ *consisting of all faces incident with at last one vertex* $v \in V$*, their edges and vertices. An* ***k-neighborhood*** $N_k(V)$ *of a set of vertices* $V \subset \mathcal{V}$ *is defined recursively as the* 1*-neighborhood of all the vertices of* $(k-1)$*-neighborhood of the set* V*. To the* ***k-neighborhood*** *it is often referred as to* ***k-ring neighborhoods****.*

There are several special kinds of cell complexes like **2-simplex** (if $\mathcal{F} \subset \mathcal{V}^{[3]}$), or **quadrilateral complex** (if $\mathcal{F} \subset \mathcal{V}^{[4]}$), and many other mostly generated by tilings of a plane (see Figure 2.5).

2.2.2 Topological Refinements

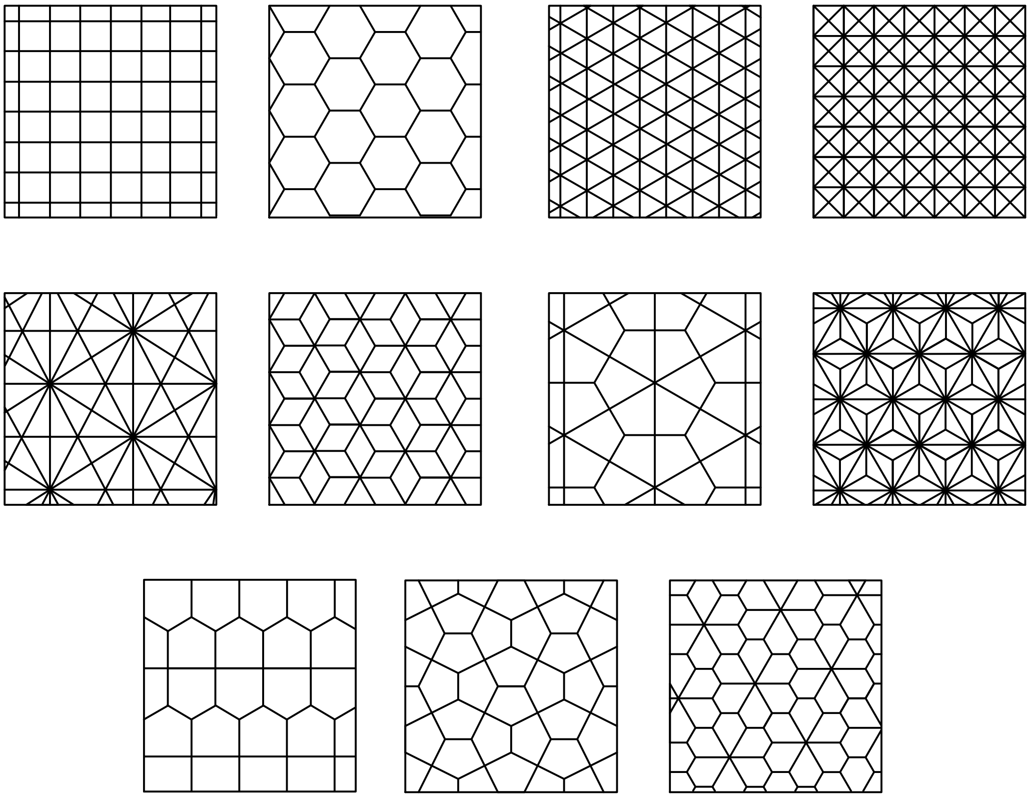


Figure 2.5: 11 known isohedral tilings of the plane.

The starting point for refinement rules are the isohedral tilings[3] (see Figure 2.5) and their dual tilings of the plane. We say a refinement rule is an algorithm to obtain a finer tiling of the same type. Isohedral tilings cannot be used to cover the surface of the whole constructed object. Because of the arbitrary topology of a common object, some irregularities appear. We define the ***extraordinary vertex*** as a vertex with other degree as

[3] A tiling is called isohedral, or Leaves, if all tiles are identical, and for any vertex the angles between successive edges meeting at the vertex are equal.

usual in specific tiling (these are e.g. the vertices of degree other than 4 in the interior for quadrilateral tilings, other than 6 for triangular, other then 3 for the boundary vertexes for the quadrilateral and other than 4 for triangular tilings). All other vertices are called ***regular***.

Every cell complex can be regarded as (or is homeomorphic to) a set of connected n-gons in sufficiently large space, therefore every cell complex can be immersed into the space. We will call the immersion a **realization** of $\mathcal{C}$ and denote it as $|\mathcal{C}|$. The topology of the image is then induced by the immersion.

Definition 2.2.5. *We say, that two cell complexes $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ and $\mathcal{C}' = (\mathcal{V}', \mathcal{E}', \mathcal{F}')$ are **homeomorphic**, if there is a homeomorphism between their realizations $|\mathcal{C}|$ and $|\mathcal{C}'|$.*

Definition 2.2.6. *Let $\mathbf{C}$ be the category of all 2-dimensional cell complexes together with continuous maps between them as morphism. **Refinement operator** $\Re$ is a functor from special subcategory $\mathbf{G}$ of $\mathbf{C}$ to $\mathbf{G}$ such that every cell complex $\mathcal{C} \in \mathbf{G}$ is homeomorphic with $\Re(\mathcal{C})$. The complex $\Re(\mathcal{C})$ is called the **refinement of the cell complex** $\mathcal{C}$.*

In the graph theory the coarse cell complex $\mathcal{C}$ as a graph G represents a minor of the graph G' as a graph of the refined complex $\mathcal{C}'$ (hence some vertices and edges of the refined graph may be omitted and some edges contracted; or in other words, new vertices, edges and faces arise in the coarser graph).

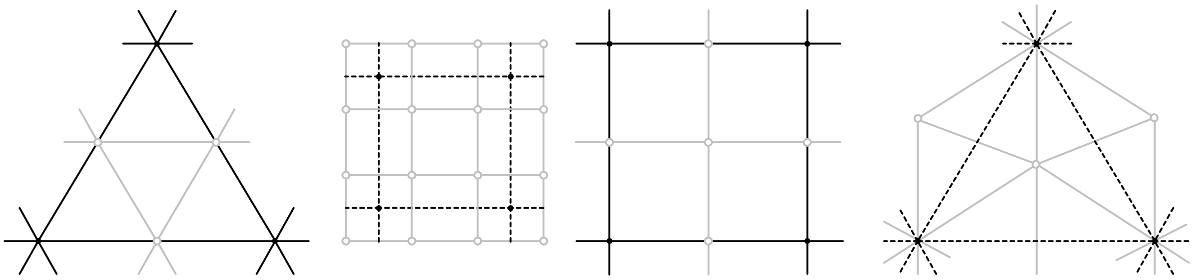


Figure 2.6: An example of some special refinements of triangular and quadrilateral tilings. New vertices are shown as hollow circles, new edges light gray, old edges black, not used edges dotted. On the left only new edge vertices are inserted and then connected together so they split the face they are connected to – we call this face-split. *On the next picture the old vertices were split into several face vertices – we got the* vertex-split *subdivision as the result. The next is a face-split scheme created using face and edge vertices. The last one, the $\sqrt{3}$ subdivision, is an example of quite a different subdivision rule – new faces are created across the boundaries of the old faces.*

The general pattern of refinement, as we construct finer tiling, can be various (how one can see from the refinement definition, and from the variety of the plane tilings). Two main approaches used to generate a refined tiling are the ***face-split*** and the ***vertex-split***. Besides this, many different refinements can be found (see Figure 2.6), but this two are the most simple and most common ones. In the face-split subdivisions each face of the cell complex is split into several finer faces, while the old vertices and edges remain. Vertex-split method instead refines the dual cell complex, therefore it seems, that every vertex is split into each incident face, to gain a finer face of the dual complex. The old vertices, edges and faces are replaced by finer faces. The schemes using the face-split method are often called ***primal***, and the vertex-split schemes are called ***dual*** since they lead to dual tilings.

In the intuitive meaning we can define the ***odd*** vertices of a refined cell complex $\mathcal{C}' = \Re(\mathcal{C})$ as $\mathcal{V}'_O := \mathcal{V}' \setminus \mathcal{V}$ (this are the vertices which were just created by the refinement operator) and the ***even*** vertices are $\mathcal{V}'_E := \mathcal{V}' \bigcap \mathcal{V}$ (this are all the old vertices).

2.2.3 Functions on Cell Complexes

By subdivision algorithms we deal with two objects. On the one hand we use the cell complexes as an abstract domain, to perform topological operations at it – refine it to obtain finer domain.

On the other hand we want to define a smooth surface over this domain, which is mostly a set of points in $\mathbb{R}^d$ – hence we work with the embedding of the cell complexes into the vector space over the field $\mathbb{R}$ (or even any other field e.g. $\mathbb{C}$), which is a map defined over the cell complex.

Definition 2.2.7. *Let $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ be an abstract cell complex, $\mathcal{B}$ a vector space over field $\mathbb{R}$.* ***Function defined over the cell complex*** *is a map $f : \mathcal{V} \to \mathcal{B}$. We will denote the* ***vector space*** *of all these functions over the field $\mathbb{R}$ as $\mathcal{P}(\mathcal{V})$.*

To get a more intuitively view, the function over the cell complex can be regarded as to a mesh, which is the embedding of the graph into the modelling space. There exist many meshes for a simple graph, therefore we think upon a vector space of all such functions.

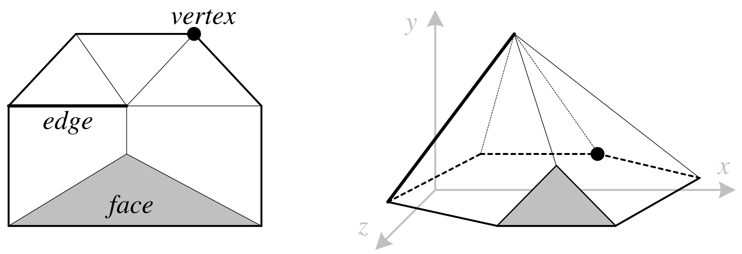


Figure 2.7: The graph $G = (V, E, F)$ consisting of vertices, edges and faces, and corresponding mesh $M = (P, G)$ consisting of points, defined over the graph G.

We are interested in all linear[4] operators $\mathcal{L}[\mathcal{C}]$ from the vector space $\mathcal{P}(\mathcal{V})$ over the

[4]The linearity is good for the simplicity and efficiency, however the subdivision theory is not limited to it. Just the stationary subdivision schemes mostly use linear relaxation rules. There are already other subdivision schemes known, based on different principles of space refinements – the variational subdivision schemes.

complex $\mathcal{C}$ into the vector space $\mathcal{P}(\mathcal{V}')$ over the the refined complex $\mathcal{C}'$, which in some manner will define the relaxation operator (the space refinements of the mesh).

