Comenius University in Bratislava

Faculty of Mathematics, Physics and Informatics

Refining Procedures on Mesh via Algebraic Fitting

Master's Thesis

Tibor Stanko

Study field: **9.1.1 Mathematics**

Study programme: **Computer Graphics & Geometry**

Supervisor: **doc. RNDr. Pavel Chalmovianský, PhD.**

Code: **a34d7381-217f-4edf-86d6-f090378d4b13**

Bratislava 2014

page intentionally left blank

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Tibor Stanko
Study programme:	Computer Graphics and Geometry (Single degree study, master II. deg., full time form)
Field of Study:	9.1.1. Mathematics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Refining procedures on mesh via algebraic fitting

Aim: We consider a mesh with/without normals at vertices. Each triangle is associated with a subset of neighboring point/normals. A procedure fitting to the data associate to each triangle a low degree algebraic surface is constructed. A pont with/without normal is picked out of the surface so that the mesh is better with respect to a chosen objective function. We study the existing methods and experiment with various setups.

Literature: Unstructured Mesh Generation, Jonathan Richard Shewchuk, University of California, Berkeley
Hoschek, Lasser: Fundamentals of CAGD, AK Peters, 1992

Keywords: mesh generation, refinement, fitting, algebraic surface

Supervisor:	doc. RNDr. Pavel Chalmovianský, PhD.
Department:	FMFI.KAGDM - Department of Algebra, Geometry and Didactics of Mathematics
Head of department:	prof. RNDr. Pavol Zlatoš, PhD.
Assigned:	03.07.2013

Approved: 25.06.2013 — doc. RNDr. Andrej Ferko, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgements

> "The most important thing we learn at school is the fact that the most important things can't be learned at school."
>
> Haruki Murakami in *What I Talk About When I Talk About Running*

This thesis would not be written without the help of certain important people.

First of all, I would like to thank my supervisor Pavel Chalmovianský for his invaluable guidance, his ideas, answers and comments. In the past year, we spent countless hours talking (not only) about my work. I was full of thoughts every time I was leaving his office.

Many thanks go to Andrej Ferko – for being an inspiration to me, for sharing his rich knowledge and experience, for teaching us to respect people who deserve respect and for pointing me in the right direction when I needed it.

I am eternally grateful to Lujza, my parents, family and friends for being the support they are.

The contents of this thesis were created using LaTeX, Asymptote, Texmaker, Gimp, Inkscape, MeshLab, OpenMesh, Armadillo and LibreOffice. All figures were created by me unless stated otherwise in the figure caption. The meshes used to test the presented algorithm were kindly provided by Moravské zemské muzeum in Brno and EDICO SK, Inc (Dolní Věstonice statuettes) and the Stanford University Computer Graphics Laboratory (Stanford bunny).

I hereby declare the work presented in this thesis is original and has not been submitted elsewhere in any form for another degree or diploma. The presented results come from my own research, pursued with the help of my supervisor and the referenced literature.

Tibor Stanko
Bratislava, May 2014

Abstract

Tibor Stanko. *Refining Procedures on Mesh via Algebraic Fitting.* Master's thesis, Comenius University, Bratislava. May 2014.

In this thesis, we introduce a nonlinear interpolating scheme for the refinement of a triangular mesh. Each triangle is associated with a small set of neighbouring points and normals. A low degree algebraic surface (quadric) is fitted to this set with respect to the well-chosen chosen objective function. The new vertex is taken from the computed quadric. Such a setup allows modelling with normals, hence overcoming the limitations of linear schemes. An application of the proposed method to mesh compression is shown, including certain geometric analysis of the result. If the parameters (weights) are chosen properly, the scheme is able to reconstruct a close approximation of a quadratic surface from a coarse input mesh.

keywords: triangular mesh, mesh refinement, nonlinear scheme, subdivision surfaces, compression

Abstrakt

Tibor Stanko. *Refining Procedures on Mesh via Algebraic Fitting.* Diplomová práca, Univerzita Komenského, Bratislava. Máj 2014.

V tejto práci je prezentovaná nelineárna interpolačná schéma na zjemňovanie trojuholníkovej siete. Každý trojuholník je asociovaný s malou množinou susedných bodov a normál. Skonštruujeme kvadratickú plochu, ktorá najlepšie aproximuje túto množinu s ohľadom na vybranú dobre definovanú optimalizačnú funkciu. Nový vrchol je vybraný z vypočítanej kvadriky. Týmto spôsobom vieme limitnú plochu modelovať s využitím normál, čím prekonávame odmedzenia lineárnych schém. Výslednú metódu geometricky analyzujeme a využívame na kompresiu trojuholníkovej siete s veľkým množstvom vrcholov. Pri správnej voľbe parametrov (váh) je schéma schopná zrekonštruovať blízku aproximáciu kvadratickej plochy z dostatočne hustej vstupnej siete.

kľúčové slová: trojuholníková sieť, zjemňovanie siete, nelineárna schéma, rafinačné plochy, kompresia

Contents

List of Figures

List of Tables

List of Algorithms & Source Code

page intentionally left blank

Chapter 1

Introduction

The problem of efficient and accurate geometry modelling of solids has been present in computer graphics from the very beginning. A new approach for boundary representation of three-dimensional objects has emerged in the late 1970s. What became known as *subdivision surfaces* is now widely used in domains such as CAGD, geometry modelling for animation, level-of-detail modelling, multiresolution analysis.

While linear refinement methods for surface subdivision have been closely studied in the past decades, nonlinear methods have received little attention. Linear schemes work well in cases when the positions of vertices are known. The difficulties arise when we try to make use of data coming from derivatives, such as tangent or curvature. Nonlinear schemes seem to be the right mechanism to bridge this gap.

A new nonlinear refinement algorithm for surfaces is presented in this work. Our scheme operates on triangular meshes and interpolates input data. Even though the scheme is nonlinear, it only requires solving a well-formed system of linear equations for each triangle of the subdivided mesh.

We begin by reviewing the common means for surface representation in chapter 2. This chapter also introduces the definition of *mesh*, which is one of the central notions of the thesis.

Chapter 3 gives a more detailed introduction to the world of subdivision surfaces. We give a brief motivation for modelling with surfaces defined as limit, as well as the general classification of such surfaces. The main part of this chapter consists of an overview of linear and nonlinear methods for surface refinement.

The proposed nonlinear refinement scheme is derived in chapter 4. The algorithm is based on fitting a quadratic surface to the local (vertex and normal) neighbourhood of each triangle. This process corresponds to the minimization of the objective function, which leads to the system of linear equations. To achieve the best results, the weights are assigned to the vertices and to the normal vectors.

The implementation of our method is briefly described in chapter 5. The application was programmed on top of the the *Subdivider* tool from the library *OpenMesh*. The

commented C++ source code of the application is attached to the thesis.

In chapter 6, we experiment with various sets of weights and analyse the influence of the normal vectors on the limit surface generated by our method. We also show how the resulting method can be used to compress triangular mesh obtained from laser scanning. Such a mesh usually consists of large datasets, typically $\sim 10^5 - 10^6$ vertices. Applying our scheme on properly chosen decimation of the input mesh, we are able to reconstruct scanned data very accurately. The proposed scheme can also be used for the reconstruction of quadratic surfaces from a coarse approximating mesh. We provide a demonstration by reconstructing the sphere, the hyperbolic paraboloid, the elliptic paraboloid and the cylinder.

The thesis is concluded by chapter 7. Several ideas for the future work and improvements of the scheme are given.

Chapter 2

Surface Representations

To design dynamic and robust algorithms in computer graphics, we need to be able to represent three-dimensional objects efficiently. Typical method for representing an object is to specify the boundary surface of an object. Such representation is often called *boundary representation* or simply *B-rep*.

In this chapter, we survey the basic surface representation techniques. We also introduce important notions that will be used throughout next chapters, such as a polygonal mesh, a boundary edge, an extraordinary vertex. For more details on the surface representation, see also [BPK+07], [CF01] or [FHK02].

Standard definition of a surface in the context of computer graphics is that of *an orientable continuous two-dimensional manifold embedded in* $\mathbb{R}^3$ [BPK+07]. Objects with complicated shape are usually difficult to represent using only one function. For that reason, the function domain is split into smaller parts, and the representation is constructed over each sample separately. This *piecewise definition* allows for local approximation of the desired surface. In terms of global representation, one needs to ensure the overall surface is $\mathcal{C}^q$ continuous up to some specified degree of continuity q.

2.1 Parametric representation

A parametric surface is defined as a map of some two-dimensional parameter domain $\mathcal{D} \subset \mathbb{R}^2$ to three-dimensional space,

$$\mathbf{f} : \mathcal{D} \to \mathcal{S} \subset \mathbb{R}^3, \quad (u, v) \mapsto \mathbf{f}\,(u, v) = \begin{pmatrix} f_x(u, v) \\ f_y(u, v) \\ f_z(u, v) \end{pmatrix} \in \mathcal{S}. \tag{2.1}$$

This form of representation has been widely used in computer-aided geometric design (CAGD) for various reasons. Parametric surfaces are easy to sample and visualise. Another asset of the parametric representation is that it allows for transformation

of some problems in the three-dimensional space to problems in the two-dimensional space of the parameter domain. For instance, consider the search for the *geodesic γ-neighbourhood*

$$\mathcal{N}_g(\mathbf{p}_0, \gamma) = \{\mathbf{p} \in \mathcal{S} \mid d_g(\mathbf{p}, \mathbf{p}_0) < \gamma\} \tag{2.2}$$

of point $\mathbf{p}_0 = \mathbf{f}(u_0, v_0) \in \mathcal{S}$, where $d_g(\mathbf{p}, \mathbf{p}_0)$ is the geodesic distance on the surface $\mathcal{S}$ – the shortest distance between the points $\mathbf{p}, \mathbf{p}_0$ along the surface $\mathcal{S}$. The problem of finding such geodesic neighbourhood can be simplified by considering only neighbourhood points in the parameter domain $\mathcal{D}$.

2.1.1 Spline surfaces

A spline surface consists of patches, defined piecewise by *control points* and *basis functions*. The control points and basis functions are chosen in a way that allows the whole spline surface to be $\mathcal{C}^q$ continuous for some $q \in \mathbb{N}_0$.

Tensor-product spline surfaces are defined over the piecewise rectangular domain $\mathcal{D}$. Without the loss of generality, we assume $\mathcal{D} = [0,1] \times [0,1]$ as any rectangle $\mathcal{R} = [a,b] \times [c,d] \subset \mathbb{R}^2$ can easily be transformed to the unit square. As an example, consider a Bézier patch of degree $(m,n) \in \mathbb{N} \times \mathbb{N}$

$$\mathbf{b}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(u)\, B_j^n(v)\, \mathbf{p}_{ij}, \tag{2.3}$$

where $\mathbf{p}_{ij} \in \mathbb{R}^3$ are the control points, $B_k^d(t) = \binom{d}{k} t^k (1-t)^{1-k}$ is the univariate Bernstein polynomial of degree d. Bézier splines can be extended to rational Bézier splines and generalized to locally supported *non-uniform rational basis splines* (NURBS).

Triangular spline surfaces are defined over the triangular domain $\mathcal{T} = \mathbf{t}_0\mathbf{t}_1\mathbf{t}_2 \subset \mathbb{R}^2$. As an example, consider a Bézier patch once again, this time triangular. Every point $(u,v) \in \mathcal{T}$ from the domain is associated with the barycentric coordinates (α, β, γ) with respect to the triangle $\mathcal{T}$

$$(u,v) = \alpha\, \mathbf{t}_0 + \beta\, \mathbf{t}_1 + \gamma\, \mathbf{t}_2, \quad \alpha + \beta + \gamma = 1. \tag{2.4}$$

The triangular Bézier patch of degree $n \in \mathbb{N}$ over triangle $\mathcal{T}$ is defined as

$$\mathbf{b}(u,v) = \sum_{i+j+k=n} B_{ijk}^n(\alpha, \beta, \gamma)\, \mathbf{p}_{ijk}, \tag{2.5}$$

where $\mathbf{p}_{ijk} \in \mathbb{R}^3$ are the control points, $B_{ijk}^n(\alpha, \beta, \gamma) = \dfrac{n!}{i!j!k!} \alpha^i \beta^j \gamma^k$ are the bivariate Bernstein polynomials of degree n. As in the case of tensor-product splines, triangular Bézier splines can be extended to rational triangular splines and generalized to triangular B-splines. For the details on the topic, see for example [HLS93] or [Far96].

