Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics
Department of Computer Graphics and Image Processing

Master Thesis

Author: Peter Szinek
Supervisor: RNDr. Pavel Chalmovianský, PhD.

Bratislava, April 2003

I honestly declare that I have written the submitted mater thesis by myself and I have used only the literature mentioned in the bibliography

.....
Peter Szinek

I wish to thank to my supervisor Pavel Chalmovianský
for his assistance in the preparation of my master thesis

Contents

Chapter 1

Introduction

1.1 Overview of Surface Generation

There is a lot of surface generation methods in area of geometric modeling. We briefly review them in order to see their important features. After introducing some other commonly used surface generation methods, we focus on subdivision surfaces and their rendering time improvements.

1. **Implicit surfaces:**
   The set of surface points S_c is defined

   $$S_c = \{[x, y, z] \in R^3 \mid f(x, y, z) = c\}, f : R^3 \rightarrow R \ (continuous)$$

   (e.g. $S_c : f(x, y, z) = x^2+y^2+z^2$ over the domain R^3 for some parameter c - represents sphere with radius c. Implicit surface representation defines the surface as the zero contour of 3 variables.
   Advantages of this approach are:

   - we can easily check whether the point is 'inside' or 'outside' the surface - if $f(x, y, z) < 0$ the point is 'inside', if $f(x, y, z) > 0$ it is 'outside'
   - we can relatively easily calculate the tangent plane, normals, curvature explicitly at any given point
   - good at operations such as space deformation, blending or morphing

   Disadvantages are the following:

   - it is difficult to look for points on the surface

- hard (or impossible) to define bounded portion of an implicit surface
- problem to join piecewisely such surfaces, while maintaining smoothness and continuity

Other popular usage of implicit surfaces are *metaballs* or *blobby objects*. We can imagine a metaball object as a particle surrounded by a density field, where the influence of the density decreases from the particle location. The really powerful feature of metaballs is that they can be combined easily (See figure 1.1) by summing the influences of each metaball on a given point, producing smooth blendings of the two spherical fields. Metaballs are widely used for creating organic models

Figure 1.1: Combining of two 'metaball' objects

(trees, creatures...) but due to their hard explicit parametrization they are not used much in geometric modeling.

2. **Parametric surfaces** - defined as

$$Q : \Omega \in R^3 \rightarrow \mathcal{S} \in R^3$$

$$Q(u,v) = [x(u,v),\ y(u,v),\ z(u,v)], u, v \in [0,1],$$

where x, y, z are real functions defined over the plane $\Omega[0,1] \times [0,1] \subseteq R^2$. Although these surfaces are easy to visualize (simply by evaluating the function f in sufficiently many points and joining the resulting points with polygons) the way of visualization is relatively expensive. Among the advantages are the ability to calculate derivates at any given point, compute intersections. The drawbacks include expensive visualization and bad local control of the generated surface.

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

   Bézier patches If we are having $(n+1)*(m+1)$ control points $P_{i,j}$ arranged in a control grid, we are defining the Bézier tensor-product patch as

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

where B_i^k are the Bézier basis functions:

$$B_i^k(t) = \begin{pmatrix} k \\ i \end{pmatrix} t^i (1-t)^{k-i},\ t \in R,\ k \in N_0,\ i \in \{0, 1, \ldots, k\}$$

Bézier patches are easy to model by the aid of the control grid, easy to differentiate at any given point. It is also relatively easy to compute intersections of such surfaces.

Problems arise when trying to join two patches (with prescribed continuity) into a single surface - special rules are needed on the boundaries (defined by the boundary control points) to preserve the required smoothness. Another common disadvantage is the limited variety of shapes that can be modeled with Bézier patches - some quadrics (such as sphere or ellipsoids) are not available. The local control of the generated surface is also not sufficient - by changing the coordinates of one control point, the whole shape is affected.

B-spline surfaces, defined as

$$Q(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} N_i^n(u) N_j^m(v), \qquad u, v \in [0,1],$$

where $P_{i,j},\ i \in \{0, 1, \ldots, n\}\ j \in \{0, 1, \ldots, m\}$ are the control points and N_i^k are the B-spline basis functions:

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

$$N_i^k(t) = \frac{(t - t_i)}{(t_{i+p} - t_i)} N_i^{k-1}(t) + \frac{(t_{i+p+1} - t)}{(t_{i+p+1} - t_{i+1})} N_{i+1}^{k-1}(t)$$

k is the order of the polynomial segments of the B-spline curve

$$t \in [t_{k-1}, t_{n+1}), \{t_i : i = 0, \ldots, n+k\}$$

B-spline surfaces have many advantages over the previously defined Bézier patches: Mainly local control (changing the coordinates of a control point does not affect the entire surface, just a smaller portion around the control point). Also these surfaces are invariant with respect to affine transformations.

Non Uniform Rational B-Spline (NURBS) surfaces Defined over B-spline basis functions as

$$Q(u,v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j}\ P_{i,j}\ N_i^n(u) N_j^m(v)}{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j}\ P_{i,j}\ N_i^n(u) N_j^m(v)}, \qquad u, v \in [0,1],$$

where $P_{i,j}$ are the control points and $w_{i,j}$ are the weights for each control point. If the denominator equals to zero, $Q(u, v) = 0$ by definition. The usage of the rational coefficients makes possible the generation of even larger variety of surfaces as in the case of B-splines - conic/spherical sections for example, such as quadrics.

4. **Procedural surface representations**
   The formerly introduced methods described the surface with different mathematical methods. The approach of procedural representations is different - the input is a mesh (mesh - net of points in the object space defining a solid body - for more rigorous definition please refer to section 2.1) and a set of rules working on this mesh, producing finer approximation of the desired object. Typical application of this model are subdivision surfaces, fractal generation methods, Lindermayer systems (L-systems). In the next sections we will go through the advantages and disadvantages of subdivision surfaces, and compare them with the introduced ones.

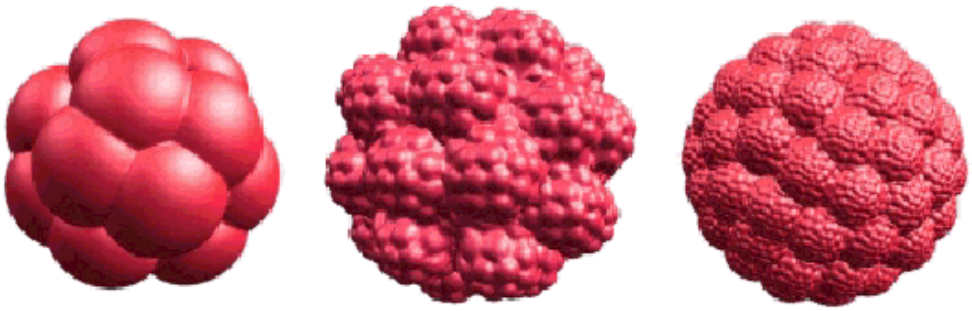

Figure 1.2: Example of a berry created with Procedural shape synthesis on subdivision surfaces

1.2 Advantages of Subdivision Surfaces

Constructing surfaces through subdivision brings a lot of advantages over the formerly described methods:

Arbitrary topology Subdivision generalizes classical spline patch approaches to arbitrary topology. In practice this means efficient solution of problems introduced joining two or more patches together with a desired

continuity (C^2 or higher, since C^1 is always possible) or curve trimming for instance.

Uniformity of representation There are two mainstream approaches of solid modeling in traditional CAGD[1]: polygonal meshes or spline patches. Subdivision offers the "golden ratio" between these two approaches: Surfaces can behave as if they are made of patches, or as if built up from polygons.

Numerical stability The surfaces produced by subdivision have a lot of the "nice" properties that finite element solvers require. As a result of this, subdivision surfaces are highly suitable for various numerical simulation tasks.

Multiresolution surfaces generation Because of its recursive structure, subdivision naturally accommodates LOD[2] rendering and adaptive approximation with error bounds. Therefore subdivision algorithms are able to work depending on the hardware configuration, thus producing sufficient results also in the case of limited hardware resources.

Special surface features Modified subdivision schemes have been proposed (for example combined subdivision schemes, see [8]) to control the shape and size of features, such as creases, grooves, sharp edges, cusps, darts and more by manipulating the subdivision coefficients. For more details see also [10].

Code simplicity is very important for implementation and practical usage. Although the mathematical machinery behind subdivision is of high level, implementation and run time performance is efficient.

1.3 Disadvantages of Subdivision Surfaces

Bad explicit parametrization is the source of a lot of disadvantages subdivision surfaces come with.

Determining various geometric properties such as length, curvature, explicit analytical formulas for tangent vectors and surface normals. At the moment there are no known general formulas for computing these values, although formulas for determining the end positions of vertices, computing the normal and tangent vectors have been proposed for each

[1]Computer Aided Graphics Design
[2]Level-of-Detail

subdivision scheme, these are working only at the control points of the mesh at arbitrary level. Therefore if one wants to determine these values at a random surface point, approximative methods have to be used.

Continuity of the generated surface Schemes proposed so far produce C^2 or even C^3 continuous surface almost everywhere - problems arise with continuity at extraordinary[3] vertices, where only C^1 continuity is ensured, which is not sufficient in most of the cases. There have been proposals for C^2 continuous schemes, it has been proved that such schemes have large support or necessarily have zero curvature at extraordinary vertices.

Computing intersections, differences and other boolean operations widely used in geometric modeling (e.g. in CSG [4] or F-rep) is still an open question - no general solution of this issue exists at the moment.