2.2.4 Subdivision scheme

Definition 2.2.8. *Let* $\mathbf{G}$ *be the category of cell complexes together with continuous maps between them as morphism, let* $\Re : \mathbf{G} \to \mathbf{G}$ *be the refinement functor and* $\mathcal{L}[\mathcal{C}] : \mathcal{P}(\mathcal{V}) \to \mathcal{P}(\mathcal{V}')$ *the set of all linear operators for* $\mathcal{C} \in \mathbf{G}$. *Further denote* $\mathcal{L}[\mathbf{G}] = \bigcup_{\mathcal{C}\in\mathbf{G}} \mathcal{L}[\mathcal{C}]$, *and let* $\Im : \mathbf{G} \to \mathcal{L}[\mathbf{G}]$ *be a functor for each* $\mathcal{C} \in \mathbf{G}$ *such that* $\Im[\mathcal{C}] \in \mathcal{L}[\mathcal{C}]$.
Subdivision scheme $\mathbf{S}$ *is a couple* $(\Re, \Im)$ *with topological functor* $\Re$ *and relaxation functor* $\Im$ *defined over* $\mathbf{G}$.

Definition 2.2.9. *Let* $\mathbf{S} = (\Re, \Im)$ *be the subdivision scheme over* $\mathbf{G}$, $\mathcal{C}_0 \in \mathbf{G}$ *and* $f_0 \in \mathcal{P}(\mathcal{V})$. *The sequence* $\{(\mathcal{C}_i, f_i)\}_{i=0}^{\infty}$ *with* $\mathcal{C}_{i+1} = \Re(\mathcal{C}_i)$ *and* $f_{i+1} = \Im[\mathcal{C}_i](f_i)$ *for* $i \in \mathbb{N}$ *is called* ***subdivision process*** *of a pair* $(\mathcal{C}_0, f_0)$.

Definition 2.2.10. *Let* $\mathcal{C}_0 \in \mathbf{G}$ *and* $f_0 \in \mathcal{P}(\mathcal{V})$, *over the vector space* $\mathcal{B}$, *for a given category* $\mathbf{G}$ *of cell complexes together with continuous maps between them as morphism. A* ***subdivision scheme*** $\mathbf{S} = (\Re, \Im)$ ***is convergent***, *if there exists a continuous function* $f : \mathcal{C}^* \to \mathcal{B}$ *for the subdivision process* $\{(\mathcal{C}_i, f_i)\}_{i=0}^{\infty}$ *such that*

$$\lim_{i\to\infty} \sup_{v\in\mathcal{V}_i} \|f_i(v) - f(v^*)\| = 0\,,$$

where $v^* \in \mathcal{V}^*$ *correspond to* $v \in \mathcal{V}_i$ *in homeomorphism of* $\mathcal{C}_0$ *and* $\mathcal{C}_i$ *in the refinement, and* $\mathcal{C}^* = \lim_{i\to\infty} \mathcal{C}_i$ *(supposing the topology of* $\mathcal{C}_i$ *does not change in the limit).The function* f *is called the* ***limit function*** *of the subdivision scheme for the specified starting function* f_0 *and starting cell complex* $\mathcal{C}_0$, *or the* ***subdivision surface*** *of* $(\mathcal{C}_0, f_0)$.

2.2.5 Subdivision Matrix

As we mentioned about the linearity of the relaxation operator, it enables us now take a different view at the refinement. Let $\{(\mathcal{C}_i, f_i)\}_{i=0}^{\infty}$ be the subdivision sequence, than:

$$f_{i+1}(w_j) = \sum_{v_k\in\mathcal{V}_i} a_{jk} f_i(v_k)\,,$$

for $w_j \in \mathcal{V}_{i+1}$, where a_{jk} are some coefficients defined by the linear relaxation of the f_i.

Now define the **control points** at level i as a set $P_i := \{p^i \,|\, p^i = f_i(v), v \in \mathcal{V}_i\}$. If we denote the control points as a vector $\mathbf{p}^i = [\ldots, p_0^i, p_1^i, p_2^i, \ldots]$, the above formula can be rewritten in matrix form:

$$\mathbf{p}^{i+1} = \mathcal{S}\mathbf{p}^i\,.$$

Now $\mathcal{S}$ is the **subdivision matrix** of the subdivision scheme in specified subdivision sequence. The **subdivision matrix** relates the control points in a fixed neighborhood of a vertex at two subsequent subdivision levels. For an example we show the subdivision matrix for the Loop scheme near extraordinary control point of valence 3 (see Appendix A.1 for the rules).

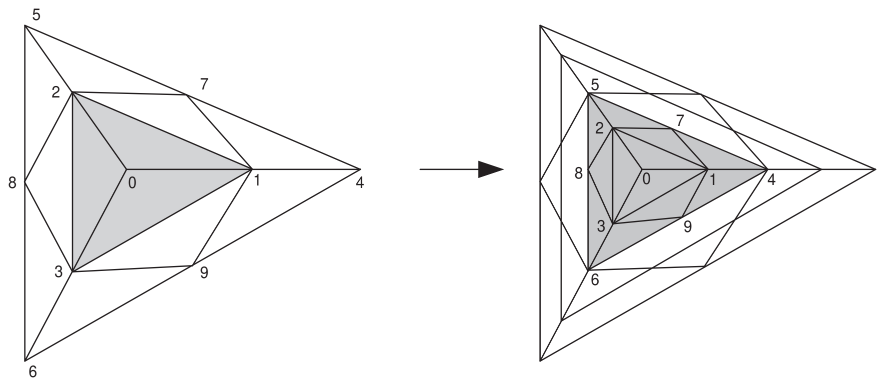


Figure 2.8: The coarser and the refined mesh of Loop subdivision scheme near a vertex of degree 3. By one refinement step, near the considered vertex, the vertices in the shaded area are created.

If we want to make the refinement around this vertex, we have to define positions of all control points generated by this refinement (see Figure 2.8 points in the shaded area). Although the Loop scheme uses just 1-ring neighborhood to define the position of any new vertex, we need to take in regard a 2-ring neighborhood of the considered vertex (remember, that if we want to examine the position of the vertex 7 in the refined mesh, we have to work with the points 0,1,2,7 of the coarser level). The relation between this control points is given by matrix on Figure 2.9.

$$\begin{pmatrix} p_0^i \\ p_1^i \\ \vdots \\ p_9^i \end{pmatrix} = \frac{1}{8} \begin{pmatrix} 8-3\beta & \beta & \beta & \beta & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha & \gamma & \alpha & \alpha & \alpha & 0 & 0 & \alpha & 0 & \alpha \\ \alpha & \alpha & \gamma & \alpha & 0 & \alpha & 0 & \alpha & \alpha & 0 \\ \alpha & \alpha & \alpha & \gamma & 0 & 0 & \alpha & 0 & \alpha & \alpha \\ 1 & 3 & 3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 3 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_0^{i-1} \\ p_1^{i-1} \\ \vdots \\ p_9^{i-1} \end{pmatrix}$$

Figure 2.9: The subdivision matrix of the Loop scheme for the extraordinary vertex of degree 3. The appropriate coefficients have to be used: $\beta = 8\frac{3}{16}$, $\alpha = 8\frac{1}{16}$, $\gamma = 8(1-6\alpha)$

The subdivision matrix holds the information of the properties of the subdivision scheme: the smoothness of the limit surface, the end positions of control points or the normal vectors, etc. For implementation purposes simple **stencil masks** were educed from the rows of the matrix, which help us to determine the position of a single control point in the next iteration level (see the Appendix section for some schemes and their stencil masks).

2.3 Analysis of Subdivision Surfaces

Several conditions which are required by the geometric constructions must be maintained when studying the subdivision process. There is no problem to investigate the surface properties in regular setting, since we know, that many schemes are just a generalization of already well known spline patches. We have to concentrate on analysis of the behavior of the schemes near extraordinary vertices, where many difficulties arise. Fortunately we only need to cover the cases, where extraordinary vertices are surrounded by regular vertices in the whole required neighborhood, since after several subdivision steps all extraordinary vertices laps with the regular ones.

By the technique of subdivision meaningful role maintains the eigen-analysis of the subdivision matrix. As we know the eigenvectors and the eigenvalues[5] reflect the main characteristics of the subdivision matrix, which in fact prescribes the generation of the subdivision surface.

We will give an overview of the basic geometric criteria that are reqired of the subdivision surfaces. For more work one should consult [Zor98], [Rei99], [ZSD+00], [Lev00].

Finite support: For a scheme to be usable in the computer graphics design, every subdivision matrix must consist only of a finite nonzero elements. The scheme is of finite support if the region over which a point influences the shape is finite, what means the counts of nonzero elements in each row of the subdivision matrix must be bounded to a constant independent on vertex.

Local definition of scheme: The rules used to determine new point position depend only on the structure of a finite part of the neighborhood of this point. Thus everywhere on the mesh, when we find the same structure (isomorphic neighborhoods), we can use the same rules.

Affine invariance: To ensure the affine invariance of the subdivision process all rows of the matrix $\mathcal{S}$ must sum to 1. In matrix notation this means $\mathcal{S}\vec{\mathbf{1}} = \vec{\mathbf{1}}$, or in other words, the **vector $\vec{1}$ is an eigenvector of the subdivision matrix $\mathcal{S}$ and $\lambda_0 = 1$ the corresponding eigenvalue.**

Take any point p^j of level j and a vector $\vec{\mathbf{v}}$ which defines the affine translation. Applying subdivision to the point we get:

$$\mathcal{S}(p^j + \vec{\mathbf{v}}) = \mathcal{S}p^j + \mathcal{S}(\vec{\mathbf{v}}) = p^{j+1} + \mathcal{S}(\vec{\mathbf{v}})$$

which implies that for the affine invariance we need $\mathcal{S}(\vec{\mathbf{v}}) = \vec{\mathbf{v}}$.

Convergence of subdivision scheme: Assume, that all eigenvectors $\vec{\mathbf{x}}_0, \vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_{n-1}$ of the local subdivision matrix $\mathcal{S}$ (of size $n \times n$) form a base with corresponding eigenvalues $|\lambda_0| \geq |\lambda_1| \geq \cdots \geq |\lambda_{n-1}|$.

The first eigenvalue $\lambda_0 = 1$ and $|\lambda_0| > |\lambda_i|$, $i \in \{1, \ldots n-1\}$ must hold.

[5] recall from linear algebra that an eigenvector x of the matrix M is a non-zero vector such that $Mx = \lambda x$, where λ is a scalar - we say λ is the eigenvalue corresponding to the right eigenvector x

By our setting of eigenvectors we can decompose any point p as a linear combination of eigenvectors:

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

where the coefficients a_i are given by the inner product $a_i = \vec{\mathbf{x}}_i \cdot p$. We can compute the result of subdivision (by linearity of $\mathcal{S}$)

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

Applying subdivision rules $\mathcal{S}$ j-times, we obtain

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

Hence in order for the sequence $\mathcal{S}^j p^0$ to converge at all, it is necessary that all eigenvalues are at most 1 (it is also possible to show, that only a single eigenvalue may have magnitude 1).