Spline surfaces are widely used in traditional CAGD modelling, although they have their disadvantages. Objects with complex topology modelled with smooth splines require many smoothly blended patches and are difficult to construct. Moreover, it is complicated to perform additional geometric modifications.

2.1.2 Polygonal meshes

A polygonal mesh $\mathcal{M}$ is defined as a pair

$$\mathcal{M} = (\mathbf{V}, \mathcal{P}), \quad \mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}, \quad \mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}, \tag{2.6}$$

where $\mathbf{V}$ is a collection of vertices $\mathbf{v}_i = (x_i, y_i, z_i)^\top \in \mathbb{R}^3$ and $\mathcal{P}$ is a collection of polygons. Each polygon $\mathcal{P}_j \in \mathcal{P}$ is specified as an ordered list of vertices $\mathbf{v}_i$. The set $\mathbf{V}$ is called the *geometric component* of the mesh $\mathcal{M}$, while the set $\mathcal{P}$ is called the *topological component* of the mesh $\mathcal{M}$. In case all of the polygons in $\mathcal{P}$ are triangles, we refer to a *triangular* or a *triangle mesh*. Similarly, we refer to a *quadrilateral mesh*, a *hexagonal mesh*, etc.

Another way of thinking about the topological component of the mesh $\mathcal{M}$ is to consider a graph structure with a set of vertices

$$\mathcal{V} = \{v_1, \ldots, v_V\}, \tag{2.7}$$

and a set of edges

$$\mathcal{E} = \{e_1, \ldots, e_E\} \subset \mathcal{V} \times \mathcal{V}. \tag{2.8}$$

Formally, a mesh must be a cell complex [She12], hence further conditions are placed on $\mathcal{E}$. The collapsed edges (loops) are forbidden, $[v_i v_i] \notin \mathcal{E}$, and the order of the end vertices of the edge is not relevant, $[v_i v_j] = [v_j v_i]$. Alternatively, one can specify the mesh connectivity with a set of faces instead of a set of edges. For example, for the triangle mesh $\mathcal{M}$,

$$\mathcal{F} = \{f_1, \ldots, f_F\} \subset \mathcal{V} \times \mathcal{V} \times \mathcal{V} \tag{2.9}$$

is a set of triangles of $\mathcal{M}$. Again, the set $\mathcal{F}$ must conform to some additional conditions. If $[v_i v_j v_k] \in \mathcal{F}$, then i, j, k differ, and all the permutations of $[v_i v_j v_k]$ represent the same triangle face. The geometric component of $\mathcal{M}$ is specified by associating a position in three-dimensional space with every vertex, thus embedding the set $\mathcal{V}$ in $\mathbb{R}^3$,

$$E : \mathcal{V} \mapsto \mathbb{R}^3, \quad \mathbf{v}_i = \mathit{position}\,(v_i) = (x_i, y_i, z_i) \in \mathbb{R}^3. \tag{2.10}$$

Such embedding also determines the embeddings of the sets $\mathcal{E}$ and $\mathcal{F}$ in $\mathbb{R}^3$ together with the induced metric. When referring to sets of vertices, edges and faces of the mesh $\mathcal{M}$, we use the notation $\mathcal{V}(\mathcal{M}), \mathcal{E}(\mathcal{M})$ and $\mathcal{F}(\mathcal{M})$.

The piecewise parametric form of a triangle mesh represents each triangle $T = \mathbf{v}_i\mathbf{v}_j\mathbf{v}_k \in \mathcal{F}(\mathcal{M})$ as a convex combination of its vertices,

$$T(u,v) = u\,\mathbf{v}_i + v\,\mathbf{v}_j + (1-u-v)\,\mathbf{v}_k, \quad (u,v) \in [0,1]^2\,. \tag{2.11}$$

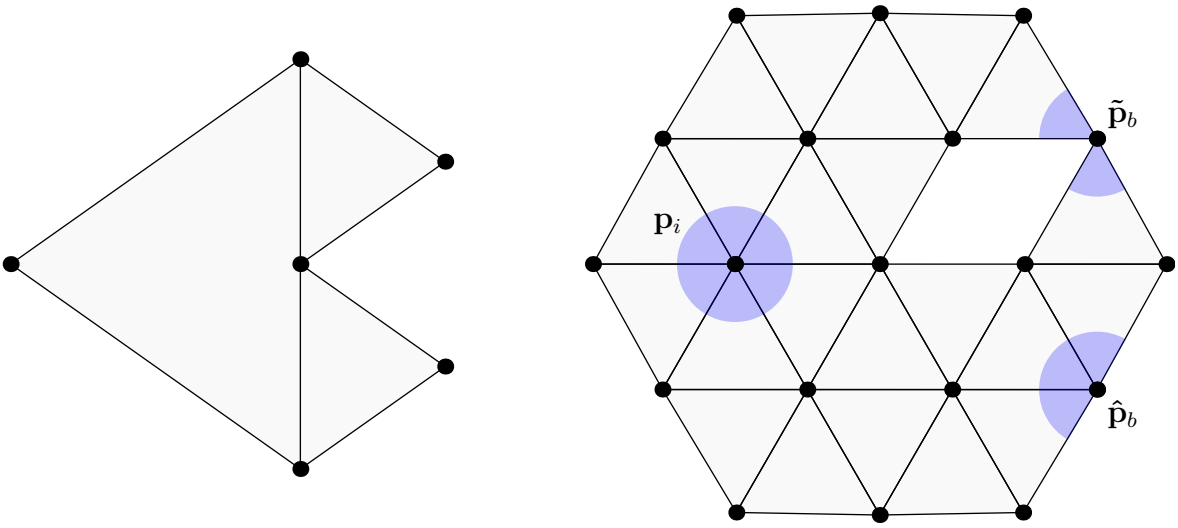


Figure 2.1: (*Left*) Example of a non-manifold mesh, in which one edge is shared by three triangles. (*Right*) The neighbourhood of a vertex (or any point) in a manifold mesh is topologically equivalent to a disc or a (piecewise) half-disc.

Every edge in *2-manifold mesh*, or simply manifold mesh, is shared by exactly one face (*boundary edge*) or by exactly two faces (*non-boundary* or *interior edge*). If an arbitrary point (vertex) of the mesh $\mathcal{M}$ is lying on some boundary edge, we refer to it as a *boundary point* (*vertex*). Otherwise, we refer to an *interior point* (*vertex*). A mesh with only non-boundary edges is *closed*, otherwise we refer to a *mesh with boundary*.

Examples of various types of vertices are shown in figure 2.1. The neighbourhood of an interior point $\mathbf{p}_i$ is topologically equivalent to a disk. The neighbourhood of a boundary point $\mathbf{p}_b$ is either topologically equivalent to a half-disk ($\hat{\mathbf{p}}_b$ on the figure), or, if the neighbourhood without $\mathbf{p}_b$ consists of more than one component, each component is equivalent to a half-disk ($\tilde{\mathbf{p}}_b$ on the figure).

The *valence* or *valency* of a vertex is defined as the number of edges adjacent to this vertex. When dealing with triangular or quadrilateral meshes, two types of vertices can be recognized. Consider the regular polygonal tessellation of the half-plane

$$\left\{(x,y) \in \mathbb{R}^2 : y \geq 0\right\}, \tag{2.12}$$

see figure 2.2. All interior (boundary) vertices in a regular tessellation have the same valence. We call such valence the *regular valence* of interior (boundary) vertex in triangular (quadrilateral) mesh. For the triangular tessellation, the regular valence of an interior (boundary) vertex is 6 (4). In case of the quadrilateral tessellation,

the regular valence of an interior (boundary) vertex is 4 (3). A *regular vertex* of the (triangular or quadrilateral) mesh $\mathcal{M}$ is a vertex, whose valence is regular. Otherwise, a vertex is *extraordinary*.

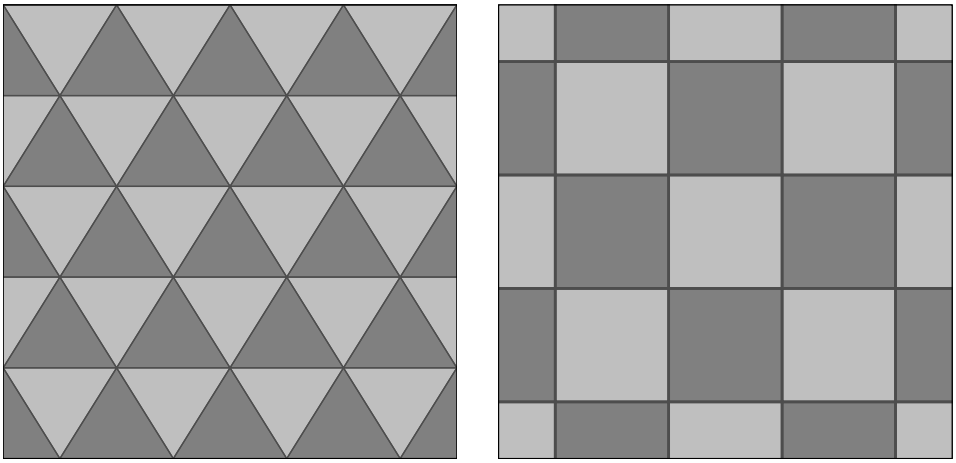


Figure 2.2: Regular triangular and quadrilateral tessellations of the plane.

2.1.3 Subdivision surfaces

Subdivision is a way of representing smooth shapes in computer [Sab10]. Subdivision surfaces can be thought of as a generalization of spline surfaces and are therefore a parametric representation. Like spline surfaces, they are controlled by a coarse control mesh, but unlike spline surfaces, they are able to represent arbitrary topology. The subdivision surface $\mathcal{S}$ is defined as a limit of successive refinement of an initial mesh $\mathcal{M}^0$,

$$\mathcal{S} = \lim_{k \to \infty} \mathcal{M}^k. \tag{2.13}$$

More detailed description of subdivision surfaces follows in chapter 3.

2.2 Implicit representation

An implicit surface $\mathcal{S}$ is defined as a non-empty zero-set of scalar-valued function $F : \mathbb{R}^3 \to \mathbb{R}$,

$$\mathcal{S} = \left\{ \mathbf{x} \in \mathbb{R}^3 \mid F(\mathbf{x}) = 0 \right\}. \tag{2.14}$$

Additionally, the points from the interior of the volume bounded by the surface $\mathcal{S}$ satisfy $F(\mathbf{x}) < 0$, while outside points satisfy $F(\mathbf{x}) > 0$.

Implicit surfaces are very convenient for giving answers to spatial queries, such as *Is the point inside, outside of the bounded volume, or on the surface $\mathcal{S}$?* With the surface $\mathcal{S}$ defined implicitly, this question simply reduces to determining the sign of the function F. On the other hand, some of the tasks that are easily performed with a

parametric representation become difficult when the surface is represented implicitly. For instance, sampling an implicit surface or finding the geodesic neighbourhood of a point are hard to accomplish.

Chapter 3

Subdivision Surfaces

Subdivision surfaces have come a long way since the first papers on subdivision modelling have been published side-by-side in the same volume of the *Computer-Aided Design* magazine, see [CC78], [DS78]. They now represent a powerful alternative to the traditional means of the geometric modelling, such as B-spline surfaces. In this chapter, we first overview the fundamentals of the subdivision modelling and the classification of subdivision schemes. We then review the principles of the linear subdivision and the nonlinear subdivision, including the description of the most common schemes.