Joining of subdivision surfaces with required smoothness is a related problem. To achieve smooth connections, special rules have to be used at the boundaries - combined subdivision themes (see [8]) and piecewise smooth subdivision surfaces with normal control (see [2]) are proposals for the solution of this issue.

Large amounts of data Basically there are two main types of subdivision schemes: primal (face split) or dual (vertex split). In either case, the number of faces of the mesh grows linearly in each subdivision step based on the number of faces (primal scheme) or number of vertices (dual scheme). This fact and the recursive structure of subdivision implies the polynomial growth of the number of faces during the subdivision process. Therefore either adaptive refinement rules or variational schemes have to be used.

[3]Vertices with valence (number of edges meeting at this vertex) more or less than 4 (for quadrilateral meshes) or 6 (for triangular meshes)

[4]Constructive Solid Geometry

Chapter 2

Mathematical Backgrounds of Subdivision

Before describing subdivision surfaces and it's properties, we will take a look at the mathematical description of subdivision process. For more detailed work refer to [12], [3].
Subdivision is defined upon the cell complex theory, therefore in the next chapter we will define this structure in detail. Based on these definitions, we will define the mesh as an application of this abstract structure - embedding in to vector space R^3.

2.1 Cell Complexes

Let $\mathcal{A}$ denote a set. The set of all possible selections of n elements created from the elements of the set $\mathcal{A}$ without repetition will be denoted $\mathcal{A}^n$. By $(a_0, \ldots, a_{n-1})$ we denote the elements of $\mathcal{A}$, called **ordered n − tuples**. Let $\mathcal{A}^{[n]}$ be the set of all *cyclically* and *reverse unordered* $n - tuples$. The elements of $\mathcal{A}^{[n]}$ are denoted $[a_0, \ldots, a_{n-1}]$, where $a_i \in \mathcal{A}, i \in Z_i$. The set $\mathcal{A}^{[n]}$ is the set $\mathcal{A}^n$ factored by relation of equivalence $\simeq$ generated by relations:

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

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

According to this $\mathcal{A}^{[n]} = \mathcal{A}^n/_{\simeq}$.

We say that the element $[b_0, \ldots, b_{k-1}]$ is a **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 $(a_0, \ldots, a_{n-1}) \simeq (b_0, \ldots, b_{k-1}, c_0, \ldots, c_{n-k-1}$. We say that $[b_0, \ldots, b_{k-1}]$ is **incident** with $[a_0, \ldots, a_{n-1}]$ and vice versa.

Definition 2.1.1 *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 - called also the set of 0-faces,*

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

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

satisfying the following conditions:

1. *if $u \in \mathcal{V}$, there exists $v \in \mathcal{V}, u \neq v$ such that $[u, v] \in \mathcal{E}$ (that is, there are no isolated vertices),*
2. *if $[v_0, v_1, \ldots, v_{k-1}] \in \mathcal{F}$, then $[v_0, v_1], [v_1, v_2], \ldots, [v_{n-2}, v_{n-1}], [v_{n-1}, v_0] \in \mathcal{E}$ ($\mathcal{E}$ must contain all edges of a face)*
3. *$[u, v] \in \mathcal{E}$, then there is a face $F \in \mathcal{F}$ such that $[u, v] \subset F$ (all edges are contained in a face),*
4. *if $[u, v] \in \mathcal{E}$, there are no more than two faces $F_1, F_2 \in \mathcal{F}$ such that $[u, v] \subset F_1$ and $[u, v] \subset F_2$ (maximally 2 faces are sharing an edge),*
5. *if $v \in \mathcal{V}$ and the set $Loop(v) = \{u \in \mathcal{V} \mid \exists F \in \mathcal{F} \ \wedge \ v \in F \ \wedge \ u \in F \ \wedge \ u \neq v\}$ is finite with k elements $u_0, \ldots, u_{k-1}$, then there exists an ordering of elements $(u_0, \ldots, u_{k-1})$ of Loop(v) so that $[u_i, u_{i+1}]$ is edge of $F \in \mathcal{F}$ with $v \in F \ \forall i = 0, \ldots, k-2$.*

Definition 2.1.2 *A **Mesh M(V,E,F)** is a special case of cell complex - the vertex set is made up from points in R^3. An edge e is called a boundary edge of M(V,E,F) if it is not shared by two faces. A vertex v is called boundary vertex if it belongs to a boundary edge. The mesh is said to be closed if it has no boundary edges. Otherwise, M(V,E,F) is an open mesh.*

The valence of a vertex is the number of edges meeting at this vertex. There are 2 types of vertices: regular *or* extraordinary*. If the valence of a vertex equals with the preferred vertex valence of the scheme (6 for triangular, 4 for quadrilateral) the we say that the vertex is regular, otherwise it is extraordinary.*

Similar definition of the can be found in [5]. For an illustration see Figure 2.1.

Definition 2.1.3 *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 an **abstract dual cell complex** to complex $\mathcal{C}$ if and only if:*

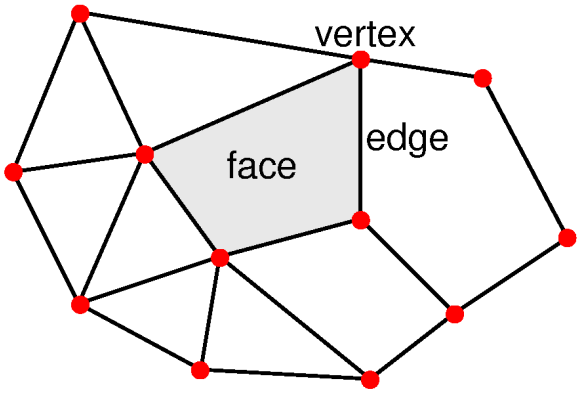

Figure 2.1: A mesh

1. $\mathcal{V}' = \mathcal{F}$,
2. $[u', v'] \in \mathcal{E}' \Leftrightarrow \exists [u, v] \in \mathcal{E}, [u, v] \subset u' \wedge [u, v] \subset v'$
3. *for each* $v \in \mathcal{V}$ *there is corresponding face* $F_v \in \mathcal{F}'$ *such that* $v' \in F_v \Leftrightarrow v \subset v'$.

There are various special kinds of cell complexes based on the above definition. Let us mention a few that are used in the later presented subdivision schemes:

- *If each face* $F \in \mathcal{F}$ *is a 2-simplex (face is a 2-simplex if* $\mathcal{F} \subset \mathcal{V}^{[3]}$*), we say that the cell complex is an* ***abstract simplical complex***
- *If each face* $F \in \mathcal{F}$ *is a quadrilateral complex (that is,* $\mathcal{F} \in \mathcal{V}^4$*), the cell complex is called a* ***quadrilateral cell complex***
- *The number of edges incident with every non-boundary vertex is four. This type of cell complex is called* ***dual quadrilateral complex***

Definition 2.1.4 *A* ***subcomplex*** $\mathcal{C}' = (\mathcal{V}', \mathcal{E}', \mathcal{F}')$ *of a complex* $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ *is a complex satisfying* $\mathcal{V}' \subseteq \mathcal{V}, \mathcal{E}' \subseteq \mathcal{E}, \mathcal{F}' \subseteq \mathcal{F}$.

Definition 2.1.5 *Let* $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ *be a complex. An* ***1-neighborhood*** $N_1(V) = (V', E', F')$ *of a set of vertices* $V \subset \mathcal{V}$ *is the subcomplex of* $\mathcal{C}$*, where*

$$\mathcal{F}' = \{F \in \mathcal{F} \mid \exists v \in \mathcal{V} \text{ incident with } F\}$$

$$\mathcal{E}' = \{\ e \mid e \subset \mathcal{F}'\ \}$$

$$\mathcal{V}' = \{\ v \mid v \in \mathcal{F}'\ \}$$

Based on this definition, we define the ***k-neighborhood*** *(or* ***k-ring neighborhood*** *$N_k(V)$ of a set of vertices $V \subset \mathcal{V}$ recursively as the 1-neighborhood of all the vertices of (k-1)-neighborhood of the set V.*

For an example see Figure 2.2

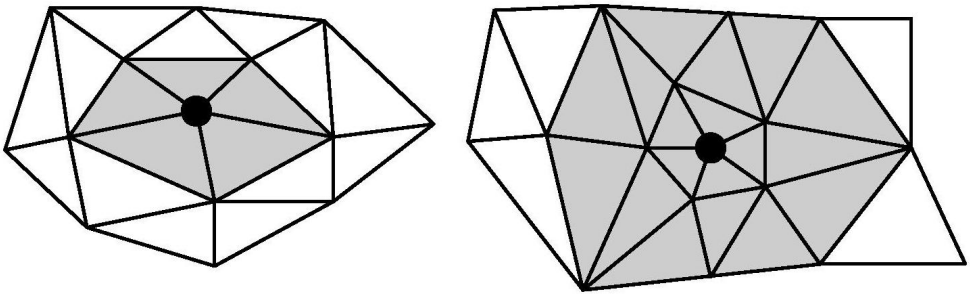

Figure 2.2: The 1-ring and 2-ring neighborhoods of a vertex

2.2 Refinement of Cell Complexes

In section 3.2 the concept of face and vertex split refinement is defined, which is used by the majority of stationary schemes. However, there are some recently proposed schemes that are not obtained using these ($\sqrt{3}$, Velho's 4-8, Honeycomb). To extend the refinement rules to be able to handle a bigger variety of schemes we will introduce tilings of the plane.

The starting point for refinement rules are the *isohedral tilings* (see 2.3) and their dual tilings, called also *Archimedian tilings*. 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.