2.4 Known Subdivision Schemes

In this section we give some basic classification of present known schemes and describe the rules of some notorious stationary subdivision schemes.

2.4.1 The Subdivision Classification

As all other surfaces also the subdivision surfaces can be organized to several groups. The common criteria to classify most of the schemes are:

- the scheme is interpolating or approximating
- the type of used tiling (mostly used triangular and quadrilateral meshes)
- type of refinement rule used (face-split or vertex-split)
- smoothness of the limit surface for regular and extraordinary vertices

There are even wholly different subdivision categories:

Stationary subdivisions: These schemes compute the new points by applying fixed banded convolution operator to the original mesh. The split operations is chosen uniform such that new vertices are regular and the number of extraordinary vertices remain constant. The weights in refinement rule depend only on the valences of the vertices and always the same rule is applied in each step of the iterative refinement. These schemes can generate surfaces of various continuity[6].

[6]Although just $\mathcal{C}^1$-continuity was achieved at the extraordinary vertices yet. See the table on the next page for an overview.

Combined subdivisions: New approach to subdivision generating of surfaces introduce the combination with arbitrary generated curves. In general the curves represent the boundary or interpolation conditions. By the refinement the regard for boundary conditions is taken to compute the new vertices. The weighted combinations of the vertices of refined mesh are mixed with the exact point location of the bounded curve to obtain requested conditions. The enumeration of the exact point location at the boundary curve respond to the curves chosen form (e.g. NURBS, B-Spline or even some implicit function). More about this can be found by A. Levin in [Lev00]. Combined subdivision schemes enable the designer to prescribe finer behavior at the border as well as at the interior.

Variational subdivisions: Because in terms of surface fairness the continuity is not sometimes sufficient, some sophisticated geometric modelling applications measure the smoothness of surfaces on physically based energy functionals. This leads to variational subdivisions where a combination of subdivision and fairing is used to obtain desired properties of surface.
The basic idea of variational subdivision is that after the first step, where more vertices are introduced, the smoothing operation follows, where the vertices are shifted in order to increase the smoothness. The variational schemes try to minimize the energy function which controls the surface quality. The major difficulties arise from the fact, that geometrically meaningful energy functionals depend on the control vertices in a highly nonlinear fashion. See [Kob96, Kob96b] for details.
This schemes are called variational since the rules for mesh refinement vary from level to level.

The following table shows the classification of mostly used known subdivision schemes.

Scheme	Approximating	Interpolating	Quadrilateral	Triangular	Face Split	Vertex Split	Smoothness
Loop	$\star$			$\star$	$\star$		C^2/C^1
Butterfly		$\star$		$\star$	$\star$		C^1
Catmull-Clark	$\star$		$\star$		$\star$		C^2/C^1
Kobbelt		$\star$	$\star$		$\star$		C^1
Doo-Sabin	$\star$		$\star$			$\star$	C^1
Midedge	$\star$		$\star$			$\star$	C^1
Bi-Quartic	$\star$		$\star$			$\star$	C^3/C^1
$\sqrt{3}$	$\star$			$\star$			C^2/C^1

Figure 2.10: A short overview of major stationary subdivision schemes: Notice, that the smoothness C^j/C^i means, that the scheme produces C^j smooth surface in the regular setting, but just C^i smoothness is achieved near extraordinary vertices.

Besides this schemes already new exist – above all the 4-8 scheme by L. Velho and J. Gomes, Honeycomb by E. Akleman and V. Srinivasan, some de Rahm generalizations of Doo-Sabin schemes, many combined schemes by A. Levin, variational subdivisions by L. Kobbelt, subdivisions with normal control by D. Zorin, A. Levin and H. Biermann, and many others occur in the last time.

Chapter 3

Implementation

As long as we got the main ideas and an overview of many difficulties, which we have to watch by implementing various subdivision schemes, we can start the discussion about best possibilities and the way we choose in the software design step.

A variety of implementations have been proposed in the last years for this kind of surface representation, mostly done by the inventors of some scheme – but these were mainly concerned on presentation and visualization of the new invented scheme and the becoming results. Therefore they were specially designed just for that one concrete scheme, everything – the data structures, the vertex insertion and face splitting has been adapted to the scheme. Also some implementation were presented for fast rendering of subdivision schemes (see [PS96]), but these were also adapted just for particular type of cell complexes.

If someone wanted to present a new scheme, a new application had to be worked out, or an old application had to be changed – mostly from the grounds. These changes got even much worse if the subdivision principles of the new scheme were not too similar to the old ones.

Our goal was to create a robust "mesh object" which will enable to see the properties, and the art of surface creation, of many known subdivision schemes. Mesh object that will have enough methods for easy creation of a new scheme. Object, whose functionality can be used from any application handily.

3.1 Input, output constraints

Our mesh object is dealing just with topologically correct solids with boundaries. In the mathematical meanings we have to ensure that for every vertex there exists a neighborhood of the vertex topologically equivalent to a disk or a half–disk. In the application we have to ensure that these rules must be maintained:

- every vertex is either an inner vertex (surrounded by faces meeting at this vertex), or is a part of maximum one boundary (two boundary edges are sharing this vertex);

- every edge is shared by at least one (if that's a boundary edge) and mostly by two faces;
- all faces sharing a vertex can be arranged in such an order, that two subsequent faces always share an edge.

The input for the mesh object consists of appropriate function call of methods for creating new vertices and new faces (see Section 3.2.2). As an output we get either a list of faces, or list of coordinates of all vertices in each face (the vertices are enumerated counter-clock-wise), or a list of all vertices.

3.2 The design

3.2.1 Client-Server layout

To enable better separation of the mesh object and all subdivision schemes we choose to create one object – **mesh server**, maintaining all main functionality like the data storage and data management. The other objects, let us call them **mesh manipulator**, should be as simple as possible. They just use the methods proposed by the server to create new vertices and faces, manipulate them or to investigate the surface properties. The main mesh manipulator objects are the *subdivision object*, which applies some subdivision scheme, and the *error function object*, which finds out the discrete curvature at the faces or vertices to choose whether to split the face in the next subdivision step or not. There are also objects like **mesh import**, which loads the initial mesh from a file and fills the data structures of mesh server, and **mesh export** to transform the mesh server data to required form for any external application.

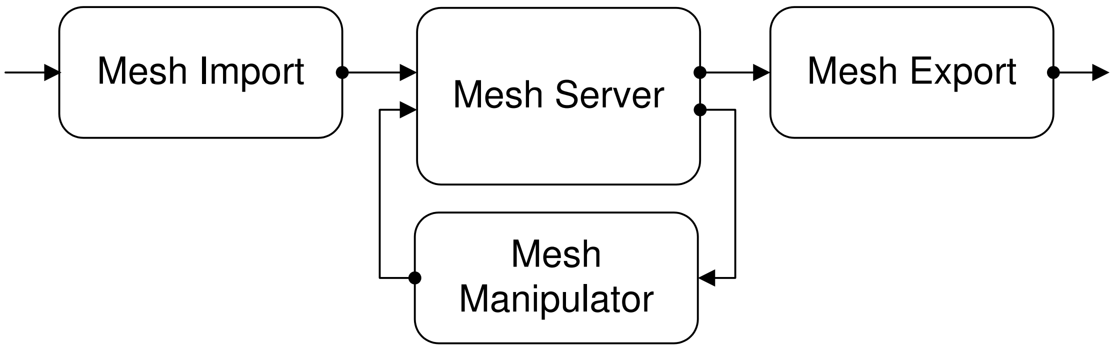


Figure 3.1: The object scheme consisting of abstract classes

Any programmer, who wants to implement new scheme or any new feature concerning the mesh, has to create just a simple mesh manipulator. The programmer is just dealing with simple objects like faces and vertices, which he does not need to bind together or even remember, the mesh server cares about memory allocation, vertex and face association and creation, and other support functionality.

The client-server layout also supposes to create many independent small objects, which can be combined together to obtain a variety of applications. The main difficulty was to create appropriate interface to offer easy, but sufficient, client implementation and best performance.

3.2.2 Interface

All objects (C++ classes) are encapsulated in COM interface objects. Every interface object provides a pair of functions for easy object binding – the *Assign Output* (or *Assign Component*, *Assign Mesh*) which attaches any other object to the output of the component, and on the other side *Get Interface* which enables to use the object's functionality through its interface. The concrete input-output interface varies from one object to the other (as seen on the Figure 3.2), therefore a specified concept of object connection must be maintained (see the following Section 3.2.3).

In the **Mesh Manipulator** object we choose a special tactics with three functions which control the output stream coming from **Mesh Server**. Two of the functions are optional, only one of them is necessary – the *Execute* function, which treats the data coming by any component. This one must be implemented in every **Mesh Manipulator** or **Mesh Export** object[1]. By the first one, *Pre-Execute*, the client can choose what kind of data we want the **Mesh Server** to send to the component, the second one, *Post-Execute*, sets whether to continue with another "execute round" or not.

For more details on implemented interfaces one is referred to the software documentation.

3.2.3 Our configuration

In our system we use one simple client application (constructed in Visual Basic) which creates all required objects and joins them correctly together.

The **Mesh Loader** reads data from a specified file type and creates the coarsest mesh by creating vertices and faces and by setting appropriate boundary flags. **Mesh Drawer** uses a special **Viewer** (it is a simple Open GL application based on automation architecture[2]) to visualize data coming from the **Mesh Server** after a simple transformation. The manipulator objects like **Subdivision Object** and **Error Function** are used to create finer approximations of the initial mesh. There is a little bit difference when using error function object before the subdivision object – a special subdivision object must be created, which can handle dead faces and many special cases on the connection between dead and alive faces. Of course we could use several smaller subdivision objects (each one for each special case) and let the error function object choose the specified one, but then the error function object would become more complicated (see Figure 3.4). We rather use the error function object just to measure the discrete curvature over the mesh with specific metrics and as a conclusion mark new dead faces.