3.1 On subdivision and refinement

Various perspectives for seeing subdivision surfaces emerged over the past few decades. Overview of these perspectives follows, inspired by Peters and Reif [PR08].

Smooth geometry

Subdivision surfaces can naturally be seen as a tool for modelling almost smooth (up to extraordinary points) geometric surfaces with arbitrary topology. This simple, intuitive approach determined their use in various domains of the geometric modelling, such as the character animation [DKT98], the wavelet generation [SDRS96] or the multiresolution analysis and editing [ZSS97].

Surface as limit

Classical approach computes subdivision surfaces as a result of schemes for recursive refinement of control mesh. In early literature, subdivision was described as a generalization of the spline knot insertion to arbitrary control mesh. Indeed, Catmull-Clark [CC78] and Doo-Sabin [DS78] schemes can be respectively thought of as generalization of tensor product bicubic and biquadratic uniform B-splines to arbitrary topology. In this context, the subdivision surface $\mathcal{S}$ can be seen as the limit of the

sequence of the control meshes

$$\mathcal{S} = \lim_{k \to \infty} \mathcal{M}^k, \tag{3.1}$$

where the mesh $\mathcal{M}^{k+1}$ is obtained by applying set of refinement rules on $\mathcal{M}^k$, the mesh $\mathcal{M}^0$ is initial.

Nested rings sequences

In [PR08], Peters and Reif emphasize another way of understanding subdivision surfaces. They do so in order to formally characterize and study the analytical properties (particularly continuity) from the differential geometry point of view. They propose two possible means of looking at the subdivision process, each of them relevant for certain applications.

- The *mesh refinement* generates a sequence of refined control nets. Such a setup is suitable for implementation purposes.
- The *subdivision* generates nested sequences of surface rings. It is used for studying differential traits of the limit surface.

In this thesis, our main focus is on the *mesh refinement*. Study of the analytic properties is out of scope of this work, although we plan to do it. In chapter 7, we specify some ideas for the future work. These ideas also include the analytical approach to the proposed method.

3.2 General classification

In practice, a subdivision scheme consists of two main steps. First, the topological structure of the next iteration is determined (*splitting* or *topological step*). Typical operations in this step include inserting new vertices and leaving out some of the old, introducing new edges and faces, performing edge flip. Second, positions of old and/or new vertices are perturbed (*geometric step*).

The nature of splitting step yields another possible classification into *face* schemes, which split faces, and *vertex* schemes, splitting vertices.

Stationary schemes use the same set of subdivision rules over all refinement steps, as opposite to *nonstationary* or *variational schemes*. Locally in terms of fixed refinement level, a scheme is termed *uniform* or *shift-invariant* if the same rules are applied for each computed vertex throughout refinement step. Otherwise, a scheme is *adaptive* or *shift-variant*.

Finally, a subdivision scheme is *interpolating* if the limit surface interpolates the vertices of the initial mesh. Otherwise, the scheme is *approximating*.

3.3 Linear schemes

Generally, linear refinement scheme takes form

$$\mathbf{v}_i^{k+1} = \sum_{\mathbf{v}_j^k \in \mathcal{V}(\mathcal{M}^k)} a_{ij}^k \mathbf{v}_j^k \,. \tag{3.2}$$

The set of vertices $\mathcal{V}\left(\mathcal{M}^k\right)$ of the mesh $\mathcal{M}^k$ at level k is affinely mapped on the set vertices of mesh $\mathcal{M}^{k+1}$ using refinement coefficients $a_{ij}^k \in \mathbb{R}$, $\sum a_{ij}^k = 1$. The equation (3.2) can be narrowed using matrix notation to

$$\mathbf{P}^{k+1} = \mathbf{A}^k \mathbf{P}^k \,, \tag{3.3}$$

where $\mathbf{P}^k$ is a column vector of control points $\mathbf{v}_j^k \in \mathcal{M}^k$ and $\mathbf{A}^k = \left\{a_{ij}^k\right\}$ is a matrix with real coefficients called *subdivision matrix*.

The coefficients in subdivision matrix are chosen carefully to ensure desired properties of the limit surface, such as $\mathcal{C}^q$ continuity for some q. The proof of the convergence of a particular method leads to the structure of eigenvalues and eigenvectors of the matrix $\mathbf{A}^k$ [PR08].

Special subdivision rules need to be applied for boundary vertices. Such boundary rules are generally chosen in a way that ensures the limit boundary curve is a B-spline curve. When discussing the individual schemes, we omit the definition of the boundary rules. The details of the boundary rules can be found in [SZ00].

First subdivision schemes for surface refinement were inspired by work of Chaikin [Cha74]. His algorithm for curve generation uses a procedure known as *corner cutting*, using simple linear vertex combinations to compute vertex coordinates in the next iteration. The limit curve produced by the Chaikin's algorithm is in fact a piecewise quadratic $\mathcal{C}^1$-continuous B-spline.

For example, consider a closed polygon $\mathcal{P}$ with n vertices $\mathbf{v}_i = \mathbf{v}_i^0$ for $i = 0, \ldots, n-1$, such as in figure 3.1. Then the $k+1$st iteration of the corner cutting algorithm yields a refined control polygon with $2^k n$ vertices $\mathbf{v}_j^{k+1}$, computed according to the scheme

$$\mathbf{v}_{2i}^{k+1} = \tfrac{1}{4}\mathbf{v}_{i-1}^k + \tfrac{3}{4}\mathbf{v}_i^k, \tag{3.4a}$$

$$\mathbf{v}_{2i+1}^{k+1} = \tfrac{3}{4}\mathbf{v}_i^k + \tfrac{1}{4}\mathbf{v}_{i+1}^k, \tag{3.4b}$$

for $i = 0, \ldots, 2^k n - 1$. The indices in subscripts are taken modulo n.

3.3.1 Doo-Sabin (1978)

Doo-Sabin subdivision scheme was developed by extending the Chaikin's algorithm to surfaces, generalizing biquadratic uniform tensor-product B-splines for arbitrary

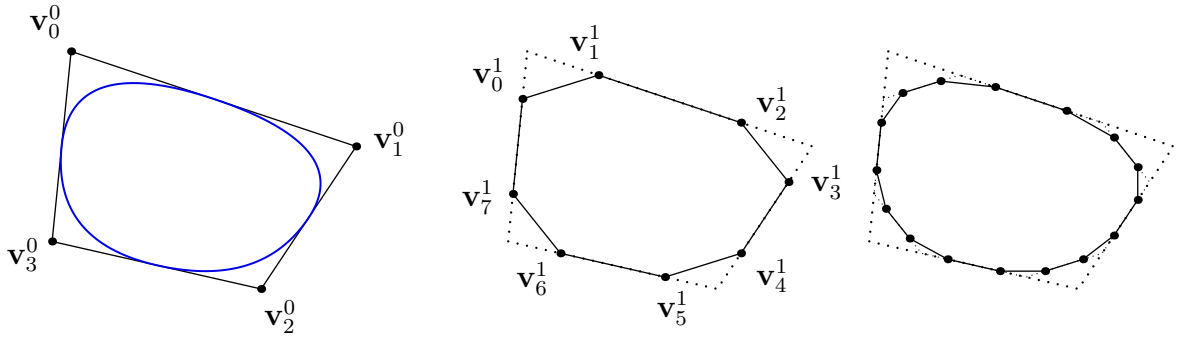


Figure 3.1: Chaikin's scheme for curves. (*Left*) Control polygon and limit curve, (*middle* and *right*) first two iterations of corner cutting algorithm.

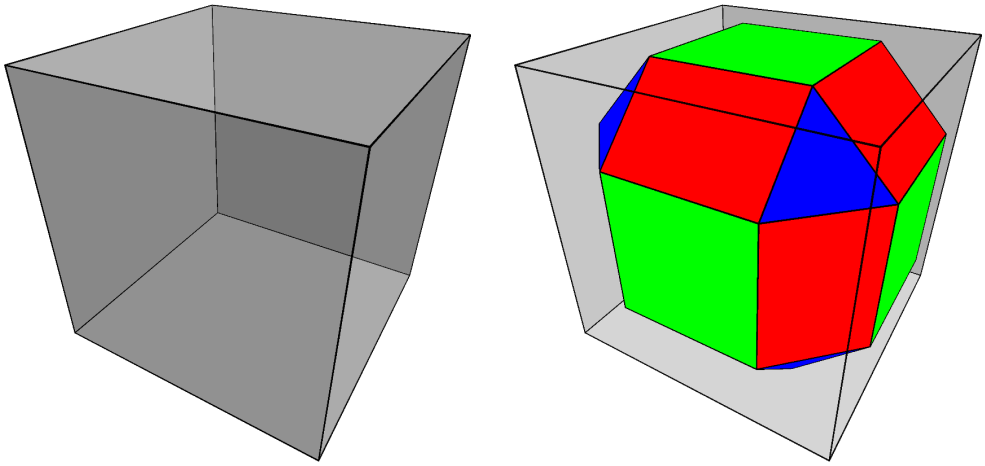


Figure 3.2: Cube subdivided once using the Doo-Sabin scheme. V-faces are rendered blue, E-faces are red, F-faces are green.

control net [DS78]. The limit surface is globally $\mathcal{C}^1$ continuous.

Topological step of the Doo-Sabin scheme uses corner cutting for surfaces. For each vertex $\mathbf{v}$ incident with n faces, n new vertices are introduced and connected to form a V-face. Each old edge e is replaced by new E-face, connecting newly inserted vertices incident with e. Similarly, each face f is replaced by a new F-face, connecting newly inserted vertices incident with f. For better illustration, see the first iteration of the scheme applied on cube visualised on figure 3.2.

The position of the new vertex is computed as a weighted average of the positions of n vertices $\mathbf{v}_i$ from the old face (see figure 3.3)

$$\hat{\mathbf{v}} = \sum_{i=0}^{n-1} \alpha_i \mathbf{v_i}. \tag{3.5}$$

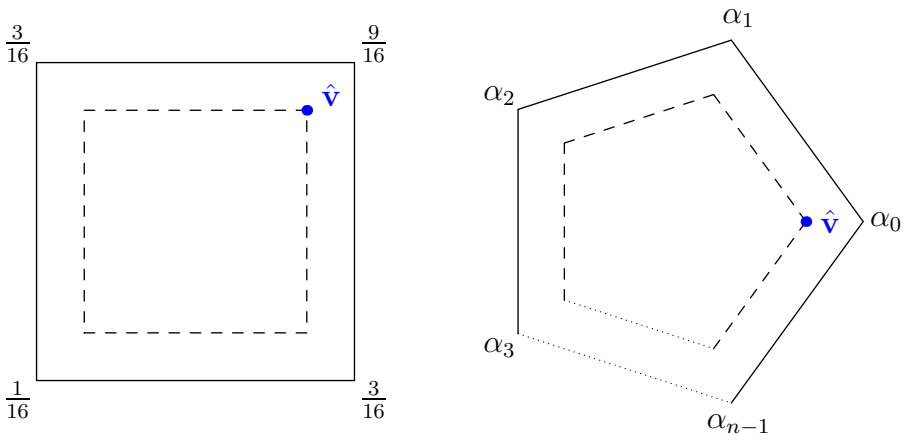


Figure 3.3: Masks for regular (*left*) and extraordinary vertex $\hat{\mathbf{v}}$ (*right*) in the Doo-Sabin scheme. The weights α_i are computed as in equation (3.6) or (3.7).