Definition 2.2.1 *Every cell complex 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 $|\mathcal{C}|$.*

Definition 2.2.2 *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 $|\mathcal{C}'|$.*

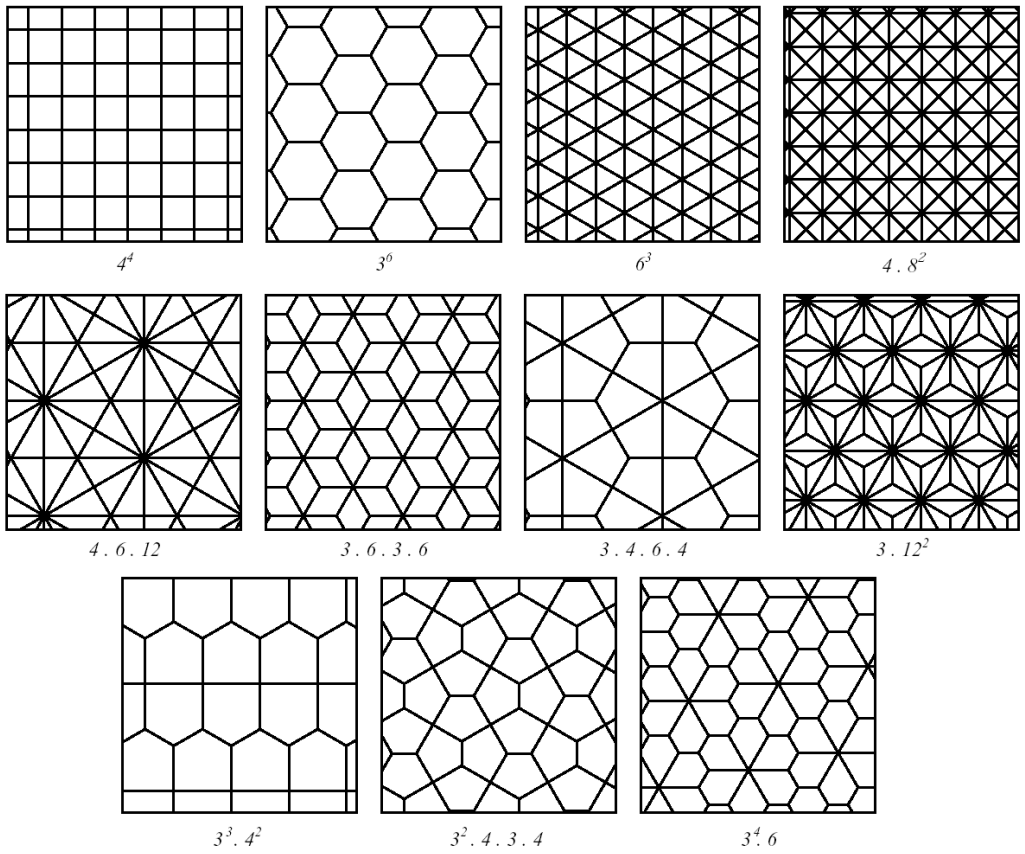

Figure 2.3: Isohedral (Leaves) tilings

2.3 Subdivision Scheme

Before going to define the notion of subdivision scheme, we will define the embedding of cell complexes into the Euclidean affine vector space over some special field (usually R). Cell complexes are used as a mathematical tool to describe topology as an abstract domain and define topological operations at it - especially refinement process for obtaining finer domain.

The need for embedding of cell complexes into the vector space arises when we want to define a smooth surface over this domain - mostly a mesh. We prescribe this embedding in the following definition:

Definition 2.3.1 *Let $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ be an abstract cell complex, $\mathcal{B}$ a vector space over field R.* ***Function defined over the cell complex*** *$\mathcal{C}$ is a map $f : \mathcal{V} \to \mathcal{B}$. We will denote the vector space of all these functions over the field R as $\mathcal{P}(\mathcal{V})$.*

The function over a cell complex can be thought of as a mesh, which is the embedding of the graph into the modeling space. There is an infinite possibility to perform such embedding. The set of all such embeddings forms a vector space. Figure 2.4 illustrates one possible mesh $M = (V, E, F)$ as the embedding of the cell complex $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$:

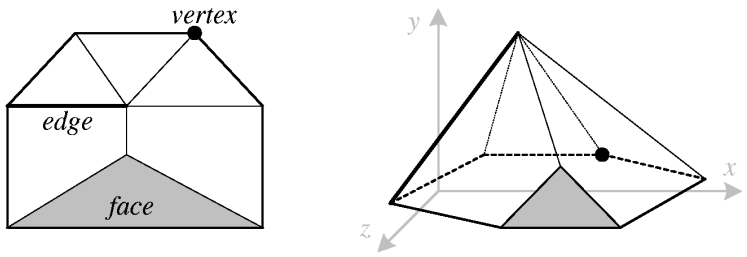

Figure 2.4: Cell complex and its corresponding mesh

Definition 2.3.2 *Let* $\boldsymbol{G}$ *be a set of cell complexes together with continuous maps between them, let* $\Re : \mathbf{G} \to \mathbf{G}$ *be the refinement operator 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 we denote* $\mathcal{L}[G] = \bigcup_{\mathcal{C}\in\mathbf{G}} \mathcal{L}[\mathcal{C}]$*, and let* $\Im : \mathbf{G} \to \mathcal{L}[\mathbf{G}]$ *be an operator for each* $\mathcal{C} \in \mathbf{G}$ *such that* $\Im[\mathcal{C}] \in \mathcal{L}[\mathcal{C}]$*.*
Subdivision scheme $\boldsymbol{S}$ *is a pair* $(\Re, \Im)$ *with topological operator* $\Re$ *and relaxation operator* $\Im$ *defined over* $\boldsymbol{G}$*.*

Definition 2.3.3 *Let* $\boldsymbol{S} = (\Re, \Im)$ *be the subdivision scheme over* $\boldsymbol{G}$*,* $\mathcal{C}_0 \in \mathbf{G}$ *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* $i \in N$ *is called* ***subdivision process*** *of a pair* $(\mathcal{C}_0, f_0)$*.*

Definition 2.3.4 *Let* $\mathcal{C}_0 \in \mathbf{G}$ *and* $f_0 \in \mathcal{P}(\mathcal{V})$*, over the vector space* $\mathcal{B}$*, for a given set* $\boldsymbol{G}$ *of cell complexes together with continuous maps between them. We say that the subdivision scheme* $S{=}(\Re, \Im)$ ***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 $v^* \in \mathcal{V}^*$ *corresponding to* $v \in V_i$ *in homeomorphism of* $\mathcal{C}_0$ *and* $\mathcal{C}_i$ *in the refinement, and* $C^* = \lim_{i\to\infty} \mathcal{C}_i$ *(supposing topology of* $\mathcal{C}_i$ *does not change 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* $(\mathcal{C}_0, f_0)$*.*

Chapter 3

Basic notions

3.1 The Idea of Subdivision

Subdivision is a method for generating smooth surfaces by producing denser and denser net of surface points. Although originally it was introduced as an extension of splines to arbitrary topology control nets it is much more general than the knot insertion algorithm used for spline generation and therefore it offers a wider variety of surfaces with specific properties.

We can say that the subdivision is based on iterated transformations. Let $\mathcal{G}_i, i \geq 0$ be a sequence of geometrical shapes. Let F be a function defined on shapes, that maps geometrical shape $\mathcal{G}_i$ into another geometrical shape $\mathcal{G}_{i+1}$. If $\mathcal{G}_0$ is the initial mesh, we define the infinite sequence of meshes as

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

If F meets certain requirements, there exists a limit mesh $\mathcal{G}$ that is a fixed point of F:

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

Based on the above definition, we can think of the basic idea of subdivision as follows:

Subdivision method defines a smooth curve or surface as the limit of a sequence of successively refined polyline or polyhedral meshes.
See figure 3.1 for an illustration.
The refined meshes are generated by adding new vertices to the mesh and connecting them to form new edges and new faces according to special rules (see Section 3.2 for an overview of different approaches). There are many possible ways to determine this, as far as we consider the following rules:

Efficiency: The speed of the real-time performance is one of the most important characteristics of a subdivision scheme. Therefore, the loca-

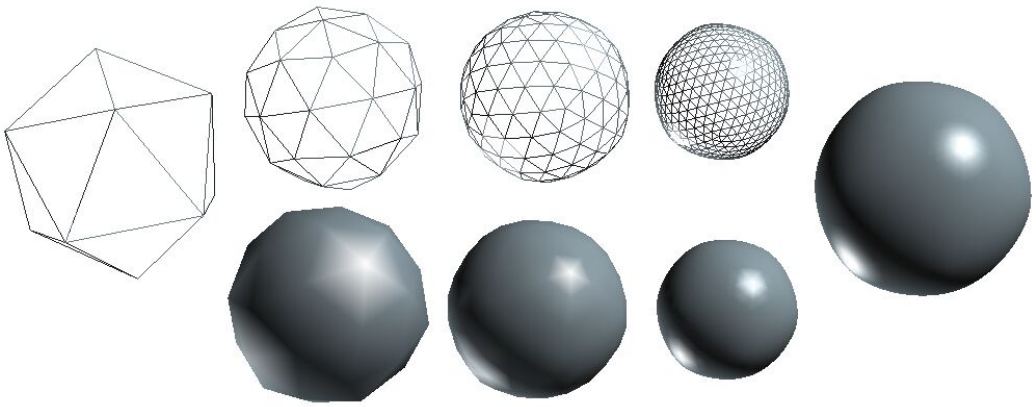

Figure 3.1: An example of Loop's subdivision scheme applied on a very rough geosphere (4 iteration steps are shown - left: the initial mesh, middle: 3 successive refinement steps, right: the 4th level of the subdivision)

tion of the new points should be computed with the least small possible number of processor-demanding operations (typically floating point operations).