[1] As you can observe, the Mesh Export differs from the Mesh Manipulator only a little, just like Mesh

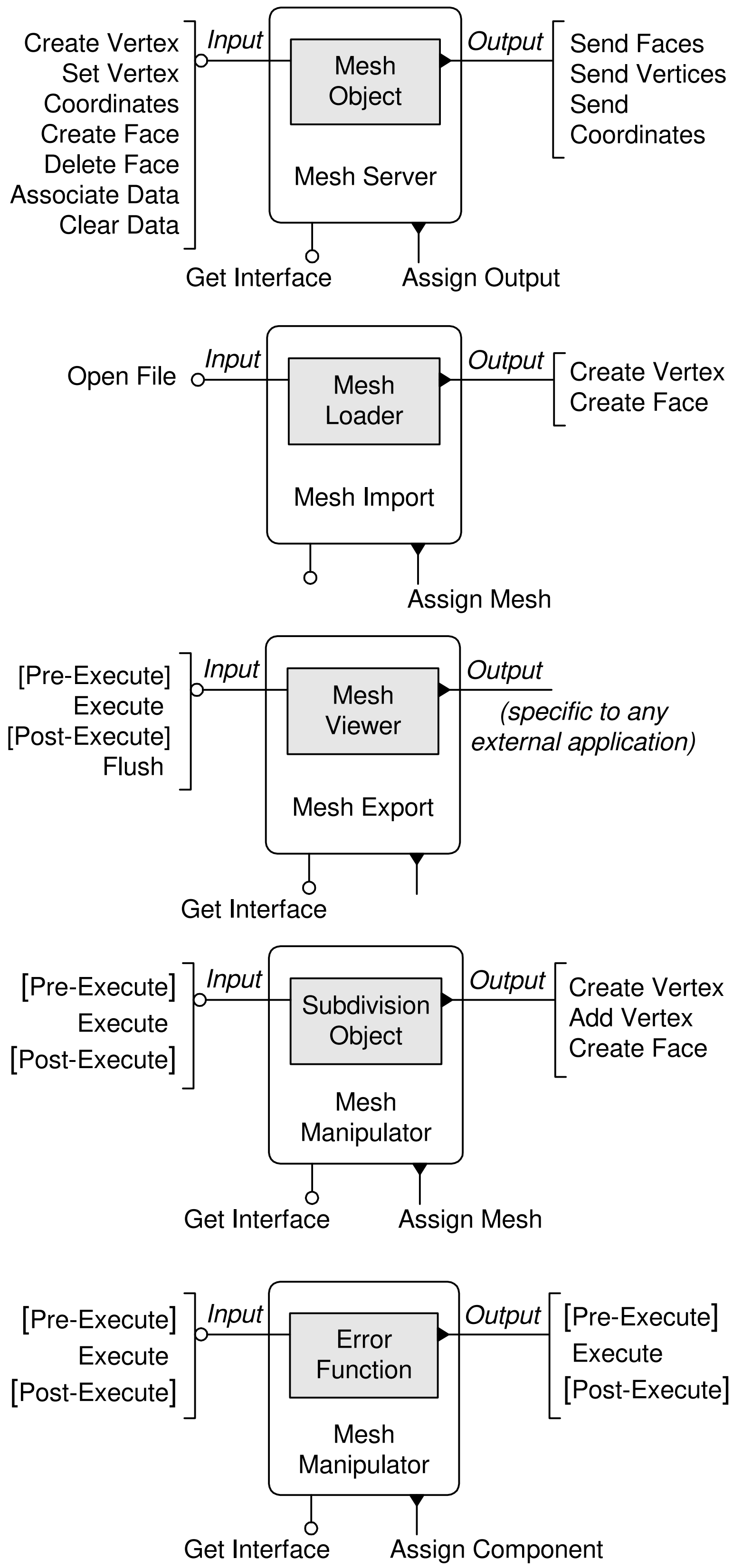


Figure 3.2: The interface description of used objects

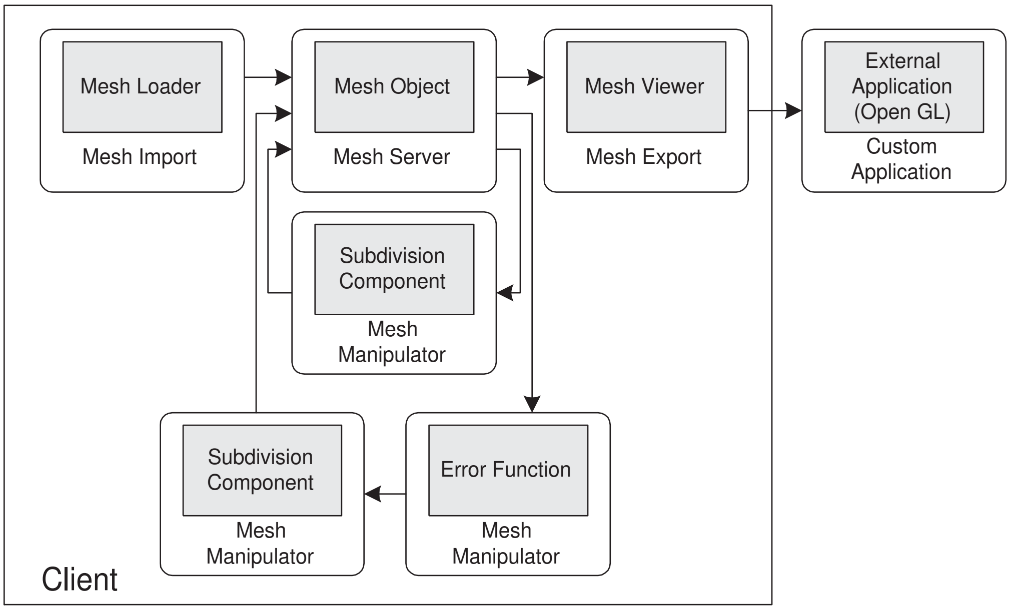


Figure 3.3: The actual configuration used in our system

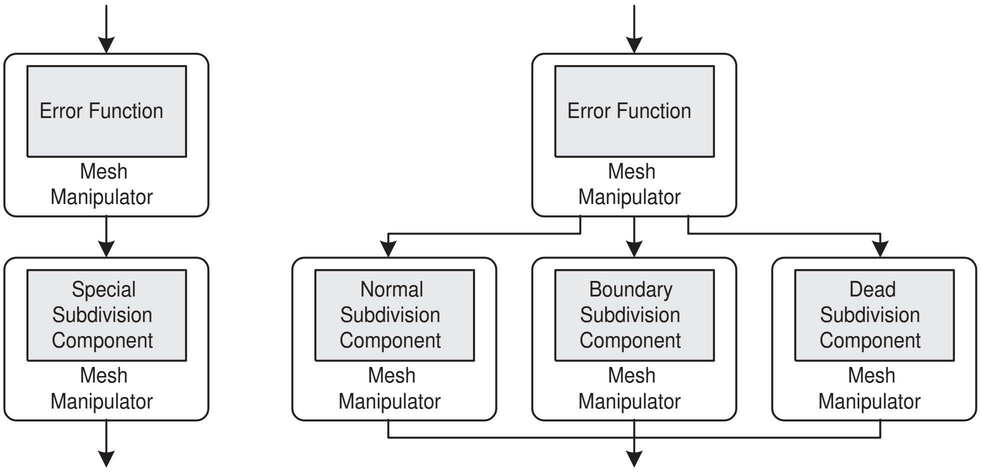


Figure 3.4: Different alternatives of error function use. Left: One special subdivision object is used. Right: Several smaller objects for each special case are used.

3.2.4 Used Technology

We choose to take advantages of Microsoft® COM+ software architecture[3] in creating client-server applications because of its mechanism that allows different software components, to connect and to communicate with each other in a well-defined manner.

Import does. Actually all these are the same objects, just some of them have different input or output.

[2]see Microsoft MSDN articles for more details

[3]COM – Component Object Model; COM defines a standard for component interoperability

Let us mention the benefits of COM programming (for a technical overview see [MSDN]):

- like today's programming standard it is based on OOP;
- the distribution of all components consist in binary code distribution, therefore no source code of other components is needed to build a new one employing the functionality of the other components;
- the creation of all objects is done dynamically, so they can be bind to other objects at runtime;
- this programming method manages the communications between components, even across process and network boundaries; the use in distributed applications and multithreading is well supported;
- COM provides memory sharing management between components;
- there is a nice support of many programming languages (C++, VB, Java, Delphi, COBOL) – actually COM is language independent; COM is also available on multiple platforms (Microsoft® Windows®, Microsoft Windows NT^{TM}, Apple® Macintosh®, UNIX®);
- at last, much of commercial software is using this technology and therefore possible embedding of our software would be much easier.

All the algorithms and objects used in this work were implemented in C++ to obtain best performance of all algorithms, to obtain all the features of object oriented programming and to get the best possibility to low level programming of COM components.

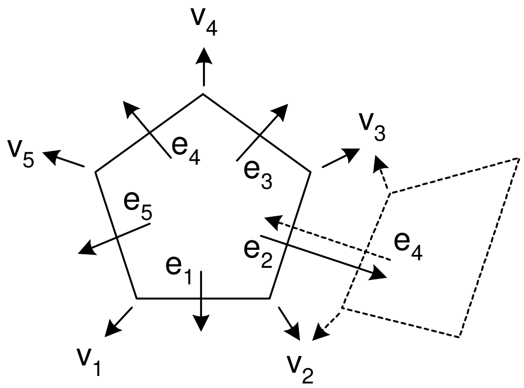
3.3 Data Structures

As suggested in [ZSD+00] we choose to use the **Double Linked Face List** structure as the main data structure in the mesh server. This structure provides better access to the faces, its vertices and the adjacent faces (in counter-clock-wise enumeration).

For each face the array of pointers to *vertices* and an array of adjacent polygons (*neighbors*) across corresponding edge number is stored. In the *edgeReenum* array we store the corresponding edge number of that edge, when seen from the neighbor across that edge.

To complete the mesh representation we add some special "inner vertices" arrays to the face structure (for description see Section 3.4.1), and data structure for vertices containing *neighboring* vertices and *adjacent* faces in CCW[4]. We do not need this two arrays at all, the mesh structure would be consistent also without them. It is just more comfortable for the end programmer (when implementing the subdivision objects), because it gives more intuitively view of the mesh object, and therefore it is useable when traversing the

[4] Counter-clock-wise enumeration




```
struct Face
{
    vector<Vertex*> vertices;
    vector<Face*>   neighbors;
    vector<short>   edgeReenum;
    . . .
}
```

Figure 3.5: C code of the DLFL structure

```
struct Face
{
    short grade;
    vector<Vertex*> vertices;
    vector<Face*>   neighbors;
    vector<short>   edgeReenum;

    vector<Vertex*> edgeVertices;
    vector<Vertex*> vertEdgeVertices;
    vector<Vertex*> vertVertices;
    Vertex*         faceVertex;

    double  *normal;
    . . .
}
```

```
struct Vertex
{
    short valence;
    vector<Vertex*> neighbors;
    vector<Face*>   adjacent;

    short dimension;
    double *coordinates;
    . . .
}
```

Figure 3.6: C code of the basic data structures

mesh or looking for the vertices in the neighborhood (when computing new coordinates of some inserted vertex). Also other variables were inserted to the structures to obtain the whole mesh functionality (see software description for more details).