Doo and Sabin proposed using the following weights

$$\alpha_i = \begin{cases} \dfrac{1}{4} + \dfrac{5}{4n} & i = 0\,, \\ \dfrac{3 + 2\cos\left(2\pi\dfrac{i}{n}\right)}{4n} & i = 1, \ldots, n-1\,. \end{cases} \tag{3.6}$$

A different set of weights for this scheme was proposed by Catmull and Clark

$$\hat{\alpha}_i = \begin{cases} \dfrac{1}{2} + \dfrac{1}{4n} & i = 0\,, \\ \dfrac{1}{8} + \dfrac{1}{4n} & i = 1, n-1\,, \\ \dfrac{1}{4n} & i = 2, \ldots, n-2\,. \end{cases} \tag{3.7}$$

3.3.2 Catmull-Clark (1978)

The Catmull-Clark scheme is a generalization of tensor product bicubic B-splines to arbitrary quadrilateral mesh, see [CC78]. The produced limit surface is $\mathcal{C}^2$ continuous everywhere except at the extraordinary vertices, where it is $\mathcal{C}^1$. An extraordinary vertex is in this case every vertex, whose valence is not equal to 4 (or 3 for boundary vertices).

Unlike the Doo-Sabin, the Catmull-Clark is face-splitting. Each face f with the vertices $\mathbf{v}_i$, $i = 0, \ldots, 3$, introduces a new face vertex $\mathbf{v}_f$, computed as a barycenter of f

$$\mathbf{v}_f = \frac{\mathbf{v}_0 + \mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3}{4}\,. \tag{3.8}$$

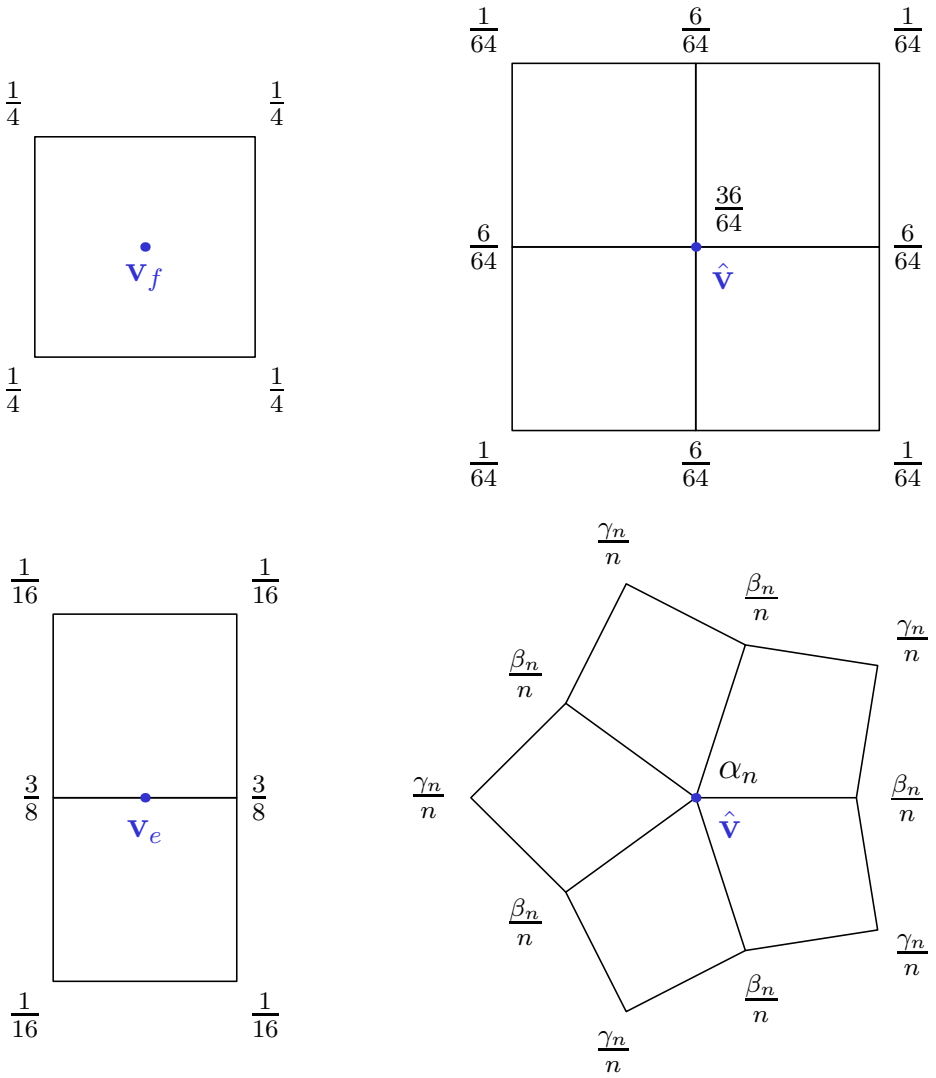


Figure 3.4: Masks in the Catmull-Clark subdivision scheme. (*Top left*) mask for face vertex $\mathbf{v}_f$, (*bottom left*) mask for edge vertex $\mathbf{v}_e$, (*top right*) mask for regular vertex $\hat{\mathbf{v}}$ with valency $n = 4$, (*bottom right*) mask for extraordinary vertex $\hat{\mathbf{v}}$ with valency $n \neq 4$. Here, $\beta_n = \frac{3}{2n}$, $\gamma_n = \frac{1}{4n}$, $\alpha_n = 1 - \beta_n - \gamma_n$.

Next, a new vertex $\mathbf{v}_e$ is added for each non-boundary edge e with endpoints $\mathbf{v}_0, \mathbf{v}_1$

$$\mathbf{v}_e = \frac{\mathbf{v}_0 + \mathbf{v}_1 + \mathbf{v}_{f_1} + \mathbf{v}_{f_2}}{4}, \tag{3.9}$$

where f_1, f_2 are faces incident with e. New edges are introduced, connecting respective incident face and edge vertices.

The last step is the perturbation of each old vertex $\mathbf{v}$. Let $\mathbf{f}$ be the barycenter of n face vertices, obtained from faces incident with $\mathbf{v}$. Let $\mathbf{m}$ be the barycenter of midpoints of n edges incident with $\mathbf{v}$. The new position $\hat{\mathbf{v}}$ of the vertex $\mathbf{v}$ is computed as

$$\hat{\mathbf{v}} = \frac{\mathbf{f} + 2\mathbf{m} + (n-3)\mathbf{v}}{n}. \tag{3.10}$$

The Catmull-Clark scheme is a very popular algorithm for mesh smoothing. In

1997, the scheme was used in Pixar's short movie *Geri's Game* (figure 3.5) for character modelling. *Geri's Game* later received Academy Award for Best Animated Short Film, marking an important milestone in short history of subdivision surfaces. For more details, see [DKT98].

Figure 3.5: Academy Award-winning short movie *Geri's Game* uses a modified version of the Catmull-Clark subdivision scheme. Picture of Geri courtesy of [DKT98].

3.3.3 Loop (1987)

Introduced in the master's thesis of Charles Loop [Loo87], the Loop's subdivision scheme generalizes quartic triangular B-splines. Operating on a triangular control mesh, the limit surface is $\mathcal{C}^2$ continuous everywhere except at the extraordinary vertices (their valence is not equal to 6), where the surface is $\mathcal{C}^1$.

The algorithm starts by cutting each edge e in two parts by introducing new E-vertex $\mathbf{v}_e$ at the midpoint of e. Each face F is split to four faces by introducing three new edges, connecting newly inserted vertices, see figure 3.6. Such topological change is called *dyadic split* and produces 1-to-4 refinement – each triangle is replaced by four sub-triangles.

If $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are the neighbours of $\mathbf{v}$ (its valence is therefore n), the position $\hat{\mathbf{v}}$ of the vertex $\mathbf{v}$ in the next iteration is computed using

$$\hat{\mathbf{v}} = (1 - n\beta)\,\mathbf{v} + \sum_{i=1}^{n} \beta \mathbf{v}_i \,. \tag{3.11}$$

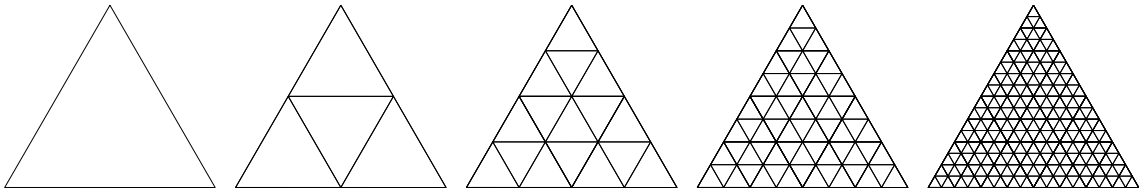


Figure 3.6: Dyadic split is used in many refinement schemes and produces 1-to-4 refinement.

The position of the edge vertex $\mathbf{v}_e$ is

$$\mathbf{v}_e = \frac{3\mathbf{v} + 3\mathbf{v}_j + \mathbf{v}_{j-1} + \mathbf{v}_{j+1}}{8}, \tag{3.12}$$

where the endpoints of e are $\mathbf{v}$ and $\mathbf{v}_j$. The indices in subscripts are taken modulo n. The original value of β proposed by Loop is

$$\beta = \frac{5}{8n} - \frac{1}{n}\left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2. \tag{3.13}$$

Warren [WW01] proposed an alternative choice of weights:

$$\hat{\beta}(n) = \begin{cases} \dfrac{3}{16} & n = 3, \\ \dfrac{3}{8n} & n > 3. \end{cases} \tag{3.14}$$

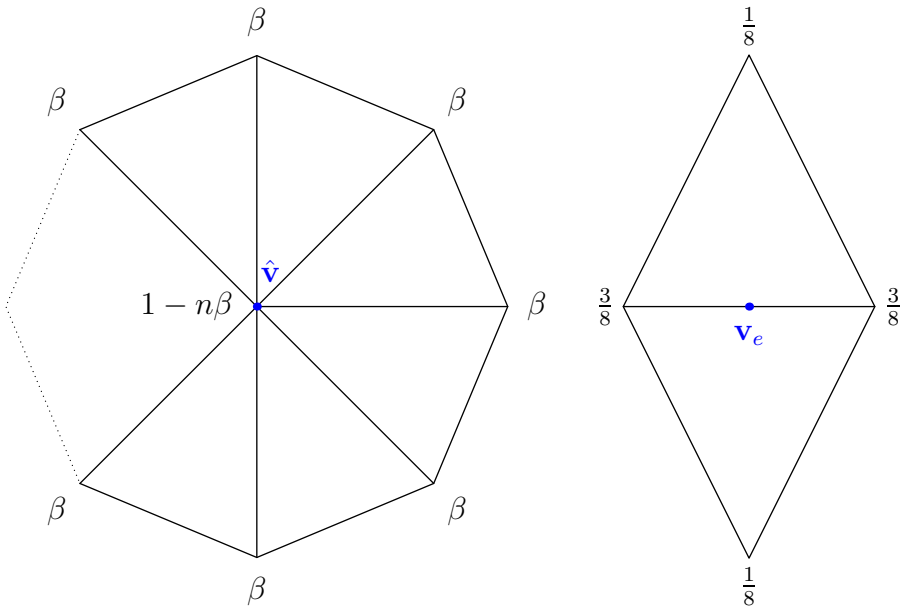


Figure 3.7: Vertex and edge masks for the Loop scheme.

3.3.4 Butterfly (1990)

The original Butterfly subdivision scheme was proposed by Dyn, Levin and Gregory [DLG90]. Designed for triangular meshes, the scheme is interpolating with the limit surface being $\mathcal{C}^1$ continuous everywhere except at the extraordinary vertices of valence $k = 3$ and $k > 7$. Zorin et al. [ZSS96] modified the scheme to be $\mathcal{C}^1$ continuous for an arbitrary mesh.

Topological step in the Butterfly scheme is the same as in the Loop's scheme. The difference between them lies in averaging step. Unlike Loop, Butterfly scheme is interpolating. Therefore, the positions of old vertices remain unchanged. For computing the position of new edge vertex $\mathbf{v}_e$, the scheme uses eight-point rule, which gave the scheme its name. The rule's mask has a shape of butterfly, see figure 3.8.