Local definition: The rules for determining the position of new vertices should depend only on the structure of a finite part of neighborhood of this point. This also means that for parts of the mesh with the same structure we can use the same rules.

Compact support: We say that the scheme is of compact (or finite) support if the region over which a point influences the resulting mesh is finite. Moreover this region is required to be as small as possible.

Simplicity of subdivision rules: Determining the rules should be preferably an offline process. The rules should be as simple as possible from the mathematical point of view. Also we require a small number of rules.

Another important properties of a subdivision scheme should be: **affine invariance**, **continuity** (At least C^1) everywhere, and **convergence**.

For more detailed work on this topic one should consult [12], [9], [13], [8].

3.1.1 Subdivision Example on Uniform B-splines

Before moving on the 3D setting, let us see an example of subdivision defined on curves. Chaikin's corner cutting algorithm (for further details refer to [4], invented in 1974 is the oldest (and one of the simplest) known subdivision scheme working on a polyline (for illustration see Figure 3.2. The position of the new points at level k is computed as a linear combination of old points at level $k-1$:

Figure 3.2: Chaikin's corner cutting algorithm

$$p_{2k}^{i+1} = \frac{3}{4}p_k^i + \frac{1}{4}p_{k+1}^i \qquad p_{2k+1}^{i+1} = \frac{1}{4}p_k^i + \frac{3}{4}p_{k+1}^i \tag{3.1}$$

It is a proven fact that the introduced scheme converges to a C^1 quadratic B-spline curve.

Now we will have a look at the subdivision property, offered by uniform B-splines. For the derivation of the equations and related work one is referred to [11] pp.11-17 or [13] pp.22-29. Let us have a look at the definition of the quadratic B-spline basis functions defined over uniformly spaced knot sequence $[\ldots, 0, 1, 2, 3, \ldots]$.

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

For an illustration, see figure 3.3. Our aim is to define the same linear space

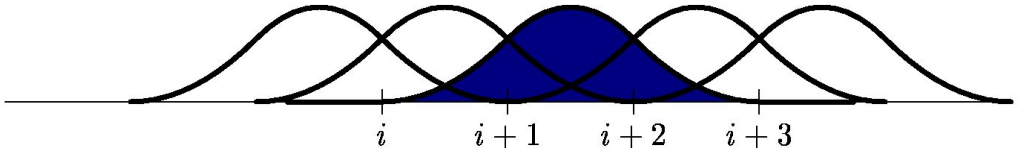

Figure 3.3: Uniform quadratic B-spline basis function

over refined knot sequence $[\ldots, 0, 1, 2, 3, \ldots]$ (see figure 3.4) and determine the matrix of transformation from one space to the other. In [13] we can found the definition of the refinement equation for B-splines of degree n:

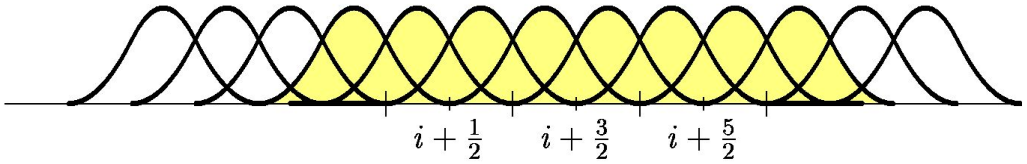

Figure 3.4: Uniform quadratic B-spline basis function defined over refined knot sequence

$$\mathbf{N}^n(t) = \frac{1}{2^n} \sum_{k=0}^{n+1} \begin{pmatrix} n+1 \\ k \end{pmatrix} \mathbf{N}^n(2t-k)$$

It can be proven that the old basis function satisfies the above presented refinement equation for $n = 2$:

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

in terms of dilates

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

If $N^2(t)$ denotes a row vector whose i-th component is $N_i^2(t)$, we can express the last equation in terms of matrices as

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

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

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

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

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

The B-spline curve can be denoted as

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

By repeating this process

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

We can derive easily the relationship between the control points at different levels:

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

The relation between the old control points p_i^j and the new control points p_i^{j+1} is a *subdivision* formula and the matrix $\mathcal{S}$ is called the *subdivision matrix* associated with the above described process. We can see that transforming the old control points with the subdivision matrix $\mathcal{S}$ the resulting control points are determined as specified in Chaikin's algorithm (see equation 3.1).

3.2 Terminology and Basic Properties

In this chapter we will describe the basic properties of subdivision surfaces, introducing the terms used in subdivision theory.

A subdivision scheme working on a mesh $M(V, E, F)$ is producing a new mesh in each step $M'(V', E', F')$. Therefore we need rules to calculate the new vertices, edges and faces. During the subdivision process, each new vertex V' is calculated as a weighted average of vertices of V. The set of weights is called a *stencil* or *stencil mask* (see figure 3.5).

A very important aspect of a subdivision scheme is whether it is *interpolating* or *approximating*. If the vertices of the control net do not lie on the surface

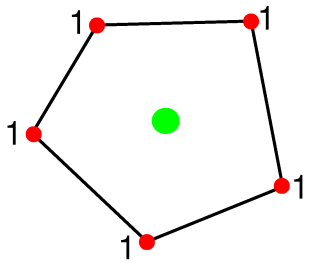

Figure 3.5: Stencil mask

itself, we are saying that the scheme is *approximating*. The main advantage of an approximation scheme that it produces very smooth surfaces, smoothing out ripples and undulations, even if the original control net was dense with sharp points. On the other hand, modeling can be more difficult to that of an interpolating scheme, because it is hard to envision the result by looking at the shape of the control net, and hard to model ripples and sharp artifacts as the scheme tends to smooth them out. Another advantage of approximating schemes is that they converge faster to the limit surface than interpolating ones (see [13] for more details).

If the vertices of the control net lie on the limit surface, the scheme is *interpolating*. This means that the rules of an interpolating scheme do not move the points of the control net at any subdivision level. The quality of surfaces produced with interpolating schemes is lower compared to approximating schemes.

Another interesting point is the *subdivision shape* of the scheme, which defines how a refined tiling is related to the original tiling. Most of the existing schemes are triangular or quadrilateral, although the most quadrilateral schemes have rules for subdividing n sided polygons, for $n > 4$. If one wants to use a triangular scheme on a non-triangular mesh, has to triangulate it in advance. This task seems to be trivial, but in fact it is not, as some schemes produce slightly different results with different triangulations of the same mesh.

Once the used tiling is chosen, we have to define the relation between the subdivided tiling and the original one. There are two main approaches: *face split* (also called *primal*) schemes and vertex split (or *dual*) schemes (see figure 4.1), but also different types are available ($\sqrt{3}$, Velho's 4-8, Honeycomb and more).

Primal schemes split every face into 3 (in case of triangular mesh) or 4 (in case of quadrilateral mesh). New vertices are inserted on the edge (in the case of a quadrilateral setting one vertex is added in the middle of each face).

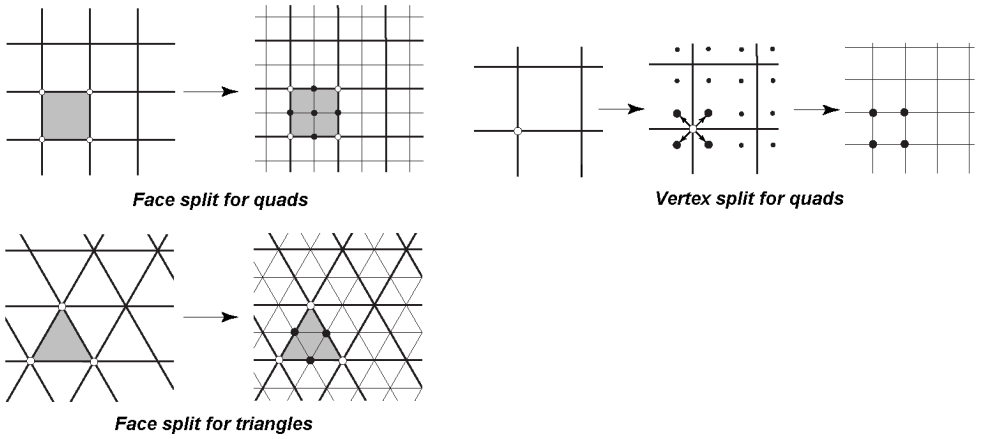

Figure 3.6: Different refinement rules

After this setup, the new vertices are joined to create the refined mesh (old vertices are retained). In the second case, new vertices are created around every old vertex, one for each face adjacent to the vertex. A new face is created for every edge, the old faces are retained - and finally, a new face is created for each vertex.

For primal schemes, we call the new vertices created at the edges *odd* vertices, while the inherited vertices from the previous level are called *even*. For quadrilateral schemes, some vertices are inserted when edges of the coarser mesh are split and some are inserted for a face. Therefore these two types of odd vertices are called *edge* and $face$ vertices, respectively.

The method of applying the subdivision rules introduces some more terms: If we apply the same set of rules throughout the whole mesh, we are using an $uniform$ scheme. If (at least) two different rules are used to subdivide the mesh, we are talking about a $non-uniform$ scheme. *Stationary* schemes use the same set of rules in every subdivision step, whereas $non-stationary$ schemes might subdivide the mesh with different set of rules at each step.