3.4 The implementation Methods

To ensure good adaptation for any tiling in the subdivision process we invented several points on the polygon, which are mostly used in the topological refinement. The new faces are then crated with the use of this points. Of course we could specify low level face creation function implied by the 12 Euler topologic operators for creation of topologically correct solids (by Mantyla 1982), but this leads to uncomfortable manipulation and updating of the internal data (in this case the WEDGE data structure would be more appropriate), and it yields no benefits when programming the topological refinements of any subdivision scheme.

3.4.1 Vertices creation

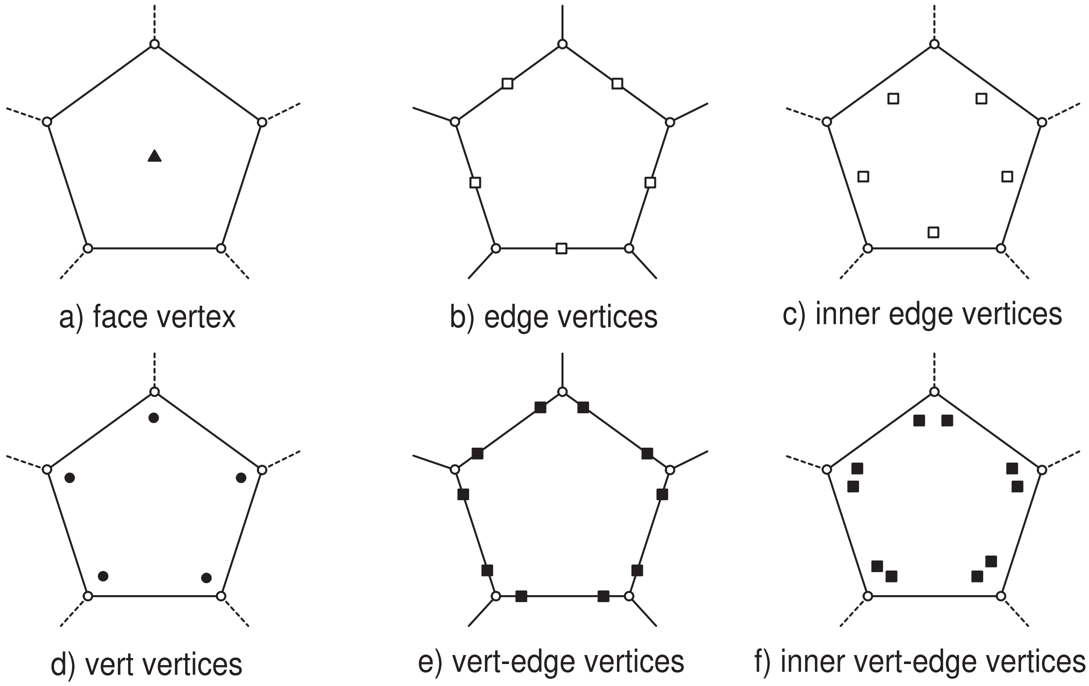


Figure 3.7: Several types of special vertices.

These are the six types of new vertices:

Face-Vertex: one vertex appertaining to the center of polygon;

Edge-Vertices: vertices belonging to each edge of the polygon; these could occur either at the edge (in that case it is also a part of the neighboring polygon), or in the inside of the polygon;

Vert-Vertices: vertices in the inside of the polygon belonging to each vertex building up the polygon;

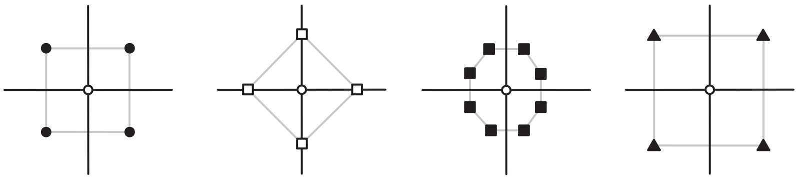
Vert-Edge-Vertices: these vertices appertain to edge and vertex concurrently; like the edge vertices also these can be either at the edge or in the inside of the polygon;

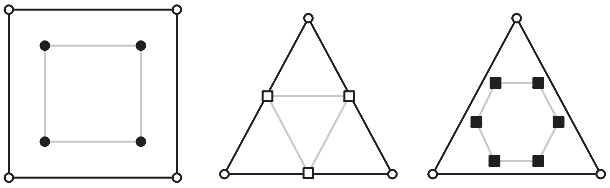
Especially the *Edge-Vertices* and the *Vert-Edge-Vertices* are mostly used in special cases, like near the boundary or near "dead faces", where we have to change the common subdivision rules.

3.4.2 Faces creation

In the subdivision process all types of just mentioned vertices and also the old vertices are used when creating new faces. Special types of faces were invented to deal with faces near boundary. These are the types of faces we distinguish between:

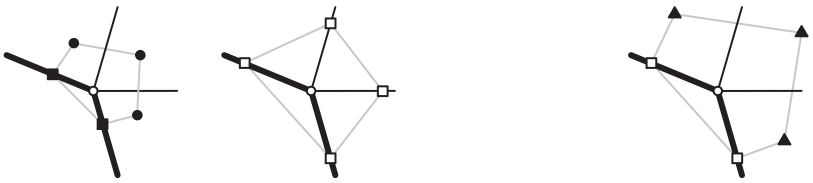
Around faces: faces crated around an old element (vertex, edge or face).

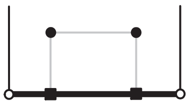
- VertRound 

- FaceRound 

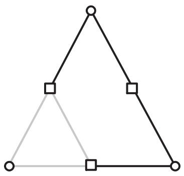
- EdgeRound 

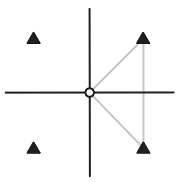
Boundary faces: when creating faces at boundary mostly not just one simple type of vertices is used – another type of vertices appear at the boundary.

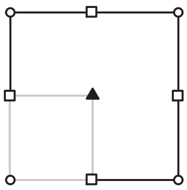
- VertRound 

- EdgeRound 

Other special faces: these are just a special sequences of vertices often used in subdivision schemes

- VertEdges 

- VertFaces 

- VertEdgeFaceEdge 

All other types of faces can be made either in two steps, or by the basic function *CreateFace* which takes a sequence of vertices (CCW ordered) as a parameter and creates a new face.

3.4.3 Data association

At last we would like to say something about associating all pointers of the data structure. When new vertices or faces are created, we do not associate the *neighbors* or *adjacent* pointer vectors of the concrete data structure. To make the mesh consistent and prepared for next subdivision process, all this work has to be done.

We separated the data association process from the face or vertex creation, because all the structures containing the information about any neighbors have to be dynamically allocated and therefore instead of creating many times small structures, we choose first to calculate the exact size of the new data[5] and than fill all the information about neighboring vertices or adjacent faces. The loading of the *neighbors* and *adjacent* arrays of the *Vertex* structure is done in one go through all faces and thereafter, also in one go through, are the *neighbors* and *edgeReenum* arrays of the *Face* filled in.

This concept enables us to skip the data association process at the last subdivision step, when we need just the rough mesh data (like which vertices are the faces consisting of and what is the position of all vertices in modelling space – the most simple data we need to visualize a surface) and when all the connection information (like adjacent faces or vertices neighbors) can by omitted. Choosing this way saves time at the last subdivision iteration, because the data association process lasts also significantly long (it is close to the tenth of the proper face and vertex creation).

3.5 Notes on improvement

The **time complexity** depends much on memory administration during the subdivision process. Old faces are destroyed, but new ones must be created (the structures are almost always of the same memory consumption). The same holds for vertices. Thus during the association process often the arrays of neighboring faces and adjacent vertices are destroyed and in a moment created again (the same data structure, of the same size, which we use for reenumeration of the neighboring faces and adjacent vertices). Much dynamically allocated data is destroyed and in a moment the same memory amount is allocated – this has a negative impact on performance.

The memory allocation could be optimized with an object, which would remember the memory storage place associated to a data structure we are just going to destroy. When allocating new memory, first the stored memory blocks will be assigned. When no memory block for the specified data structure is available, we would allocate a larger amount of memory blocks for this kind of data and associate one of them.

[5] for vertices only, we definitely know the size of the Face data structure

Memory complexity can be enhanced by using more sophisticated data structures or by avoiding the use of unnecessary data. Many different structures are known, but one should take regard for the main queries of such a data structure – above all:

- fast and easy enumeration of the 1- or 2-neighborhood of a vertex (when applying the space refinement);
- fast CCW enumeration of neighboring faces or adjacent vertices (when creating new faces around an element).

At last one could adapt the subdivision process to **multithreaded algorithm** by splitting the mesh into smaller patches which could be subdivided in separated processes. The neighborhoods of extraordinary vertices and near boundaries or any other extra-cases should be treated as individual patches and so we would get many larger regular patches, on which better data structures and faster algorithms could be used (see [PS96] for more hints).

Chapter 4

Results

4.1 Nice properties observed by subdivision schemes

In this section we will give an example of some schemes applied to a variety of meshes. We will show their nice properties, give an example of disadvantages, you would see a comparison of several schemes applied on the same mesh and at last we will show how an adaptive refinement decreases the number of faces by remaining all details.

All the pictures in this chapter were generated with the use of our software. The complicated meshes were downloaded from Caltech, Pasadena, where "Multi-Res Modelling Group" acts.

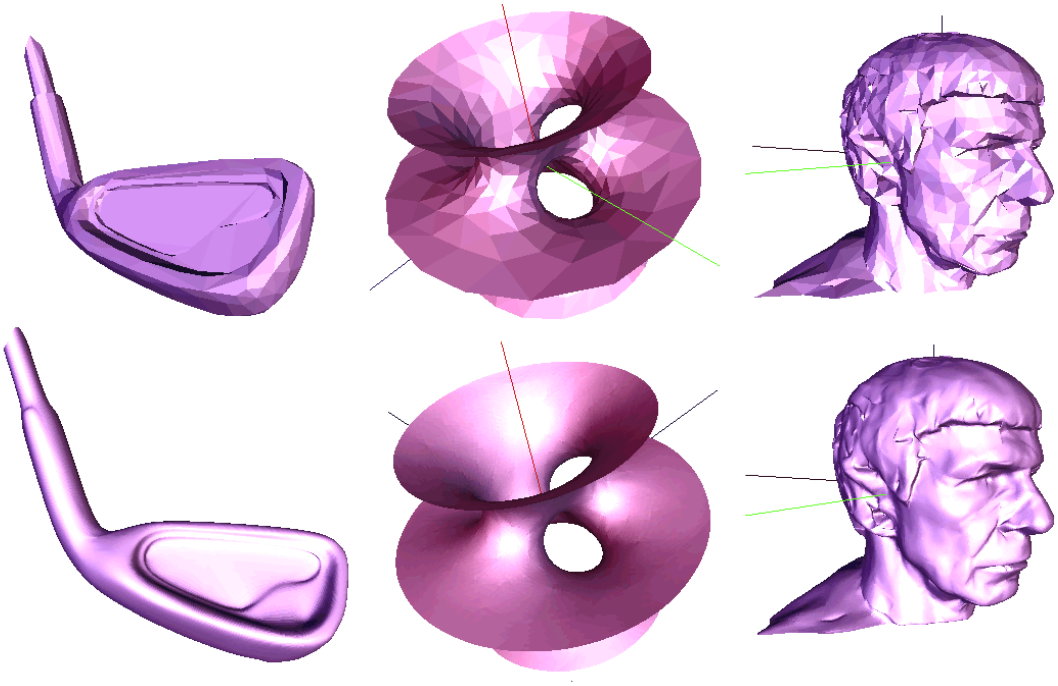


Figure 4.1: The object after 4 iterations – just a few iterations are needed to obtain a "nice" surface. The golf club modified with Loop scheme with 515 and 32960 faces respectively. The hypersheet modified with Catmull-Clark scheme with 917 and 44016 faces. And the Mr. Spock modified using Doo-Sabin scheme with 3525 and 170069 faces.