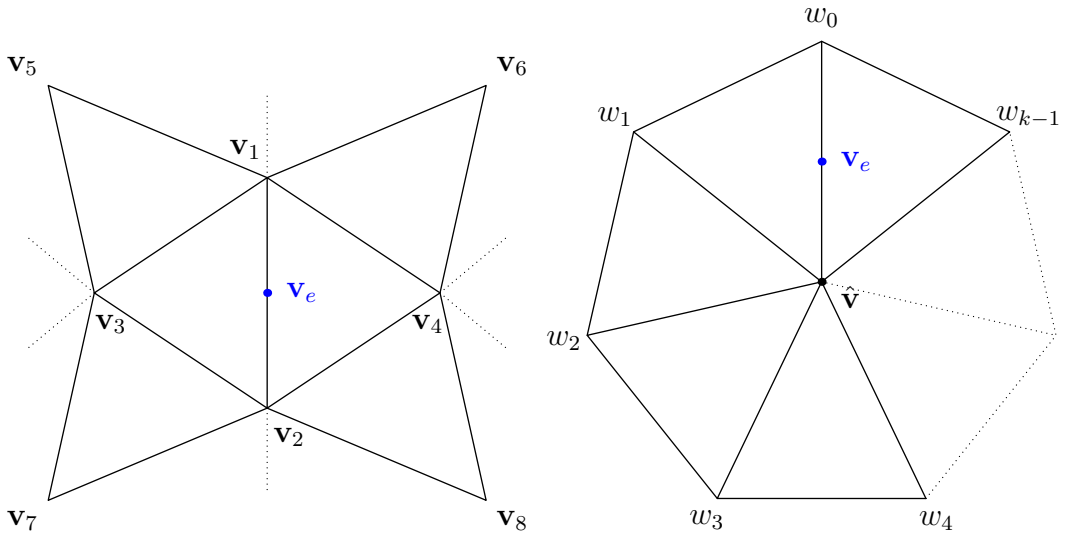


Figure 3.8: Masks for the an edge vertex $\mathbf{v}_e$ in the Butterfly scheme. (*Left*) original mask, (*right*) modified mask for an edge adjacent to an extraordinary vertex $\hat{\mathbf{v}}$ whose valency is k. Weights w_i for the modified mask are specified in equation (3.17).

Using the notation from figure 3.8, position of edge vertex $\mathbf{v}_e$ is

$$\mathbf{v}_e = \tfrac{1}{2}(\mathbf{v}_1 + \mathbf{v}_2) + 2\omega(\mathbf{v}_3 + \mathbf{v}_4) - \omega(\mathbf{v}_5 + \mathbf{v}_6 + \mathbf{v}_7 + \mathbf{v}_8). \tag{3.15}$$

The parameter w in (3.15) controls the tension of the limit surface. The more ω approaches 0, the more is limit surface tightened toward piecewise linear spline surface. Dyn et al. suggest $\omega = \frac{1}{16}$.

To achieve the local tension control, each original control vertex $\mathbf{v}_i^0$ can be assigned its weight parameter $\omega_i^0 = \omega(\mathbf{v}_i^0)$. Weights for newly inserted vertices are obtained using linear interpolation between weights of the edge midpoints. More flexibility in the tension control is possible by replacing the scalar ω with 3×3 matrix Ω and writing eight-point rule as

$$\mathbf{v}_e = \tfrac{1}{2}(\mathbf{v}_1 + \mathbf{v}_2) + \Omega\mathbf{s} \tag{3.16}$$

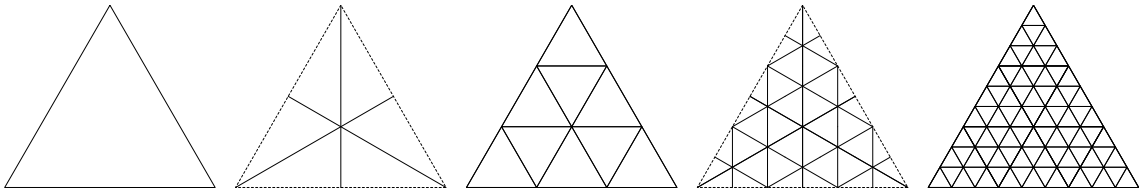


Figure 3.9: Four iterations of the splitting step of the $\sqrt{3}$-subdivision scheme, performed on an interior triangle.

with $\mathbf{s} = 2(\mathbf{v}_3 + \mathbf{v}_4) - (\mathbf{v}_5 + \mathbf{v}_6 + \mathbf{v}_7 + \mathbf{v}_8)$. This allows for the local control of the direction and the magnitude of the tension.

A modification in [ZSS96] solves the tangent plane continuity near extraordinary vertices using the edge mask similar to the one applied in Loop's scheme (see figure 3.8), with the weights

$$\begin{aligned} k = 3: &\quad w_0 = \tfrac{5}{12},\ w_1 = w_2 = -\tfrac{1}{12}, \\ k = 4: &\quad w_0 = \tfrac{3}{8},\ w_2 = -\tfrac{1}{8},\ w_1 = w_3 = 0, \\ k \geq 5: &\quad w_i = \tfrac{1}{n}\left(\tfrac{1}{4} + \cos\tfrac{2\pi i}{n} + \tfrac{1}{2}\cos\tfrac{4\pi i}{n}\right). \end{aligned} \tag{3.17}$$

3.3.5 $\sqrt{3}$-subdivision (2000)

The $\sqrt{3}$-subdivision scheme was developed by Kobbelt [Kob00]. This triangular scheme is approximating, $\mathcal{C}^2$ continuous at the regular vertices (their valence is equal to 6) and $\mathcal{C}^1$ continuous at the extraordinary vertices.

The splitting step of the scheme differs from the one used in the Loop's scheme. Kobbelt introduced a novel method for altering the triangular topology. First, a new vertex is introduced per triangle face and connected to all vertices of the triangle. Second, the old edges are flipped, see figure 3.9 and also figure 4.1.

The position of the newly introduced vertex $\mathbf{v}_T$ in triangle $T = \mathbf{v}_i\mathbf{v}_j\mathbf{v}_k$ is simply computed as the center of gravity of triangle T

$$\mathbf{v}_T = \frac{1}{3}\left(\mathbf{v}_i + \mathbf{v}_j + \mathbf{v}_k\right). \tag{3.18}$$

The relaxation of the old vertex $\mathbf{v}$ is performed similarly to the Loop's scheme. If the valency of the old vertex $\mathbf{v}$ is n and its neighbours are $\mathbf{v}_i, i = 1, \ldots, n$, then the new position of $\mathbf{v}$ is computed as

$$\hat{\mathbf{v}} = (1 - \alpha_n)\,\mathbf{v} + \frac{\alpha_n}{n}\sum_{i=1}^{n}\mathbf{v}_i, \quad \alpha_n = \frac{4 - 2\cos\left(\frac{2\pi}{n}\right)}{9}. \tag{3.19}$$

Such choice of the parameter α_n yields the desired properties of the scheme. Figure 3.10 shows both masks of the $\sqrt{3}$-subdivision.

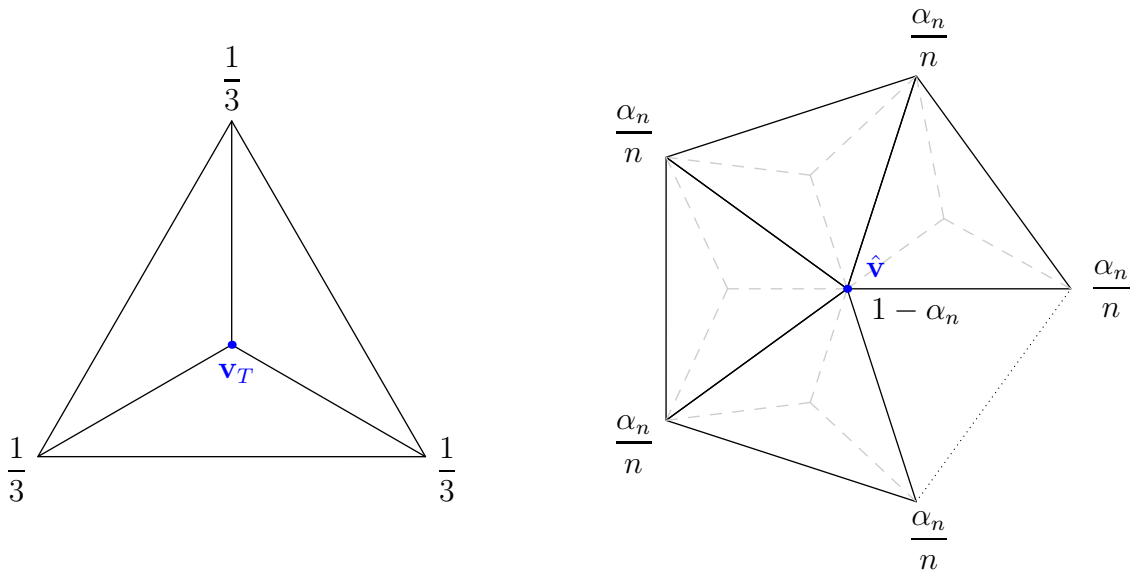


Figure 3.10: The vertex masks in the $\sqrt{3}$-subdivision scheme. The face vertex $\mathbf{v}_T$ is obtained by simply computing the barycenter of the triangle T. The new position $\hat{\mathbf{v}}$ of the old vertex $\mathbf{v}$ is computed by blending the positions of the neighbours of $\mathbf{v}$.

Before the $\sqrt{3}$-subdivision scheme was developed, the majority of the triangular subdivision schemes used *dyadic split*, see figure 3.6. The generalization of such refinement is *n-adic split*. Each edge is cut into n parts, while each triangle is cut into n^2 sub-triangles. However, the larger the constant n gets, the more subdivision rules are needed, e.g. for $n = 3$, we need four subdivision rules. This is the main reason why most triangular schemes use $n = 2$. Two consecutive applications of the $\sqrt{3}$-refinement (figure 3.11) produce 1-to-9 or *triadic* refinement of the original mesh, hence the square root in the name.

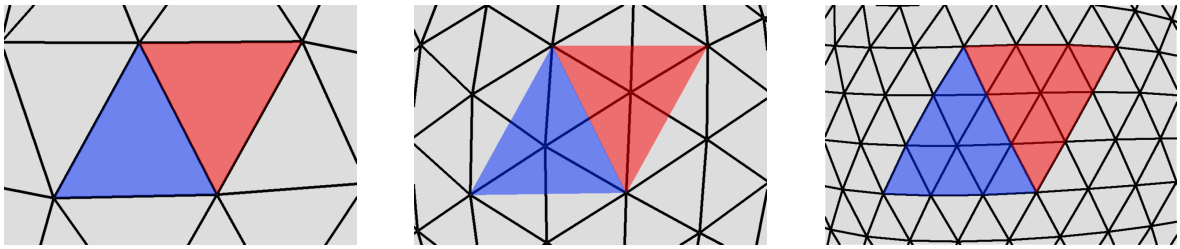


Figure 3.11: After two iterations, the splitting step of the $\sqrt{3}$-subdivision scheme produces 1-to-9 refinement (triadic split) of each triangle.

3.4 Nonlinear schemes

For the linear schemes – both approximating and interpolating – the positions of the new vertices are computed as the linear combinations of the vertices from the previous iteration. Consequently, vertices in arbitrary iteration $\mathcal{M}^j$ (particularly the limit surface $\mathcal{S} = \mathcal{M}^\infty$) can be expressed as the linear combinations of the initial mesh $\mathcal{M}^0$ in

a natural way. For the nonlinear schemes, this condition does not hold true.

The theory of the nonlinear schemes is much more difficult to handle, compared to the linear case. Here, we present a brief summary of existing nonlinear schemes for surface refinement.

The surface algorithm presented in this thesis was inspired by the nonlinear circle-preserving *curve refinement* algorithm, proposed by Chalmovianský and Jüttler [CJ07].