The mainstream subdivision schemes are all fundamentally uniform and stationary. There are extensions and modifications of these schemes that make them non-uniform or non-stationary, but only a few schemes exist that are fundamentally non-uniform or non-stationary. The problem is that the majority of the mathematical tools used for defining and analyzing schemes are unable to work with dynamically changing rules.

3.3 Overview of Existing Subdivision Schemes

In this section we will give basic classification of present known subdivision schemes generating C^1-continuous surfaces on arbitrary meshes. Later we will give a detailed description classic triangular schemes - mostly the ones used in the software attached to this book.

3.3.1 Subdivision Classification

At first sight, the classification might look difficult because of the wide variety of existing schemes. We will use four key characteristics, already defined in the previous section:

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

Based on these criteria, the following table summarizes the mostly used known subdivision schemes:

Face split schemes		
	Triangular meshes	*Quadrilateral meshes*
Approximating	Loop(C^2)	Catmull-Clark(C^2)
Interpolating	Modified Butterfly (C^1)	Kobbelt(C^1)

Overview of face split subdivision schemes

Vertex split
Doo Sabin (C^1), Bi-quartic (C^3), Midedge (C^1)

Overview of vertex split subdivision schemes

Smoothness C^i means that the scheme produces C^i smooth surface in the regular setting, but only C^1 smoothness is achieved near extraordinary vertices.

The above presented list of schemes is by far not exhaustive - it mentions only the major stationary triangular and quadrilateral schemes. Besides this classification, a lot of other schemes exist - $\sqrt{(3)}$ scheme by L. Kobbelt, 4-8 scheme by L. Velho and J. Gomes, Honeycomb by E. Akleman and V. Srinivasan, some de Rahm generalizations of Doo-sabin like schemes.

So far we were talking about stationary schemes only. Now we will briefly mention some other, non-stationary schemes:

Combined subdivisions were introduced by A. Levin (See [Lev00] for further details). They play a great role in prescribing finer behavior at the border and the interior - sharp features for example, which are normally unavailable. These schemes introduce combination of surface generation with arbitrary curves. By the refinement step the regard for boundary conditions is taken to compute the new vertices. The combination of the vertices of refined mesh are mixed with the exact point location of the bounded curve to obtain requested conditions.

Variational subdivisions: Some geometric modeling applications measure the smoothness of surfaces on physically based energy functionals.
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 smoothness. The variational schemes try to minimize the energy function which controls surface quality. For more details on this topic see [Kob96, Kob96b].

3.3.2 Detailed Subdivision Schemes

Tu by som detailne opisal par schem (najma Loopovu, kedze to pouzivam v softe a potom triangularne schemy (butterfly, $\sqrt{(3)}$) a mozno doo-sabina - tieto su implementovatelne s opisanym algoritmom.

Chapter 4

Optimized Subdivision Surface Displaying

4.1 Preliminaries

The aim of this chapter is to present the input we are going to work with, along with its properties and requirements. We will also introduce the basic geometrical tools needed for the description of the algorithm.

4.2 Input Data and Constraints

mesh M - For the basic algorithm we must ensure that all of the faces are triangular.

Subdivision algorithm (further denoted as A) - This can be any triangular scheme - although the proposed algorithm will differ in the case of vertex and face split subdivision schemes, and parts the offline process will have scheme-specific algorithms. The family of face split algorithms (Loop, modified butterfly) is more intuitive to handle with the proposed method - therefore for the first demonstration I will use Loop's method.

Level of subdivision (further denoted as l) - An integer greater or equal to 1 - (level 0 represents the initial mesh)

Distance from the viewer (further denoted as d) - a real number representing the distance of the viewer from the mesh in the virtual world. This is optional. It plays a great role in the case we want to introduce some LOD to the method.

4.3 Basic Definitions

It was mentioned in the earlier chapters that it is crucial for the algorithm to know whether the mesh is "suitable" for the proposed method. To achieve this we will define a set of property functions for analysis of the different properties of the triangles that make up the mesh M. Besides the suitability of the mesh they are the basic building stones of the algorithm itself as well.

4.3.1 Basic Geometric Formulas

We will denote a triangle by three points: $A, B, C \in R^3$. Further, we will denote $|B - A| = c; |C - B| = a; |A - C| = b$.

We will recall certain basic geometry functions that are essential for the rest of the work.

- Calculating the *inner angles* of the triangle: using dot product:

$$< B - A, C - B >= |B - A|.|C - B|. \cos\beta,$$

$$\beta = \arccos \frac{< B - A, C - B >}{|B - A|.|C - B|}$$

  , where β is the angle between the vectors $B-A$ and $B-C$ Analogously we can calculate the other two angles of the triangle.

- Calculating the *radius of the in-circle* of the triangle, further denoted as r: using the formula

$$r = \frac{1}{2}\sqrt{\frac{(b + c - a)(c + a - b)(a + b - c)}{a + b + c}}$$

- The *radius of the circum-circle* of the triangle, further denoted as R is described by the formula

$$R = \frac{abc}{\sqrt{(a + b + c)(b + c - a)(c + a - b)(a + b - c)}}$$

- And finally, using either of the above described formulas, the *area* of a triangle, denoted as S is given by the formula

$$S = \frac{abc}{4R} = rs$$

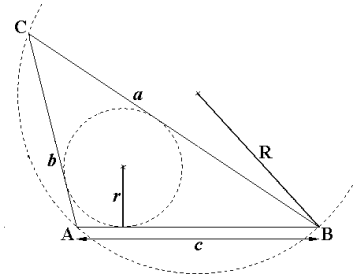

Figure 4.1: The circumscribed an inscrbed circles of a triangle

4.3.2 Triangle Characteristics

In this chapter we will define a set of functions working on a single triangle, or in some cases pair of triangles. They will be used later for:

- determining different properties of the mesh; most importantly suitability of the mesh
- partitioning the mesh into disjunct sets of triangles
- Miscellaneous auxiliary functions (for example sorting of the polygons in the pre-processing phase)

The testing functions are defined as boolean functions (binary if describing some property of a single triangle and ternary if working on two triangles) based on the result of a corresponding geometrical function.

1. ε-angle similarity based on geometrical function: Sum of the squares of the differences of angles. This property describes how similar are the two triangles compared to each other - for $\varepsilon = 0$ we say that the two triangles are similar (in the classical geometrical sense of similarity). The bigger is the value of ε, the less similar are the two triangles compared to each other.

$$f_1(\Delta_1, \Delta_2) = (\alpha_1 - \alpha_2)^2 + (\beta_1 - \beta_2)^2 + (\gamma_1 - \gamma_2)^2 \tag{4.1}$$

$$F_1(\Delta_1, \Delta_2, \varepsilon) = \begin{cases} \textbf{true} \text{ for } f_1(\Delta_1, \Delta_2) \leq \varepsilon, \text{ we say that the two triangles} \\ \qquad \text{are } \varepsilon\text{-similar} \\ \textbf{false} \text{ for } f_1(\Delta_1, \Delta_2) > \varepsilon \end{cases} \tag{4.2}$$

The next three property functions are describing how 'big' are the two triangles compared to each other in the geometrical sense.

2. ε-side similarity based on geometrical function: Sum of the squares of the difference of length of the sides. We assume that the sides of the triangle are sorted based on the side length from the shortest to the longest one. For $\varepsilon = 0$ the two triangles are congruent.

$$f_2(\Delta_1, \Delta_2) = (a_1 - a_2)^2 + (b_1 - b_2)^2 + (c_1 - c_2)^2 \tag{4.3}$$

$$F_2(\Delta_1, \Delta_2, \varepsilon) = \begin{cases} \textbf{true} \text{ for } f_2(\Delta_1, \Delta_2) \leq \varepsilon, \text{ we say that the two triangles} \\ \qquad \text{are } \varepsilon\text{-side similar} \\ \textbf{false} \text{ for } f_2(\Delta_1, \Delta_2) > \varepsilon \end{cases} \tag{4.4}$$

3. ε-area- similarity based on geometrical function: Difference of the area of two triangles

$$f_3(\Delta_1, \Delta_2) = (S(\Delta_1) - S(\Delta_2))^2 \tag{4.5}$$

$$F_3(\Delta_1, \Delta_2, \varepsilon) = \begin{cases} \textbf{true} \text{ for } f_3(\Delta_1, \Delta_2) \leq \varepsilon, \text{ we say that the two triangles} \\ \qquad \text{are } \varepsilon\text{-area}^{-} \text{ similar} \\ \textbf{false} \text{ for } f_3(\Delta_1, \Delta_2) > \varepsilon \end{cases} \tag{4.6}$$

4. ε-area/ similarity based on geometrical function: Ratio of the area of two triangles:

$$f_4(\Delta_1, \Delta_2) = (S(\Delta_1)/S(\Delta_2) - 1)^2 \tag{4.7}$$

$$F_4(\Delta_1, \Delta_2, \varepsilon) = \begin{cases} \textbf{true} \text{ for } f_4(\Delta_1, \Delta_2) \leq \varepsilon, \text{ we say that the two triangles} \\ \qquad \text{are } \varepsilon\text{-area}^{/} \text{ similar} \\ \textbf{false} \text{ for } f_4(\Delta_1, \Delta_2) > \varepsilon \end{cases} \tag{4.8}$$

5. Further we will define an auxiliary function, used in the definition of the next geometry function. It describes the 'equilateralness' of a triangle:

$$f_5(\Delta) = (r(\Delta)/R(\Delta) - 1)^2 \tag{4.9}$$

The function is used later to sort the triangles of the mesh.