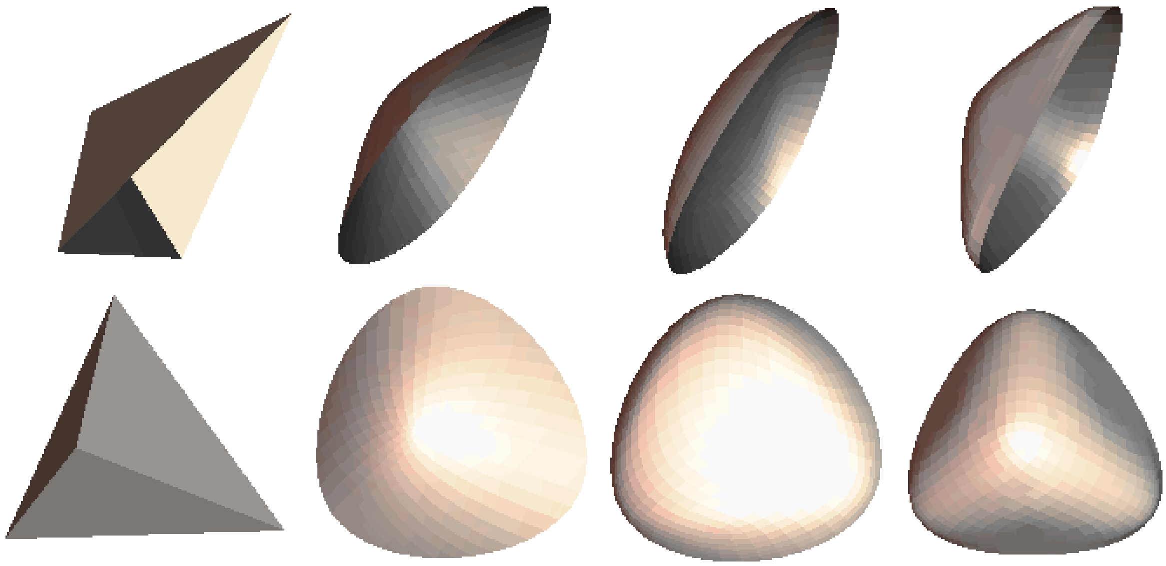


Figure 4.2: Comparison of surface generated with several schemes. From the left to the right the original coarse mesh, and the same mesh after 4 iterations with Loop, Catmull-Clark and Doo-Sabin scheme respectively is shown. One can see how the schemes behave near boundary.

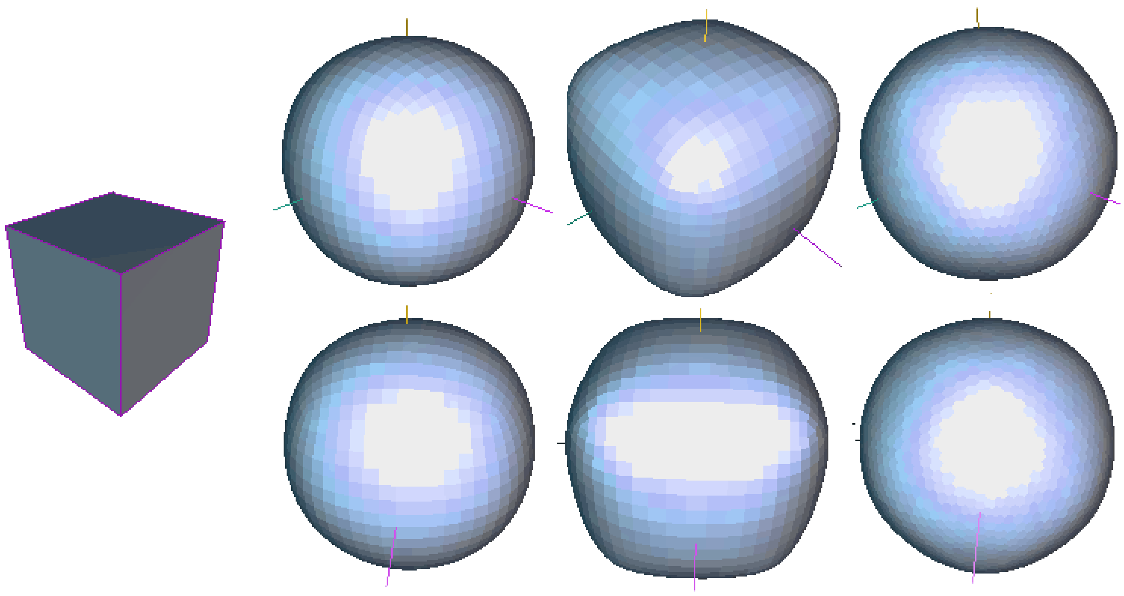


Figure 4.3: Another comparison of the Catmull-Clark, Doo-Sabin and Honeycomb (from left to right) scheme is shown – now on regular cube consisting of 6 quadrilaterals.

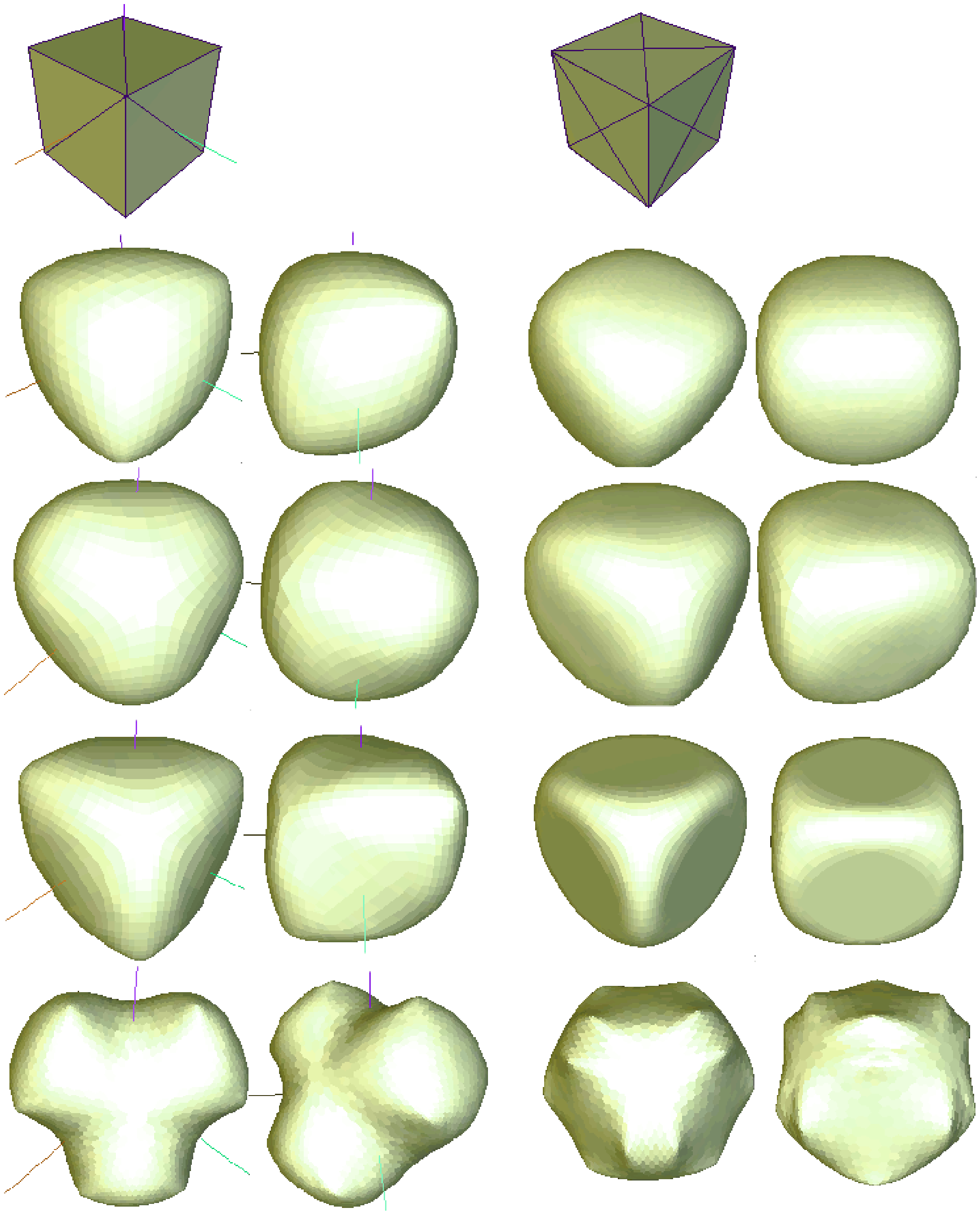


Figure 4.4: This picture shows how the coarsest mesh and especially the proper triangularization influences the final image. On the top two different triangularized cubes are shown. On the next line the same cubes are refined using Loop scheme (2 views of each cube). One can see, that on the left image, where vertices of different degree appear in the corners, that the cube does not hold its shape – on the corners with vertices of valence 3 cusps appear in contrast to the vertices of valence 6 where the cube is smooth. The picture looks more like deformed tetrahedron. On the right side the cube keeps its shape. On the next lines the same cubes are refined using Catmull-Clark, Doo-Sabin and Honeycomb scheme respectively, One can also observe how these schemes behave near extraordinary vertices.