3.4.1 Circular arcs (2000)

Karbacher et al. [KSH00] introduced nonlinear interpolating algorithm for refinement of triangular mesh. They assume the input mesh is a dense approximation of some smooth surface, e.g. a mesh reconstructed from the range images. Such mesh can be locally approximated by circular arcs using the vertex positions and the vertex normals.

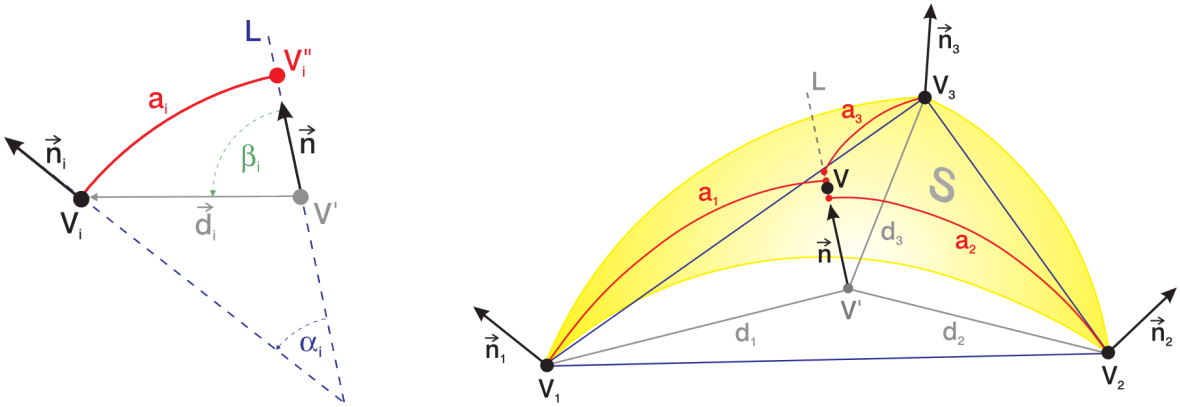


Figure 3.12: The nonlinear surface refinement method of [KSH00] uses the blending of circular arcs to obtain the local approximation of the mesh. The positions of the edge vertices are then computed by elevating the edge midpoints on S. Picture courtesy of [KSH00].

The splitting step is identical to the Loop's linear scheme, see section 3.3.3. To obtain the positions of the edge vertices in the triangle $T = \mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$, the surface $\mathcal{S}$, defined over the triangular domain T is first constructed.

Fix the point $\mathbf{p} \in T$. Such point can be expressed using the barycentric coordinates (with respect to T) as

$$\mathbf{p} = \sum_{i=1}^{3} b_i \mathbf{v}_i, \quad b_i \geq 0, \sum_{i=1}^{3} b_i = 1. \tag{3.20}$$

The normal $\vec{\eta}$ at $\mathbf{p}$ is then defined as

$$\vec{\eta} = \frac{\sum_{i=1}^{3} b_i \vec{\mathbf{n}}_i}{\left\| \sum_{i=1}^{3} b_i \vec{\mathbf{n}}_i \right\|}, \tag{3.21}$$

where the vector $\vec{\mathbf{n}}_i$ is the normal at the vertex $\mathbf{v}_i$.

Denote the distance vector $\vec{\mathbf{d}}_i = \mathbf{p} - \mathbf{v}_i$ and its length $d_i = \left\|\vec{\mathbf{d}}_i\right\|$. The pair $\mathbf{p}, \vec{\eta}$ determines the line L, passing through $\mathbf{p}$ in the direction of η. The line L and each normal $\vec{\mathbf{n}}_i$ define the circular arc a_i. Denote the angle between the normal vectors $\vec{\eta}$ and $\vec{\mathbf{n}}_i$ as α_i, the angle between $\vec{\eta}$ and the distance vector $\vec{\mathbf{d}}_i$ as β_i, see figure 3.12.

For each arc a_i, we define the point $\tilde{\mathbf{p}} = \mathbf{p} + \delta_i \eta$, where

$$\delta_i = \|\mathbf{p} - \tilde{\mathbf{p}}\| \approx d_i \frac{\cos\left(\beta_i - \frac{1}{2}\alpha_i\right)}{\cos\left(\frac{1}{2}\alpha_i\right)} \tag{3.22}$$

The surface $\mathcal{S}$ is defined as a set of points

$$\sum_{i=1}^{3} b_i \left(\mathbf{v}_i + \delta_i(\mathbf{b})\, \vec{\eta}(\mathbf{b})\right) \tag{3.23}$$

for all possible parameter values $\mathbf{b} = (b_1, b_2, b_3)$, $b_i \geq 0$, $\sum_{i=1}^{3} b_i = 1$.

The position of the newly inserted edge vertex is computed by orthogonally projecting the corresponding edge midpoint on the surface $\mathcal{S}$, see figure 3.12. Such scheme does not produce $\mathcal{G}^1$ continuous surface. [KSH00] suggests the $\mathcal{G}^1$ continuity can be enforced by smoothing the final surface. Such transition does not come for free, as the smooth limit surface no longer interpolates the input data.

3.4.2 Local spherical coordinates (2003)

Aspert et al. [AEV03] introduced the interpolating refinement scheme, which uses local spherical coordinates to obtain the subdivided mesh. They present an algorithm for univariate data, which is then used to derive an algorithm for triangular meshes.

Once again, the splitting step is the same as in the Loop's scheme (section 3.3.3). To obtain the positions of the newly introduced edge vertices, a local coordinate system is constructed at each vertex $\mathbf{v}_i$, defined by the tangent plane at $\mathbf{v}_i$ and the normal vector at $\mathbf{v}_i$, see figure 3.13. The local spherical coordinate system at the vertex $\mathbf{v}_i$ is denoted $\mathcal{R}_{\mathbf{v}_i}$.

The vector $\vec{\mathbf{v}}_{i,k}$ from the $\mathbf{v}_i$ into its neighbouring vertex $\mathbf{v}_k$ is constructed and normalized to obtain the vector $\vec{\mathbf{w}}_{i,k}$. The spherical coordinates of $\vec{\mathbf{w}}_{i,k}$ in $\mathcal{R}_{\mathbf{v}_i}$ are defined by the angles $\theta_{i,k}$ and $\phi_{i,k}$, see figure 3.13. Similarly, the coordinates of $\vec{\mathbf{v}}_{i,k}$ in $\mathcal{R}_{\mathbf{v}_i}$ are $(r_{i,k}, \theta_{i,k}, \phi_{i,k})$, where $r_{i,k} = \|\vec{\mathbf{v}}_{i,k}\|$ is the length of edge connecting $\mathbf{v}_i$ and $\mathbf{v}_k$. Define the vectors

$$\vec{\mathbf{a}}_{i,k} = \left(\frac{r_{i,k}}{2}, h\left(\theta_{i,k}\right), \phi_{i,k}\right), \tag{3.24a}$$

$$\vec{\mathbf{b}}_{k,i} = \left(\frac{r_{i,k}}{2}, h\left(\theta_{k,i}\right), \phi_{k,i}\right), \tag{3.24b}$$

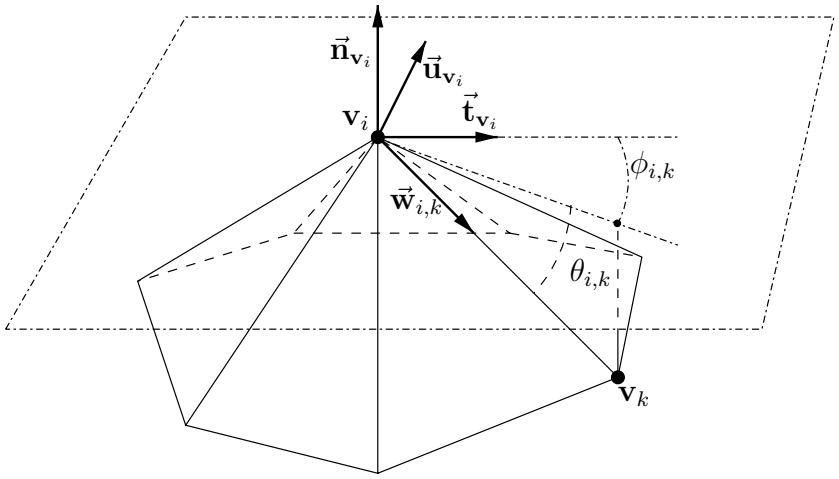


Figure 3.13: The local coordinate system around vertex $\mathbf{v}_i$ is defined by orthonormal basis of tangent plane at $\mathbf{v}_i$ (vectors $\vec{\mathbf{t}}_{\mathbf{v}_i}$ and $\vec{\mathbf{u}}_{\mathbf{v}_i}$) and the unit normal vector at $\mathbf{v}_i$ (vector $\vec{\mathbf{n}}_{\mathbf{v}_i}$). All dash-dotted lines in the figure belong to the same plane. Picture courtesy of [AEV03].

where h is a function, defined as

$$h(\alpha) = \begin{cases} \alpha, & \text{if } -\pi < \alpha \leq -\frac{\pi}{2}, \\ -\frac{1}{\pi}\left(\alpha + \frac{\pi}{4}\right)^2 \left[\frac{24}{\pi}\left(\alpha + \frac{\pi}{4}\right) + 10\right] + \frac{\alpha}{2}, & \text{if } -\frac{\pi}{2} < \alpha < -\frac{\pi}{4}, \\ \frac{\alpha}{2}, & \text{if } -\frac{\pi}{4} \leq \alpha \leq \frac{\pi}{4}, \\ \frac{1}{\pi}\left(\alpha - \frac{\pi}{4}\right)^2 \left[-\frac{24}{\pi}\left(\alpha - \frac{\pi}{4}\right) + 10\right] + \frac{\alpha}{2}, & \text{if } \frac{\pi}{4} < \alpha \leq \frac{\pi}{2}, \\ \alpha, & \text{if } \frac{\pi}{2} \leq \alpha < \pi. \end{cases} \tag{3.25}$$

Then there exist unique points $\mathbf{a}_{i,k}, \mathbf{b}_{k,i}$, such that $\mathbf{a}_{i,k} = \mathbf{v}_i + \vec{\mathbf{a}}_{i,k}$ and $\mathbf{b}_{k,i} = \mathbf{v}_k + \vec{\mathbf{b}}_{k,i}$. If we denote by $\mathbf{v}_e$ the edge vertex introduced between $\mathbf{v}_i$ and $\mathbf{v}_k$, then its position is computed as

$$\mathbf{v}_e = \frac{\mathbf{a}_{i,k} + \mathbf{b}_{k,i}}{2}. \tag{3.26}$$

3.4.3 Face/Normal based subdivision (2005)

Yang proposed two nonlinear surface refinement algorithms in [Yan05], face based scheme and normal based scheme. The latter is shown to be $\mathcal{G}^1$ continuous. Both methods are interpolating and use dyadic split in the topological step.

Consider the edge vertex $\mathbf{q} = (x, y, z)$ introduced on the edge with endpoints $\mathbf{p}_1 = (x_1, y_1, z_1), \mathbf{p}_2 = (x_2, y_2, z_2)$. Denote $\mathbf{p}_3, \ldots, \mathbf{p}_l$ the neighbouring vertices of $\mathbf{p}_1$ and $\mathbf{p}_2$, denote $T_0, \ldots, T_{l-1}$ the triangles adjacent to $\mathbf{p}_1$ or $\mathbf{p}_2$. Denote π_i the plane where the triangle T_i is lying, see figure 3.14.

In the *face based scheme*, the new vertex $\mathbf{q}$ is calculated by minimizing the objective

function

$$F(\mathbf{q}) = \sum_{i=0}^{l-1} \alpha_i \left(\vec{\mathbf{a}}_i \mathbf{q}\right)^2 + \sum_{i=1}^{2} \beta_j \left(\mathbf{q} - \mathbf{p}_j\right)^2, \tag{3.27}$$

where $\vec{\mathbf{a}}_i\mathbf{q} = a_i x + b_i y + c_i z + d$ is the distance of $\mathbf{q}$ from the plane $\pi_i, i = 0, \ldots, l-1$, $\mathbf{q} - \mathbf{p}_j$ is the distance of $\mathbf{q}$ from $\mathbf{p}_j, j = 1, 2$, the scalars α_i, β_j are weights.