6. Finally a function that yields 0 if the two triangles are congruent in the sense of equilaterality:

$$f_6(\Delta_1, \Delta_2) = (f_5(\Delta_1) - f_5(\Delta_2))^2 \tag{4.10}$$

$$F_6(\Delta_1, \Delta_2, \varepsilon) = \begin{cases} \textbf{true} \text{ for } f_6(\Delta_1, \Delta_2) \leq \varepsilon \\ \textbf{false} \text{ for } f_6(\Delta_1, \Delta_2) > \varepsilon \end{cases} \tag{4.11}$$

To move further in the description of the algorithm, we are going to define a weighted function of the above geometric functions:

$$f(\Delta_1, \Delta_2, k_1, \ldots, k_6) = k_1 f_1(\Delta_1, \Delta_2) + \cdots + k_6 f_6(\Delta_1, \Delta_2) \tag{4.12}$$

$$k_i \in R; k_i \geq 0; 1 \leq i \leq 6, i \neq 5; \sum_{i=1; i \neq 5}^{6} k_i = 1$$

$$F(\Delta_1, \Delta_2, k_1, \ldots, k_6, \varepsilon) = \begin{cases} \textbf{true} \text{ for } f(\Delta_1, \Delta_2, k_1, \ldots, k_6) \leq \varepsilon) \text{ we say that} \\ \qquad \text{the two triangles are } \varepsilon - \textit{similar} \\ \textbf{false} \text{ for } f(\Delta_1, \Delta_2, k_1, \ldots, k_6) > \varepsilon \end{cases} \tag{4.13}$$

The above defined property function (13) is a highly flexible geometric tool for analyzing different qualities of triangles sets (e.g. a mesh, as in our case). The variety of functions which define the final function gives us large flexibility through the modification of the parameters.

4.4 Intuitive Description of Algorithm

Fast(er) rendering of subdivision surfaces is still an interesting question in modeling and computer graphics. To be more formal, we want to propose a method for speeding up the rendering of the mesh that is in distance d from the viewer achieved by applying subdivision method A to the mesh M, l levels deep.

Let us have a look at the pseudocode of the classical algorithm. The pseudocode presented here is very rough and high level but it captures the main drawback of the method compared to the one proposed.

```
void subdivideMesh (Mesh M, Algorithm A, int l) {
  1.  for every triangle T in M do
  2.    while (reached level l)
  3.      subdivideTriangle(T, A);
```

}

The reader may here notified that in the above algorithm we have not used the variable d (the distance from the viewer) because the classical algorithms does not use any LOD directly. They calculate the new mesh and LOD modifications are applied in the rendering pipeline (if needed).

The basic idea of the proposed method is altering line 1. Although the subdivision methods are in most of the cases very simple, the number of their required application is the factor that raises the time cost: it grows exponentially because of the recursive nature of the subdivision rules. Therefore we could reduce the overall rendering process by leaving some triangles unsubdivided, but calculating them from other, already subdivided ones. It is the aim of the following chapters to describe this algorithm in details, along with its analysis of effectiveness on different types of meshes (also presenting The skeleton of a general framework for arbitrary (manifold) meshes effectiveness analysis) and show some practical results of the method.

4.5 Preprocessing the Input

The algorithm is divided into two parts: The pivot point of the whole method is the *preprocessing phase*, where the raw mesh data is loaded, turned into an internal representation called half-edge structure, processed with the proposed algorithm and then handed over to the *real time rendering phase*. The aim of this chapter is to describe the preprocessing phase, which can be very time consuming without proper optimization, careful design of object architecture and implementation.

4.6 Partitioning the Mesh

Definition 4.6.1 *We say that two triangles are* **strictly** $\varepsilon -$ **similar** *with respect to the subdivision scheme* **S** *if they are ε-similar and the topological conditions (usually ε-similarity of neighboring faces, valence of neighboring vertices - for details one has to refer to a concrete scheme) defined for* **S** *are met.*

Some explanation to figure 4.2: The color of the triangles being processed is red and green respectively. v_i's and w_i's are are the valences of the corresponding vertices. Two triangles Δ_1, Δ_2 are considered to be strictly ε-similar with resect to the Loop scheme if and only if $F(\Delta_1, \Delta_2) = true$, $F(\Delta_{1_{neighbor_i}}, \Delta_{2_{neighbor_i}}) = true$ and moreover $v_i = w_i$ for $i = 1, \ldots, 9$.

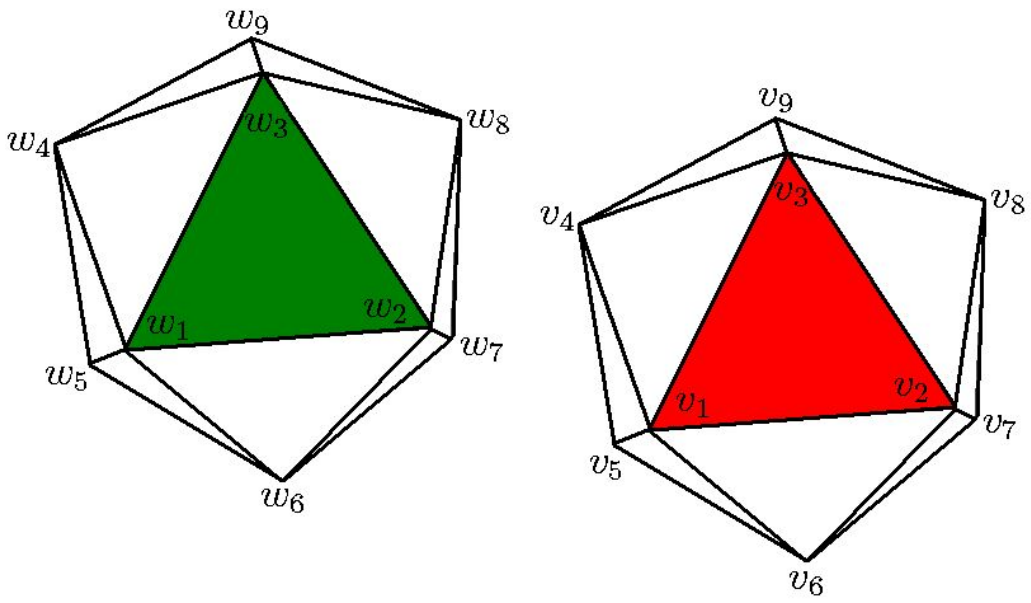

Figure 4.2: Topological conditions for the Loop scheme

$\Delta_{j_{neighbor_i}}$ denotes the i-th neighbor of the j-th triangle for $j = 1, 2$ and $i = 1, \ldots, 9$.

This simple fact is going to be the basis of the algorithm: If we can find two triangles that are strictly ε-similar we can subdivide any of them and map it to the other one. This holds for a set of triangles as well: If all the triangles in the set has the property described before, after choosing and subdividing one triangle the others can be calculated with the corresponding affine transformation between two triangles.

After turning the loaded mesh into a set of triangles, we are going to apply the function F, defined in section 4.3.2 to all pairs of triangles that can be created from the triangle set (not considering the order of triangles in the pair) setting ε=0 and the vector for the coefficients is $(1, 0, 0, 0, 0)$.

There are f subsets of triangles at the beginning of the algorithm: Every subset contains exactly one triangle and every triangle is member of exactly one subset. Applying the function F to a pair of triangles (considering the equality of the valences as well) is expressed with the following pseudocode:

List splitMesh(List triangleSets)
{
1. for *every pair*$(\Delta_1 \in T_1, \Delta_2 \in T_2; T_1, T_2 \in triangleSets; T_1 \neq T_2)$ do
2. if $F(\Delta_1, \Delta_2, 1, 0, 0, 0, 0, 0)$ then

{
3. $T_1 = T_1 \bigcup T_2$;
4. $triangleSets = triangleSets \setminus T_2$
}
5. return $triangleSets$;
}

The described process is executed for all possible pairs of triangles.After going through the described process, the original set of triangles is partitioned so that the following holds:

As the result of partitioning (by executing the procedure splitMesh()) we have divided the original triangle sets into k disjunctive subsets with the following properties:

$$\{T_1, \ldots, T_k\}, 1 \leq k \leq f; \bigcup_{1 \leq i \leq k} T_i = M \qquad (1)$$

Moreover, for every subset of triangles T_i:

$$\forall |T_i| \geq 2 : \forall \Delta_1, \Delta_2 \in T_i \colon F(\Delta_1, \Delta_2, \underbrace{1, 0, 0, 0, 0}, \underbrace{0}) \; is \; true \quad (2)$$

the vector $k_1, \ldots, k_6$ ε

Proof Before the start of the procedure splitMesh() we have arranged f triangles of the mesh so that every triangle belongs to exactly one set and every set contains exactly one triangle. Thus before the execution of splitMesh() the input list triangleSets is in the following form by definition:

$$triangleSets = \{T_1, \ldots, T_f\}; \forall T_i, T_j, i \neq j, T_i \cap T_j = \emptyset; \bigcup_{1 \leq i \leq f} T_i = M$$

If there are no $\varepsilon - similar$ triangles

The if statement in row 2. of the procedure splitMesh() is never *true*) rows 3,4 will be not executed and hence the output list will be equal to the input list, so both (1) (for $k = f$) and (2) (because $|T_i| = 1, 1 \leq i \leq f$) hold trivially.