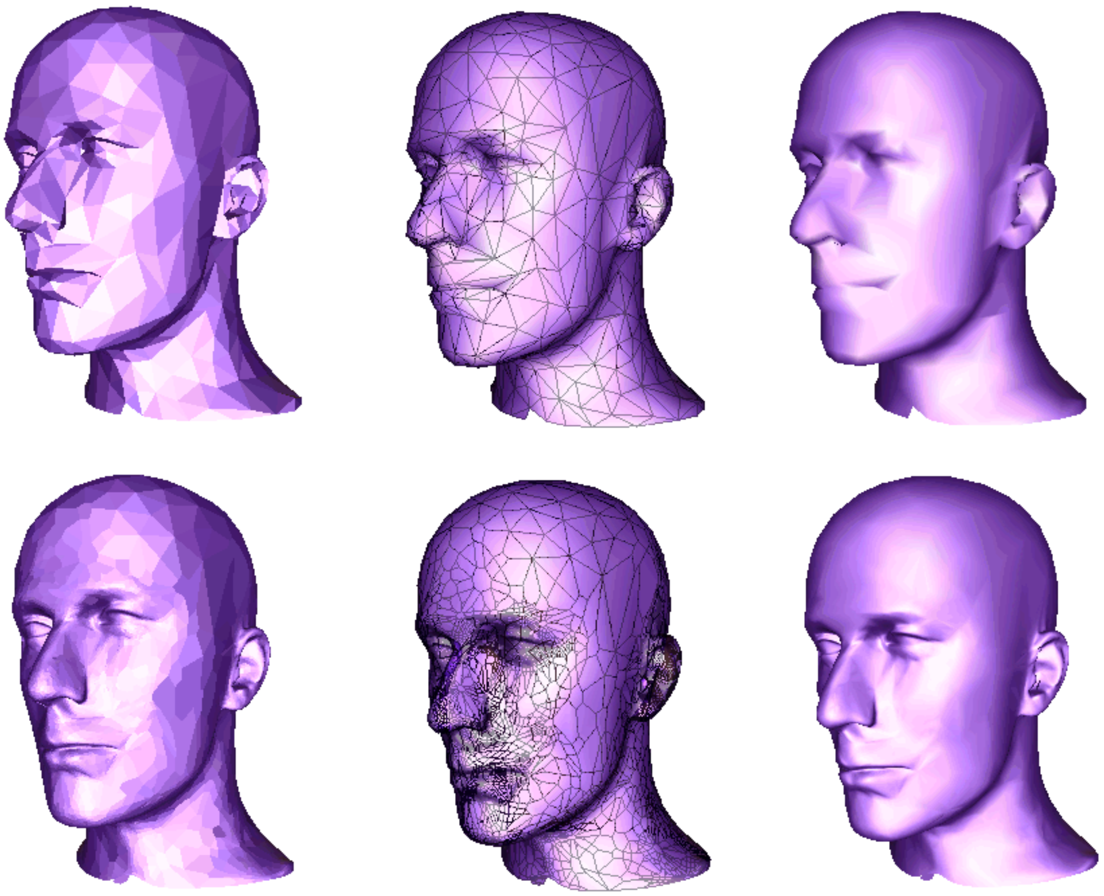


Figure 4.5: The adaptive refinement on some meshes using Doo-Sabin subdivision scheme enhanced by Error function which stops the subdivision process at vertices around whose the angles between all adjoined faces are sufficient small. On the top the coarse mesh consisting of 1355 faces, on the lower line the refined mesh with 21053 are shown flat shaded, outlined, and Gouraud shaded respectively.

Chapter 5

Conclusions

5.1 The trends in constructing subdivision solids

New topological refinements are studied, refinements on various tilings. Sometimes they produce less new faces in each step than the old schemes like Loop does, thus we can approach the limit surface more gradually. This allows finer control of the approximation. New topological refinements also lead to easier implementation of adaptive refinement.

New space refinements of the vertices must be defined if we use new tilings. Many modifications of coefficients have been made also to older schemes in order to increase the smoothness or to eliminate some surface artifacts (which appear near extraordinary vertices). Moreover to achieve uniform convergence near the extraordinary vertices, the space refinement must be adjusted.

New surface features like edges, creases, cusps, darts or concave corners and restrictions on the normals were proposed to support better mesh control and more intuitively modelling of subdivision solids.

Fine level feature editing means adding small deformations, lines such as variable width and height creases, on otherwise smooth surface.

Adaptive refinement using new error functions must be researched in addition to decrease the number of faces since this is mandatory for interactive applications to maintain high framerates.

Conversions between subdivision schemes have been observed.

Boolean operations performed on subdivision solids have been just recently described, what emphasizes the use of subdivision schemes in CAGD (see the Constructive solid geometry).

5.2 Commercial software

In nowadays graphics design software there is still not sufficient support of this kind of surfaces. *"TrueSpace"* offers to use the features of subdivision surfaces by providing one, Catmull-Clark like[1], scheme dealing with creases and cusps. That scheme seems to be 1-ring interpolatory. In some cases it produces unwanted protuberances even from a plane. Special feature, we could say concerning the adaptive refinement, was presented – the user can set up minimum degree of two planes, and if for some edge the faces meeting at that edge exceed the limit degree, the refinement across that edge is stopped. It is more like the edge becomes to be fixed boundary on which the object is split and not refined. This kind of subdivision surfaces therefore enable to easy model nice objects, but if you want to work with that object, you have to extract the final mesh and destroy the control mesh, what means it is no longer an subdivision object but other, probably spline patches based solid.

In *"3d Max Studio"* more schemes have been implemented, some similar to Loop, Catmull-Clark and Doo-Sabin, which handle any mesh with crease and cusp features very well (we did not find any unwanted or unexpected artifacts). Nice implementation of adaptive refinement is also available. In *"3d Max Studio"* the subdivision methods are included under the surface modification tools, and so it is still not individual kind of surface which could be used when generating composite scenes – it is a modification which fairs the rough mesh.

For such applications different schemes, smoothness conditions and adaptive refinement schemes are on the focus.

5.3 Summary and future work

According to the last trends, we made a framework for visualizing the process of subdivision surfaces generating. In the implementation part we unified the requirements of the refinement process of major subdivision schemes, thanks what we obtained robust and easy to use application.

With the use of our application several schemes could be observed – their properties and features, and compared to each other. Even some results in the meaning of "adaptive refinement" were presented, where we had to change the refinement rules to obtain schemes dealing with "dead faces".

What is more important, new schemes including various error functions can be easy added and this all machinery can be used in any independent application. Actually the mesh server does not need necessarily to be used just for illustration of subdivision schemes. For any modifications of the mesh, like surface fairing, mesh triangularizations or mesh trimming can the manipulator objects be extended to.

In the future work some improvements mentioned in the implementation section (3.5) may apply, new schemes can be added or even an embedding into some commercial CAGD application could occur.

[1]Different space refinements are used, but the topological refinements reminds the Catmull-Clark scheme.

Appendix A

Detailed subdivision schemes

A.1 Loop

The Loop scheme generalizes quartic box-splines and thus it is approximative and produces C^2-continuous surface over regular meshes. The scheme can be applied to any triangular meshes. In the refinement process each edge is split by inserting a vertex into the middle, after what the 3 new vertices are joined together and split the old face into 4 new triangles.

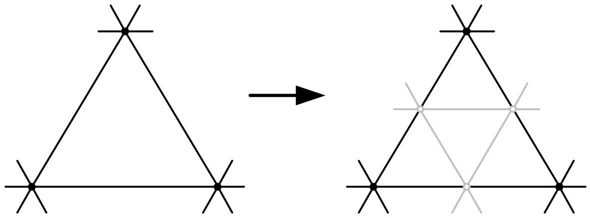


Figure A.1: The refinement of the mesh for Loop scheme.

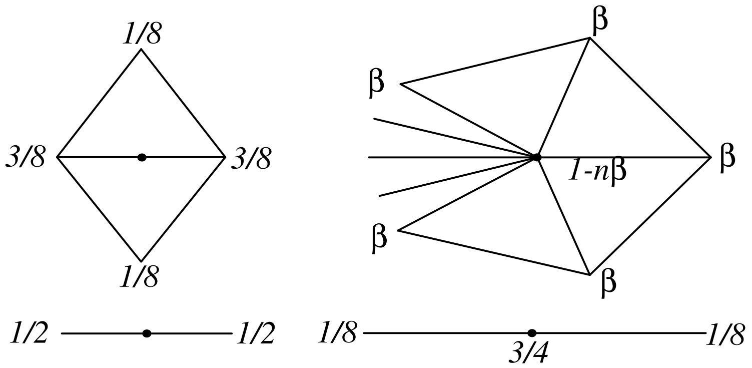


Figure A.2: The stencil masks for Loop. On the left the masks for odd vertices and on the right the masks for even vertices are shown. $\beta = \frac{1}{n}(\frac{5}{8} - (\frac{3}{8} + \frac{1}{4}\cos(\frac{2\pi}{n}))^2)$

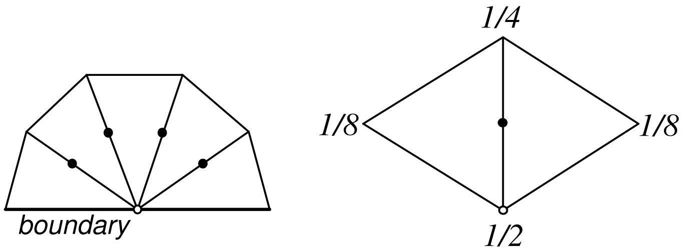


Figure A.3: Modified rule for odd vertices adjacent to a boundary extraordinary vertex.

The Loop scheme was found by C. Loop, later extended by H. Hoppe, T. DeRose and T. Duchamp with special rules defined near edges, and by J. Schweitzer rules for concave and convex corners, cusps, darts, special crease edges have been proposed.

A.2 Catmull-Clark

This approximative scheme is an extension of tensor product bicubic splines and produces C^2-continuous surfaces in the regular setting. This scheme is designed for quadrilateral meshes. In each refinement step each edge is cut in half by inserting a new vertex and a new vertex is also inserted to the center of the face. This vertices are connected and divide the old face into 4 new faces (or into k quadrilateral faces in case the face was a k-gon).

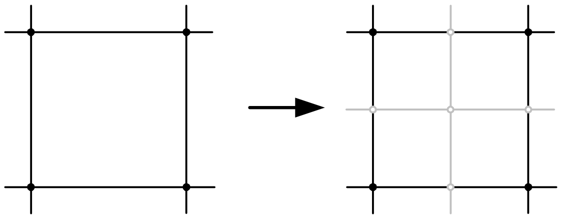


Figure A.4: The refinement of the mesh for the Catmull-Clark scheme.

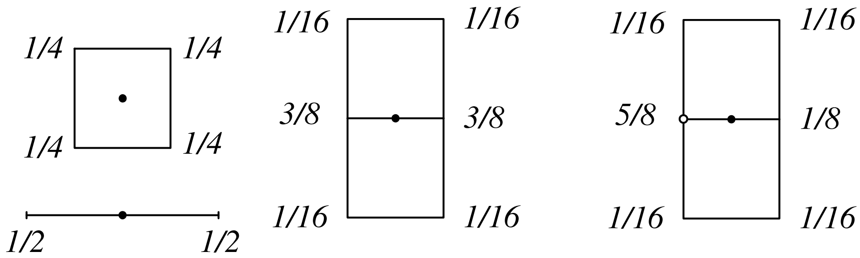


Figure A.5: The stencil masks for the odd vertices of Catmull-Clark scheme together with modified rule for odd vertices adjacent to boundary extraordinary vertex (the hollow circle).