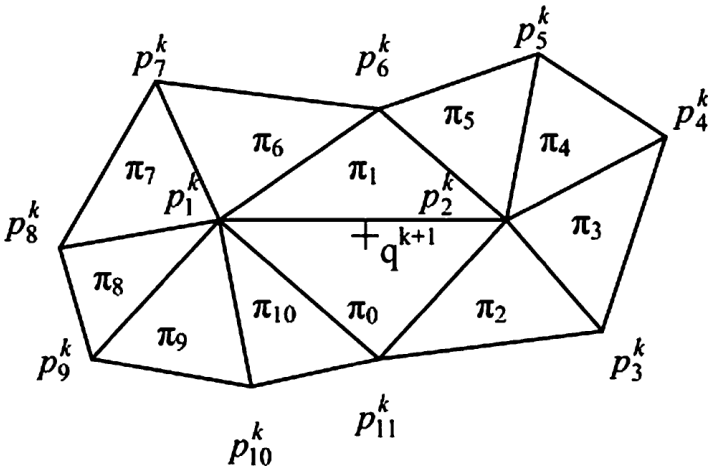


Figure 3.14: The neighbourhood of the edge $\mathbf{p}_1\mathbf{p}_2$ is used for computation of edge vertex $\mathbf{q}$. Picture courtesy of [Yan05].

The objective function F is quadratic in $\mathbf{q}$ and equation (3.27) can be rewritten as

$$F(\mathbf{q}) = \mathbf{q}^\top \left(\sum_{i=0}^{l-1} \alpha_i \vec{\mathbf{a}}_i^\top \vec{\mathbf{a}}_i \right) \mathbf{q} + \mathbf{q}^\top \left(\beta_1 \mathbf{Q}_1 + \beta_2 \mathbf{Q}_2\right) \mathbf{q} = \mathbf{q}^\top \mathbf{Q} \mathbf{q}, \tag{3.28}$$

where $\mathbf{Q} = \sum_{i=0}^{l-1} \alpha_i \vec{\mathbf{a}}_i^\top \vec{\mathbf{a}}_i + \beta_1 \mathbf{Q}_1 + \beta_2 \mathbf{Q}_2$ and

$$\mathbf{Q}_1 = \begin{pmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ -x_1 & -y_1 & -z_1 & x_1^2 + y_1^2 + z_1^2 \end{pmatrix}, \mathbf{Q}_2 = \begin{pmatrix} 1 & 0 & 0 & -x_2 \\ 0 & 1 & 0 & -y_2 \\ 0 & 0 & 1 & -z_2 \\ -x_2 & -y_2 & -z_2 & x_2^2 + y_2^2 + z_2^2 \end{pmatrix}.$$

The matrix $\mathbf{Q} = \{q_{ij}\}$ is obtained as a sum of symmetric matrices and is therefore symmetric. The solution of minimizing (3.28) is

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{pmatrix}^{-1} \begin{pmatrix} -q_{14} \\ -q_{24} \\ -q_{34} \end{pmatrix}. \tag{3.29}$$

The weights α_i are chosen proportionally to the area of T_i and anti-proportionally to the approximation of the angle between the plane π_i and the tangent plane at $\mathbf{q}$.

In the *normal based scheme*, the position of the new edge vertex $\mathbf{q}$ is computed using

the data coming from the estimations of normal vectors. Suppose $T_j, j = 0, \ldots, m_i - 1$ are all triangles adjacent to vertex $\mathbf{v}_i$ with the normal vectors η_j. The normal vector at vertex $\mathbf{v}_i$ is estimated as

$$\vec{\mathbf{n}}_i = \frac{\sum_{j=0}^{m_i-1} \phi_j \vec{\eta}_j}{\left\| \sum_{j=0}^{m_i-1} \phi_j \vec{\eta}_j \right\|}. \tag{3.30}$$

The edge vertex $\mathbf{q}$ corresponding to the edge $e = \mathbf{v}_i\mathbf{v}_j$ is computed as a midpoint of e, displaced by a linear combination of normal vectors at the endpoints of e,

$$\mathbf{q} = \frac{\mathbf{v}_i + \mathbf{v}_j}{2} + w \left(d_{ij}\vec{\eta}_i + d_{ji}\vec{\eta}_j \right), \tag{3.31}$$

where the scalar w is a parameter and d_{ij} is computed as

$$d_{ij} = \frac{1}{2} \left(\mathbf{v}_i - \mathbf{v}_j \right) \eta_i. \tag{3.32}$$

[Yan05] shows the normal based scheme converges to $\mathcal{G}^1$ limit surface if the initial mesh meets the specified criteria and the parameter w is well-chosen. For fast convergence and smooth limit surface, the choice of w between 0.2 and 0.4 is appropriate.

To further enhance the fairness of the limit surface, [Yan05] describes the improvement to the normal based scheme under tangent plane constraint. An objective function similar to (3.27) is minimized to obtain the position of the desired vertex.

Chapter 4

Refinement via Quadric Fitting

In this chapter, the new approach to nonlinear subdivision of surfaces is introduced. The basic idea of the proposed method is to look for the new points on a quadric surface, which is the best local approximation of the mesh with respect to well chosen objective function. In the text, we talk about *quadric fitting refinement* or simply QFR when referencing our method.

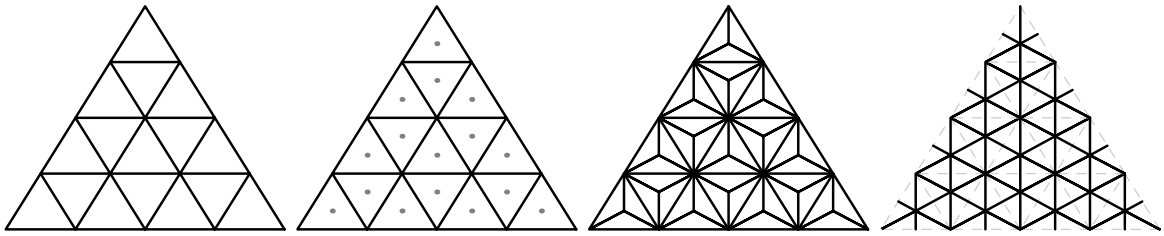
Given the mesh $\mathcal{M}^k$ with the vertices $\mathbf{p}_i^k$ and the associated normal vectors $\vec{\mathbf{n}}_i^k$, $i = 1, \ldots, N_k$, we want to refine $\mathcal{M}^k$ to obtain a denser mesh $\mathcal{M}^{k+1}$ with the vertices $\mathbf{p}_i^{k+1}$ and the normal vectors $\vec{\mathbf{n}}_i^{k+1}$, $i = 1, \ldots, N_{k+1}$. The refinement is performed locally, i.e. the positions of the new vertices are computed with respect to the local geometry of the mesh $\mathcal{M}^k$. The search for the new vertex is based on the minimization of the objective function.

4.1 Topological step

The topological step of our method is identical to the splitting step used in Kobbelt's $\sqrt{3}$-subdivision scheme [Kob00]. For the sake of conciseness and clarity, we shall refer to the $\sqrt{3}$-*refinement* instead of *the splitting step of the* $\sqrt{3}$-*subdivision scheme*. For more details on the $\sqrt{3}$-subdivision scheme, see section 3.3.5.

Without the flipping of old edges, the valency of each vertex would increase with each refinement step. The flipping ensures the valency of the old vertices remains unchanged, while the new vertices are always regular, which means their valency is equal to 6. The only extraordinary vertices in $\mathcal{M}^k$ (those with valency other than 6) are the extraordinary vertices of $\mathcal{M}^0$. Figure 4.1 shows the $\sqrt{3}$-refinement on a regular grid for better illustration.

The interpolation requirement ensures we only need to compute the position of a single vertex for each triangle in the geometric step. The choice of $\sqrt{3}$-refinement seems natural, although our initial idea was to simply subdivide each triangle at center without flipping the old edges. For reasons stated earlier in this section, we ran into



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

Figure 4.1: Splitting step of Kobbelt's $\sqrt{3}$-subdivision or $\sqrt{3}$-*refinement* used in our scheme. The topology of initial mesh (a) is altered by introducing new vertex (b) and three new edges per triangle, splitting it in four (c). Finally, old edges are flipped (d).

problems with increasing valency. This led us to the Kobbelt's method, which proved to be effective for our case.

4.2 Computing position of the new vertex

Suppose we want to subdivide the mesh $\mathcal{M}$. Let $\mathcal{V}(\mathcal{M})$ be the set of all vertices of $\mathcal{M}$. We are looking for the position of the new vertex $\mathbf{v}$ introduced in the triangle $T = \mathbf{v}_0\mathbf{v}_1\mathbf{v}_2$. The set $\mathcal{N}_\mathrm{T}$ of vertices is called an $m-$neighbourhood of T if

$$\mathcal{N}_\mathrm{T} \coloneqq \{\mathbf{p} \in \mathcal{V}(\mathcal{M}) \ : \ \mathcal{D}_\mathrm{T}(\mathbf{p}) \leq m\} \tag{4.1}$$

for some $m \in \mathbb{N}$, where

$$\mathcal{D}_\mathrm{T}(\mathbf{p}) \coloneqq \min_{\tilde{\mathbf{v}} \in \mathcal{V}(\mathrm{T})} (\mathcal{D}(\mathbf{p}, \tilde{\mathbf{v}})) \tag{4.2}$$

is relative distance of vertex $\mathbf{p}$ from triangle T and $\mathcal{D}$ is a *graph distance* on $\mathcal{M}$

$$\mathcal{D}(\mathbf{a}, \mathbf{b}) \coloneqq \text{number of edges in the shortest path connecting } \mathbf{a} \text{ and } \mathbf{b}. \tag{4.3}$$

We choose m to be the smallest natural number, for which $|\mathcal{N}_\mathrm{T}| \geq 9$. Typically, this yields $m = 1$ or $m = 2$. The choice of 9 as minimal number is justified later in the text (in section 4.3). An example of 1-neighbourhood is shown on figure 4.2.

The vertex $\mathbf{v}$ is picked out of the quadric surface

$$\mathcal{Q} \coloneqq \left\{(x, y, z) \in \mathbb{E}^3 \ : \ f(x, y, z) = 0\right\}, \tag{4.4}$$

where

$$\begin{aligned} f(x, y, z) = a_{11}x^2 + a_{22}y^2 + a_{33}z^2 + 2(a_{12}xy + a_{13}xz + a_{23}yz) + \\ + \ 2(a_{14}x + a_{24}y + a_{34}z) + a_{44} \end{aligned} \tag{4.5}$$

is an unknown trivariate polynomial with real coefficients. Using matrix notation, the

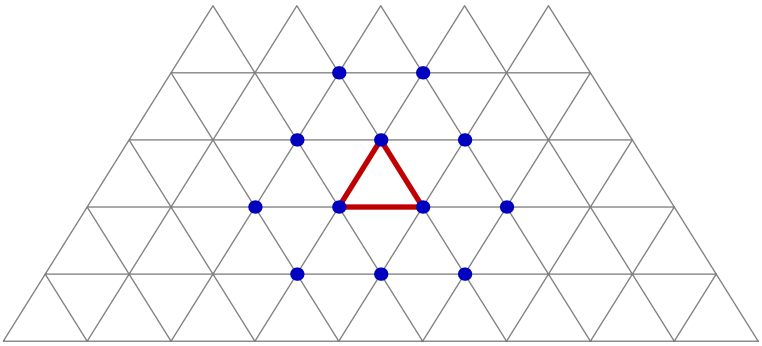


Figure 4.2: Visualisation of the set $\mathcal{N}_T$ (1-neighbourhood) for triangle T on a regular grid.

equation (4.5) can be written down as

$$f(\mathbf{x}) = \tilde{\mathbf{x}}^\top \mathbf{A}\, \tilde{\mathbf{x}}, \tag{4.6}$$

where

$$\tilde{\mathbf{x}} := \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \qquad \mathbf{A} := \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix}, \tag{4.7}$$

$\tilde{\mathbf{x}}$ stands for homogenous coordinates of $\mathbf{x} \in \mathbb{E}^3$, $\mathbf{A}$ denotes the symmetric matrix of coefficients a_{ij} from the equation (4.5).