If there are at least two similar triangles

1^o) After the first pair $\Delta_1 \in T_i, \Delta_2 \in T_j$ is found: $T_i = T_i \bigcup T_j$ and T_j is removed from the list $triangleSets$; (1) holds because

$$\bigcup_{1 \leq i \leq f} T_i = \bigcup_{1 \leq i \leq f; i \neq j} T_i \cup T_j$$

(2) holds because the only set with at least 2 elements is $T_i = \{\Delta_1, \Delta_2\}$ and we know that $F(\Delta_1, \Delta_2, \ldots) = true$
$k-1^o$) after k-1 pairs have been found:

$$triangleSets = \{T_1, \ldots, T_{f-k+1}\}, \forall T_i, T_j, i \neq j, T_i \cap T_j = \emptyset;$$

$$\bigcup_{1 \leq i \leq f-k+1} T_i = M$$

k^o) after k pairs have been found:

$$triangleSets = \{T_1, \ldots, T_{f-k}\}, \forall T_i, T_j, i \neq j, T_i \cap T_j = \emptyset;$$

$$\bigcup_{1 \leq i \leq f-k} T_i = M$$

We have to prove $k-1^o) \Rightarrow \ k^o)$ DOKONCIT TOTO..... Hadam to nebude problem ale teraz je 5:03 a mam este milion veci opravit...

Partitioning of the mesh is by far the most time consuming part of the process - quadratic complexity, depending on the number of faces.

4.7 Optimizing Triangle Sets

The concept of optimizing the triangle sets before further processing relies on one of the properties of the manifold mesh: Faces sharing a common vertex can be always ordered so that two subsequent faces always share an edge.

The idea is to find the biggest possible patches that are built up from similar triangles with the same topology - called *strictly similar patches* (see figure 4.3)

Finding such patches can significantly speed up the real time rendering process described in 4.7.2 Once two (or more) strictly similar patches are discovered, the following process takes place:

The triangles in the patches are ordered according to a previously chosen rule which has the following properties:

1. If P_1 and P_2 are two strictly similar patches with ordered triangles, then the triangle at the i-th place in P_1 is strictly similar with the triangle at the i-th place in P_2

2. In every patch with ordered triangles, the i-th triangle and the $(i+1)$-th triangles share an edge

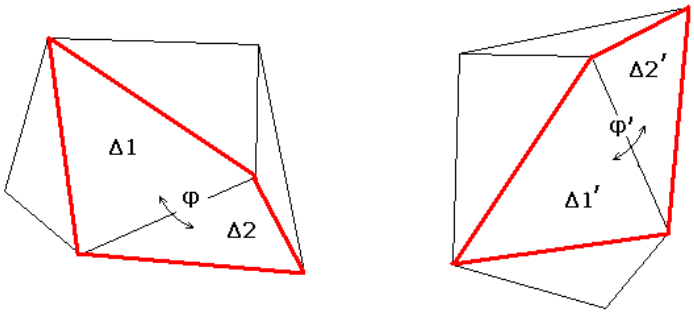

Figure 4.3: Strictly similar patches - Two strictly similar patches are marked with red - they must fulfill the following requirements: Δ_1 and Δ_1' must be strictly similar, Δ_2 and Δ_2' must be strictly similar, $\varphi = \varphi'$ angle between the planes containing Δ_1, Δ_2 and Δ_1', Δ_2' respectively

After ordering the triangles in the patches, do the following:

1. The first triangle of the first patch is subdivided and the first triangle of the second patch is calculated by transforming the subdivided triangle
2. The i-th triangle is calculated from the $(i-1)$-th one so that the existing points are not duplicated (The existing points are the common boundary of the i-th and $(i-1)$-th triangle.

This way all the shared edges are calculated only once.

4.7.1 Calculating Transformation Matrices

Once having the triangle subsets the calculation of matrices of affine transformations takes place - these matrices are describing the affine transformation that maps pairs of similar triangles onto each other. Later these matrices play an important role in the real time rendering process - they are used to calculate some triangles from the already subdivided ones.

There are some basic methods for achieving this: Either finding and mapping the normal vector of the first triangle to the second one, or (for both triangles) find a point that is not a part of the triangle (more precisely, it is not contained in the plane that is spanned by the three points of the triangle) - thus creating a tetrahedron - and mapping these tetrahedrons onto each other.

We will use the first one: Mapping the normal vector of the first triangle onto the normal vector of the other one. We are going to describe every

triangle with a matrix containing its affine coordinates and its normal vector, called the characteristic matrix of the triangles. See 4.4.

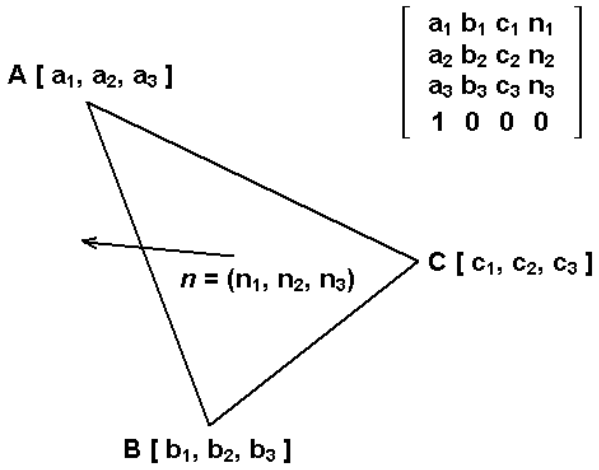

Figure 4.4: Matrix describing the position and orientation of a triangle

As the first step, we need to set up these matrices for every triangle that is in a subset with 2 or more triangles The 'first' triangle of the set that is chosen in the algorithm is called a $reference$ triangle. We us Gaussian elimination to gain the matrices of transformations.

These steps are accomplished fully off-line, thus not contributing into the time efficiency of the real time rendering. Still, for a bigger mesh the time of the processing is quite long, so after the mesh is processed once, it is stored in a file of a special format (capable of storing the subsets of triangles and the matrices etc.) so before the real time rendering this file is just loaded and ready for processing instead of going through this process in case of every rendering.

4.7.2 Real Time Rendering Phase

The *real time rendering* process is the final result of the algorithm. In this phase the original mesh is rendered after l steps of subdivision - hopefully faster than with the classical algorithm, thanks to the data calculated in the preprocessing part.

When the real time rendering begins, data are loaded from an external file. Then the following algorithm (described by it's pseudocode) is executed:

List renderTriangleSet(List triangleSets)

{
1. for *every* $T_i \in triangleSets$ do
2. $\Delta_{i_1} := T_i[1]$;
3. subdivide Δ_{i_1} *using algorithm A*
4. for *every* $T_i \in triangleSets$ with $|T_i| \geq 2$ do
{
5. $\Delta_{i_1} := T_i[1]$;
6. for *every* $\Delta_{i_j}, 2 \leq i_j \leq |T_i|$ do
7. calculate Δ_{i_j} from Δ_{i_1} using $M_{i_1 \to i_j}$
}

After having subdivided and transformed the triangles, there is just one more thing left to do: pass it to the rendering pipeline and rasterize the subdivided mesh.

4.8 Results

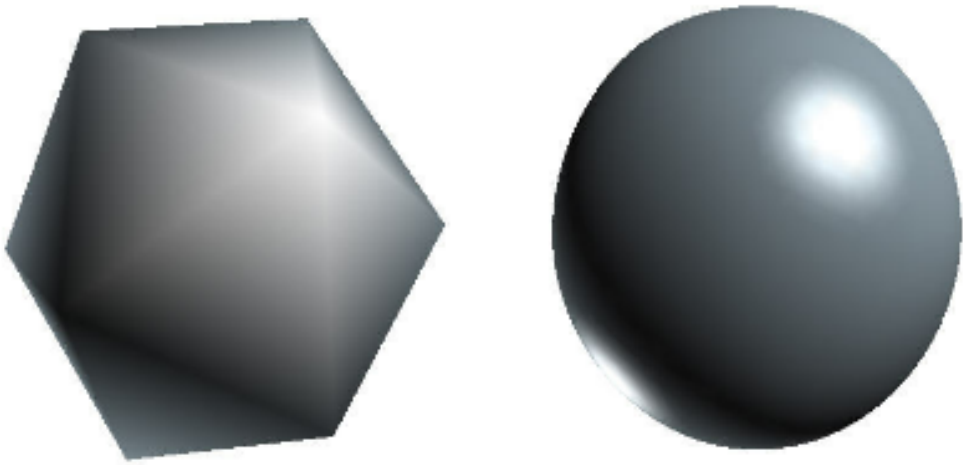

Figure 4.5: On the left: picture of an icosahedron in the initial form (20 faces). On the right, the icosahedron is subdivided 5 times (22345)

4.9 Future Work

There are rounding errors introduced during the transformation of the vertices by floating point numbers, causing problems by the computation of the

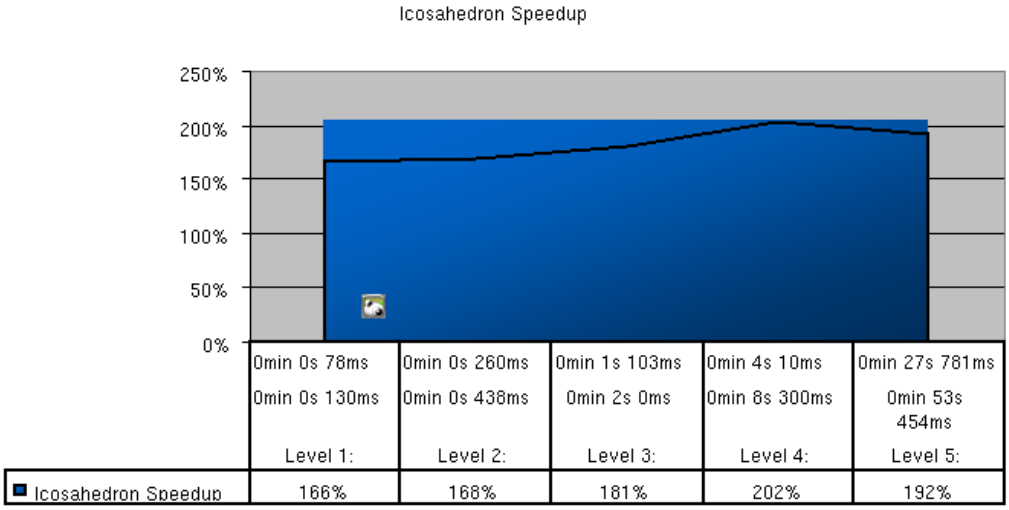

Figure 4.6: Time needed for processing subsequent levels of subdivision steps. The best result is achieved by subdividing the 4 th level: the speed is more than doubled

normal vector (see figure 4.10). We need to approximate the required precision based on the distance from the viewer, so that the numerical errors are not seen.