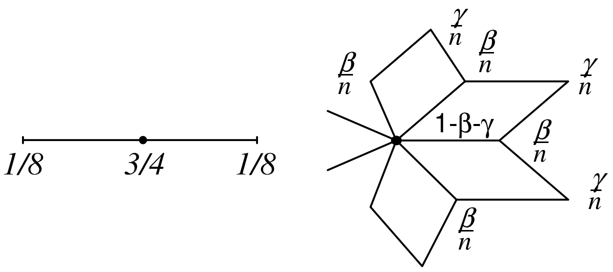


Figure A.6: The stencil masks for even vertices of Catmull-Clark scheme. The suggested coefficients at vertex of valence n are $\beta = \frac{3}{2n}$ and $\gamma = \frac{1}{4n}$.

The Catmull-Clark scheme was founded by E. Catmull and J. Clark and later the smoothness has been analyzed by A. Ball, J. Storry and by J. Peters and U. Reif.

A.3 Doo-Sabin

This scheme generalizes biquartic tensor product B-splines, thus it is an expansion of the Chaikin's scheme for curves. This scheme produces C^1-continuous surfaces. It is interpolatory in the sense that the barycenters of the faces (of the original mesh) are interpolated. This scheme belongs to the vertex-split schemes – note, that the number of extraordinary faces (faces which are not quadrilateral) remain constant. In each refinement step first vertices for each vertex and each face respectively are created, which are then connected to form a quadrilateral mesh.

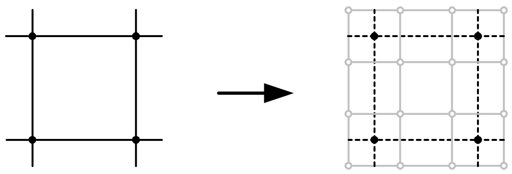


Figure A.7: The refinement of the mesh for Doo-Sabin scheme.

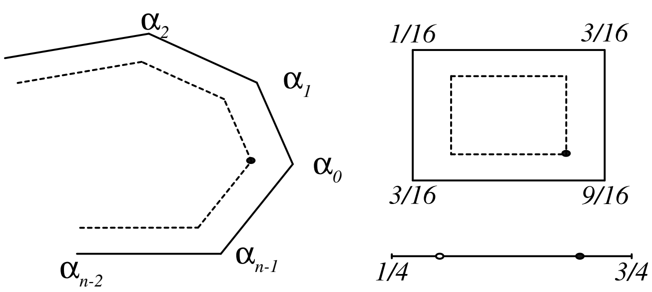


Figure A.8: The stencil masks for Doo-Sabin. $\alpha_0 = \frac{1}{4} + \frac{5}{4n}$,$\alpha_i = (3 + 2\cos(\frac{2i\pi}{n}))/4n$;

The Doo-Sabin scheme was founded by D. Doo and M. Sabin later analyzed by J. Peters and U. Reif.

A.4 Butterfly

The Butterfly scheme is interpolatory and produces C^1-continuous surface over regular meshes. This scheme is similar to the Loop scheme, but larger stencil masks are used. The scheme can be applied to any triangular meshes. In the refinement process each edge is split by inserting a vertex into the middle, after what the 3 new vertices are joined together and split the old face into 4 new triangles.

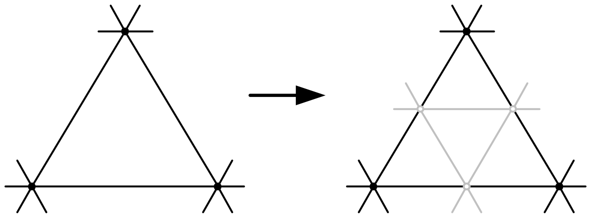


Figure A.9: The refinement of the mesh for Butterfly scheme.

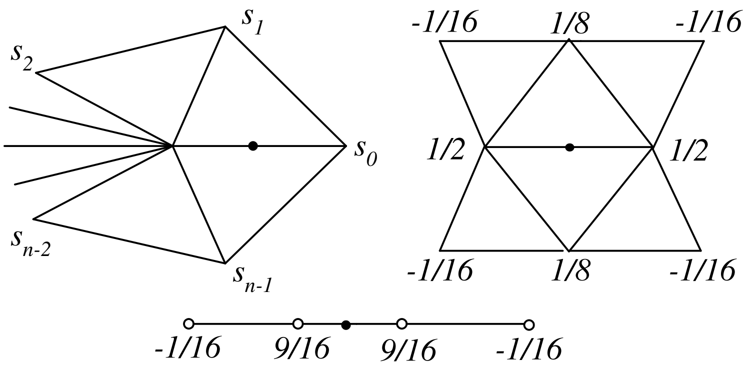


Figure A.10: The stencil masks for Butterfly. Remember, that no rule for even vertices is needed, since the scheme is interpolatory. $s_i = \frac{1}{n}(\frac{1}{4} + \cos(\frac{2i\pi}{n}) + \frac{1}{2}\cos(\frac{4i\pi}{n}))$ *for* $n > 5$, *for* $n = 3$, $s_0 = \frac{5}{12}$, $s_{1,2} = -\frac{1}{12}$; *for* $n = 4$, $s_0 = \frac{3}{8}$, $s_2 = -\frac{1}{8}$, $s_{1,3} = 0$;

The Butterfly was founded by N. Dyn, J. Gregory and D. Levin and later modified by D. Zorin, W. Sweldens and P. Schrörder to produce C^1-continuous surfaces.

A.5 $\sqrt{3}$

The $\sqrt{3}$ scheme was not based on polynomials, but it produces C^2-continuous surface almost everywhere. Every face is split in three by inserting a new vertex at its barycenter. After then the old edges are are flipped yielding a triangle mesh rotated by 30 degrees.

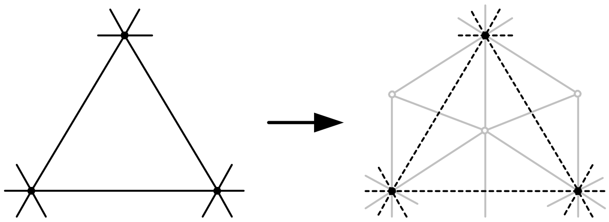


Figure A.11: The refinement of the mesh for $\sqrt{3}$ scheme.

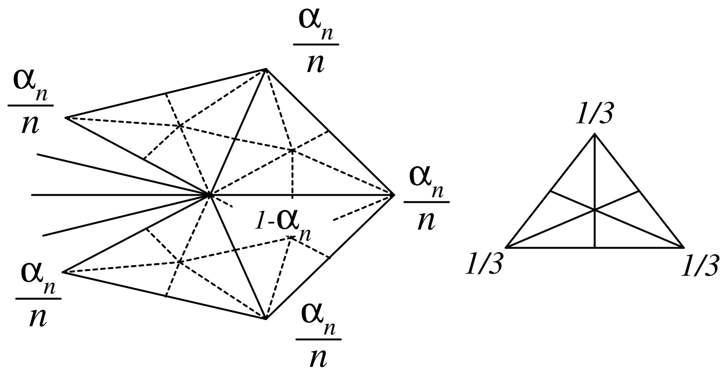


Figure A.12: The stencil masks for the $\sqrt{3}$ scheme. $\alpha_n = \frac{1}{9}(4 - 2\cos(2\pi/n))$ where n is the valence of the center vertex.

The $\sqrt{3}$ scheme was proposed by L. Kobbelt.

Bibliography

[AS02] Akelman, E., Srinivasan, V., Honeycomb subdivision. *Texas A&M University, email: ergun@viz.tamu.edu*, (2002)

[CC78] Catmull, E., and Clark, J. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design 10*, 6 (1978), 350-355

[Cha00] Chalmovianský, P. Subdivision Surfaces in Geometric Modeling. *Proceedings of seminars on Computational Geometry SCG'2000, Volume 9*, (2000), 43–49

[Cha01] Chalmovianský, P. Mathematical Methods in Subdivision Surfaces. *Dissertation thesis, Comenius University, Bratislava*, (2001)

[Cha74] G. M. Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3 (1974), 346–349

[DS78] Doo, D., and Sabin, M. Analysis of the Behaviour of Recursive Division Surfaces near Extraordinary Points. *Computer Aided Design 10*, 6 (1978), 356-360

[Has02] Hašan, M. Subclasses of F-rep Surfaces Allowing for Faster Rendering *Proceedings of the $6^t h$ Central Europan Seminar on Computer Graphics*, Budmerice, Slovakia (2002), 169-181

[Kob96] Kobbelt, L. A variational approach to subdivision. *CAGD 13*, (1996), 743–761

[Kob96b] Kobbelt, L. Discrete fairning and variational subdivision for freeform surface design. (1996)

[Lev00] Levin, A. Combined subdivision schemes. *PhD thesis, Tel-Aviv university*, (2000)

[MSDN] Microsoft Component Object Model. *MSDN, Technical Articles*, (2002)

[PS96] Pulli, K., and Segal, M. Fast Rendering of Subdivision Surfaces. *7th Eurographics Rendering Workshop, Porto, Portugal*, (1996), 61–70

[Rei95] Reif, U. A unified approach to subdivision algorithms near extraordinary points. *Computer Aided Geometric Design*, (1995), 12:153–174

[Rei95b] Reif, U. TURBS – Topologically Unrestricted Rational B–Splines. *Tech. report, Universität Stuttgart, Mathematisches Institut A*, October (1995)

[Rei96] Reif, U. A degree estimate for polynomial subdivision surfaces of higher regularity. *Proc. Amer. Math. Soc.*, (1996), 124:2167–2174

[Rei99] Reif, U. Analyse und Konstruktion von Subdivisionalgorithmen für Freiformflächen belibiger Topologie *Sheker Verlag. Habilitationsschrift*, (1999)

[War95] Warren, J. Subdivision methods for geometric design. *Unpublished manuscripts*, November (1995)

[Zor98] Zorin, D. Stationary Subdivision and Multiresolution Surface Representations. *Ph.D. thesis by Denis N. Zorin, California Institute of Technology*, (1998)

[ZSD+00] Zorin, D., Schröder, P., DeRose, T., Kobbelt, L., Levin, A., and Sweldens, W., Subdivision for modeling and animation. *SIGGRAPH 2000. Course Notes.*, July (2000), 13–114

[XK02] Zheng Xu, Kunio Kondo, Local Subdivision Process with Doo-Sabin Subdivision Surfaces. *Submit to SMI2002, Banff, Alberta, Canada*, May (2002)