Solution of this problem is the merging of these vertices into one vertex. This algorithm needs to be proposed so that the time needed for processing is not increased drastically.

Further improvements include implementation of more subdivision schemes (Doo-Sabin, $\sqrt{3}$, butterfly and more robust treating of special objects (boundaries, creases).

Tieto posledne 2 sekcie su vo velmi rannom stadiu - he to skor ukazka toho ze co tu bude. ad1 som nemal cas ich poriadne napisat, ad2 som nemal cas na poriadne testovanie, teda dufam ze pocas nasledujucich dni sa tieto kapitoly vyrazne vylepsia co sa tyka kvantity aj kvality. (su este zaujimave vysledky na vseliakych meshoch, a mam tolko napadov ze future work by som mohol mat 5 stran bez toho aby tam boli haluze... aspon si ja myslim...

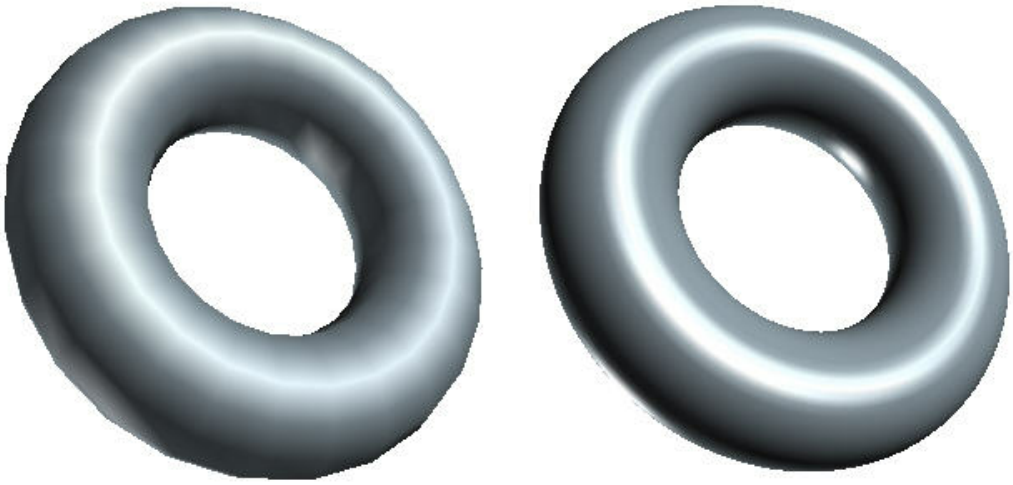

Figure 4.7: On the left: picture of a torus in the initial form (437 faces). On the right, the torus is subdivided 2 times (16393 faces)

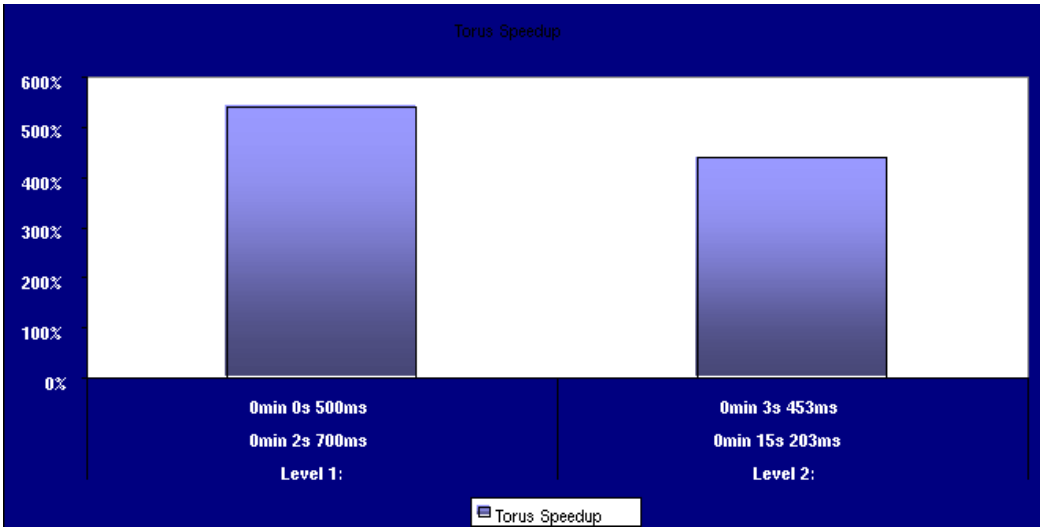

Figure 4.8: Time needed for processing subsequent levels of subdivision steps. The speedup is smaller on the second level, but still 4.4* faster than the classic algorithm

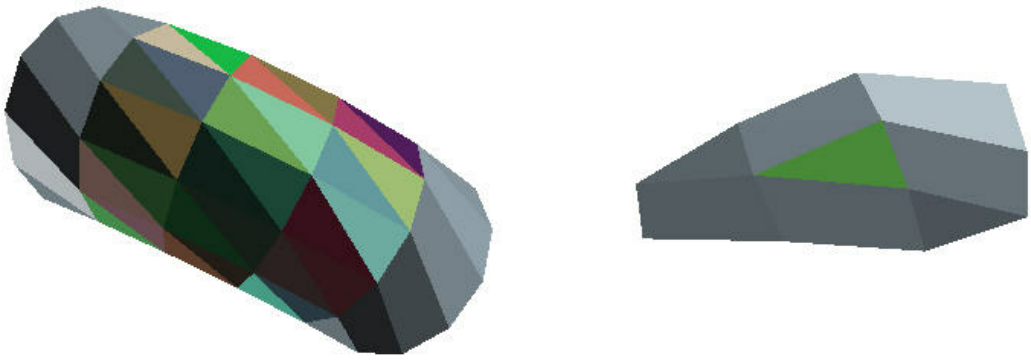

Figure 4.9: On the left: set of triangles of a torus needed to reproduce the mesh - result gained by using the vector (1,0,0,0,0) On the right: the same torus can be reproduced from one single triangle - using the vector (0,0,1,0,0)

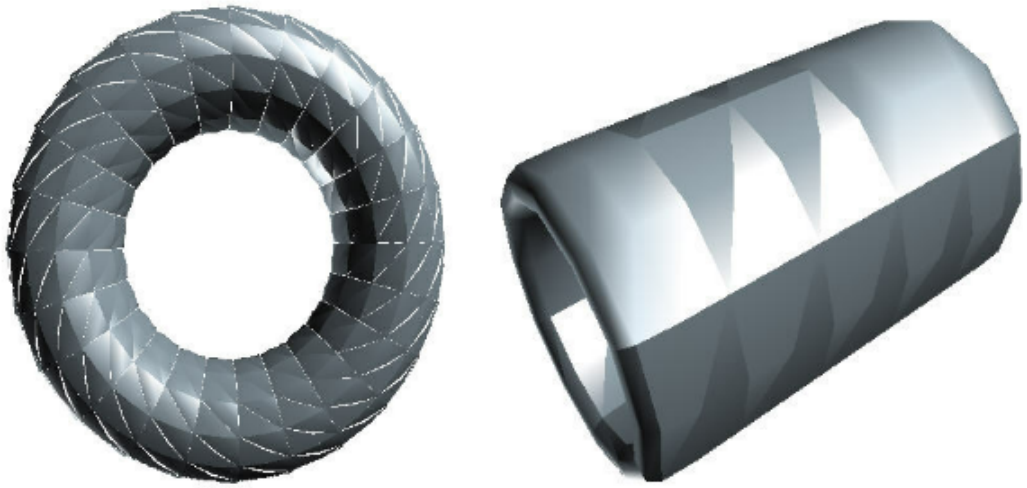

Figure 4.10: Rounding errors

Bibliography

[1] Akelman, E., Srinivasan, V. Honeycomb subdivision. *Texas A&M University* (2002)

[2] Biermann H., Levin A., and Zorin, D. Piecewise Smooth Subdivision Surfaces with Normal Control. *Tech. rep, Courant Institute, New York University, January* (1999)

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

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

[5] Kobbelt L., Tesse H., Prautzsch H., and Schweizerhof K. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Computer Graphics Forum, 15:409-420* (1996)

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

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

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

[9] Reif, U. A degree estimate for polynomial subdivision surfaces of higher regularity. *Proc. Amer. Math. Soc.* (1996), 124:2167-2714

[10] Schweitzer, Jean E. Analysis and Application of Subdivision Surfaces. *PhD thesis, University of Washington, Washington* (1995)

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

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

[13] 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