4.3 Fitting a quadric to vertices

We look for such a quadric $\mathcal{Q}$ that is the best approximation of mesh $\mathcal{M}$ in a close neighbourhood of triangle T. For this purpose, we use the set $\mathcal{N}_T$ defined in (4.1). In order to specify $\mathcal{Q}$, exactly 10 unknown coefficients $\{a_{ij},\ 1 \leq i \leq j \leq 4\}$ need to be computed up to a non-zero multiple. This clarifies the required condition on cardinality of $\mathcal{N}_T$.

To improve the reading comprehension of this section, we use the following notation for gradient operators:

$$\nabla^{\mathrm{a}} f = \left(\frac{\partial f}{\partial a_{11}} \quad \frac{\partial f}{\partial a_{22}} \quad \frac{\partial f}{\partial a_{33}} \quad \frac{\partial f}{\partial a_{12}} \quad \frac{\partial f}{\partial a_{13}} \quad \frac{\partial f}{\partial a_{23}} \quad \frac{\partial f}{\partial a_{14}} \quad \frac{\partial f}{\partial a_{24}} \quad \frac{\partial f}{\partial a_{34}} \quad \frac{\partial f}{\partial a_{44}} \right) \tag{4.8}$$

denotes the gradient with respect to variables $a_{11}, \ldots, a_{44}$, while

$$\nabla^{\mathrm{x}} f = \left(\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y} \quad \frac{\partial f}{\partial z} \right) \tag{4.9}$$

is the gradient with respect to x, y, z.

Now, suppose $\mathcal{N}_\mathrm{T} = \{\mathbf{p}_i\,,\ i = 1, \ldots, n\}$, where $\mathbf{p}_i = (x_i, y_i, z_i)$. Ideally, $\mathcal{Q}$ interpolates $\mathcal{N}_\mathrm{T}$, meaning f vanishes in every point from $\mathcal{N}_\mathrm{T}$. In general case though, such an interpolation condition cannot be guaranteed. Instead, we compute the vector

$$\vec{a} \coloneqq \begin{pmatrix} a_{11} & a_{22} & a_{33} & a_{12} & a_{13} & a_{23} & a_{14} & a_{24} & a_{34} & a_{44} \end{pmatrix}^\top \tag{4.10}$$

of unknown parameters such that the *objective function*

$$\mathrm{F}\left(a_{11}, \ldots, a_{44}\right) = \sum_{i=1}^{n} \tilde{\mathrm{w}}_\mathrm{i}\, f^2\left(\mathbf{p}_i\right) + \hat{\mathrm{w}}_\mathrm{i}\, \left\|\nabla^\mathrm{x} f\left(\mathbf{p}_i\right) - \vec{\mathbf{n}}_i\right\|^2 \tag{4.11}$$

is minimized

$$\vec{a}_{\min} = \underset{a_{11},\ldots,a_{44}}{\arg\min}\, \mathrm{F}\left(a_{11}, \ldots, a_{44}\right). \tag{4.12}$$

The objective function is quadratic with respect to $\vec{a}$. In (4.11), the vector $\vec{\mathbf{n}}_i = (\hat{x}_i, \hat{y}_i, \hat{z}_i)^\top$ denotes the normal vector of $\mathcal{M}$ at vertex $\mathbf{p}_i$, while $\tilde{\mathrm{w}}_\mathrm{i}, \hat{\mathrm{w}}_\mathrm{i} > 0$ are associated real weights. We specify the weights used for testing in chapter 6.

Necessary conditions for minima of function F from equation (4.11) give

$$\nabla^\mathrm{a} \mathrm{F}\left(\vec{a}\right) = \vec{0}, \tag{4.13}$$

a system of linear equations

$$\frac{\partial \mathrm{F}}{\partial a_{ij}}\left(a_{11}, \ldots, a_{44}\right) = 0, \quad 1 \le i \le j \le 4. \tag{4.14}$$

If we denote

$$\tilde{\mathrm{F}}\left(\vec{a}\right) = \sum_{i=1}^{n} \tilde{\mathrm{w}}_\mathrm{i}\, f^2\left(\mathbf{p}_i\right), \tag{4.15}$$

$$\hat{\mathrm{F}}\left(\vec{a}\right) = \sum_{i=1}^{n} \hat{\mathrm{w}}_\mathrm{i}\, \left\|\nabla^\mathrm{x} f\left(\mathbf{p}_i\right) - \vec{\mathbf{n}}_i\right\|^2, \tag{4.16}$$

then $\mathrm{F} = \tilde{\mathrm{F}} + \hat{\mathrm{F}}$. Therefore,

$$\nabla^\mathrm{a} \mathrm{F} = \nabla^\mathrm{a} \tilde{\mathrm{F}} + \nabla^\mathrm{a} \hat{\mathrm{F}}. \tag{4.17}$$

Every vertex $\mathbf{p}_i$ contributes to the objective function in two parts. The function $\tilde{\mathrm{F}}$ measures the distance of $\mathbf{p}_i$ to the quadric $\mathcal{Q}$, weighted by $\tilde{\mathrm{w}}_\mathrm{i}$. The function $\hat{\mathrm{F}}$ measures the deviation of the computed normal from the prescribed normal (at $\mathbf{p}_i$), weighted by $\hat{\mathrm{w}}_\mathrm{i}$.

Let us have a look at the first term on the right side of (4.17). Taking a_{ij} as

variables, denote $\phi_i \coloneqq \phi(\mathbf{p}_i)$, where

$$\phi(\mathbf{p}_i) \coloneqq (\nabla^{\mathrm{a}} f)(\mathbf{p}_i) = \begin{pmatrix} {x_i}^2 & {y_i}^2 & {z_i}^2 & 2x_iy_i & 2x_iz_i & 2y_iz_i & 2x_i & 2y_i & 2z_i & 1 \end{pmatrix}^\top. \tag{4.18}$$

Using equation (4.18) and the fact that $f(\mathbf{p}_i) = \phi_i^\top \vec{a}$, we get

$$\nabla^{\mathrm{a}} \tilde{\mathrm{F}} = \sum_{i=1}^{n} \tilde{\mathrm{w}}_{\mathrm{i}}\ 2\,(\nabla^{\mathrm{a}} f)(\mathbf{p}_i)\ f(\mathbf{p}_i) = 2\sum_{i=1}^{n} \tilde{\mathrm{w}}_{\mathrm{i}}\ \phi_i\,\phi_i^\top\,\vec{a} = 2\sum_{i=1}^{n} \tilde{\mathrm{w}}_{\mathrm{i}}\ \Phi_i\,\vec{a} = 2\,\Phi\,\vec{a}. \tag{4.19}$$

Here we used the notation $\Phi_i \coloneqq \phi_i\phi_i^\top$ and $\Phi \coloneqq \sum_{i=1}^{n} \tilde{\mathrm{w}}_{\mathrm{i}}\,\Phi_i$ for corresponding matrices,

$$\Phi_i = \begin{pmatrix} x^4 & x^2y^2 & x^2z^2 & 2x^3y & 2x^3z & 2x^2yz & 2x^3 & 2x^2y & 2x^2z & x^2 \\ x^2y^2 & y^4 & y^2z^2 & 2xy^3 & 2xy^2z & 2y^3z & 2xy^2 & 2y^3 & 2y^2z & y^2 \\ x^2z^2 & y^2z^2 & z^4 & 2xyz^2 & 2xz^3 & 2yz^3 & 2xz^2 & 2yz^2 & 2z^3 & z^2 \\ 2x^3y & 2xy^3 & 2xyz^2 & 4x^2y^2 & 4x^2yz & 4xy^2z & 4x^2y & 4xy^2 & 4xyz & 2xy \\ 2x^3z & 2xy^2z & 2xz^3 & 4x^2yz & 4x^2z^2 & 4xyz^2 & 4x^2z & 4xyz & 4xz^2 & 2xz \\ 2x^2yz & 2y^3z & 2yz^3 & 4xy^2z & 4xyz^2 & 4y^2z^2 & 4xyz & 4y^2z & 4yz^2 & 2yz \\ 2x^3 & 2xy^2 & 2xz^2 & 4x^2y & 4x^2z & 4xyz & 4x^2 & 4xy & 4xz & 2x \\ 2x^2y & 2y^3 & 2yz^2 & 4xy^2 & 4xyz & 4y^2z & 4xy & 4y^2 & 4yz & 2y \\ 2x^2z & 2y^2z & 2z^3 & 4xyz & 4xz^2 & 4yz^2 & 4xz & 4yz & 4z^2 & 2z \\ x^2 & y^2 & z^2 & 2xy & 2xz & 2yz & 2x & 2y & 2z & 1 \end{pmatrix}. \tag{4.20}$$

We will now analyze the second term on the right side of (4.17). First, the gradient of f with respect to x, y, z is computed

$$\nabla^{\mathrm{x}} f(\mathbf{p}_i) = 2 \begin{pmatrix} a_{11}x_i + a_{12}y_i + a_{13}z_i + a_{14} \\ a_{12}x_i + a_{22}y_i + a_{23}z_i + a_{24} \\ a_{13}x_i + a_{23}y_i + a_{33}z_i + a_{34} \end{pmatrix} =: \begin{pmatrix} \alpha_i \\ \beta_i \\ \gamma_i \end{pmatrix}, \tag{4.21}$$

where $\alpha_i, \beta_i, \gamma_i$ are dependent on $\vec{a}$. Recall the coordinates of normal $\vec{\mathbf{n}}_i$ at the vertex $\mathbf{p}_i$ are $(\hat{x}_i, \hat{y}_i, \hat{z}_i)$. Consequently,

$$\begin{aligned} \|\nabla^{\mathrm{x}} f(\mathbf{p}_i) - \vec{\mathbf{n}}_i\|^2 &= (\alpha_i - \hat{x}_i)^2 + (\beta_i - \hat{y}_i)^2 + (\gamma_i - \hat{z}_i)^2 \\ &= [\,2\,(a_{11}x_i + a_{12}y_i + a_{13}z_i + a_{14}) - \hat{x}_i\,]^2 + \\ &+ [\,2\,(a_{12}x_i + a_{22}y_i + a_{23}z_i + a_{24}) - \hat{y}_i\,]^2 + \\ &+ [\,2\,(a_{13}x_i + a_{23}y_i + a_{33}z_i + a_{34}) - \hat{z}_i\,]^2. \end{aligned} \tag{4.22}$$

Applying the gradient operator ∇^{a} on (4.22), we have

$$\begin{aligned}\nabla^{\mathrm{a}}\left(\|\nabla^{\mathrm{x}} f\left(\mathbf{p}_i\right)-\vec{\mathbf{n}}_i\|^2\right)&=\nabla^{\mathrm{a}}\left((\alpha_i-\hat{x}_i)^2+(\beta_i-\hat{y}_i)^2+(\gamma_i-\hat{z}_i)^2\right)=\\&=2\left(\alpha_i-\hat{x}_i\right)\nabla^{\mathrm{a}}\alpha_i+2\left(\beta_i-\hat{y}_i\right)\nabla^{\mathrm{a}}\beta_i+2\left(\gamma_i-\hat{z}_i\right)\nabla^{\mathrm{a}}\gamma_i.\end{aligned}\tag{4.23}$$

Note that applying ∇^{a} on α_i, β_i and γ_i, we get the vectors

$$\begin{aligned}\nabla^{\mathrm{a}}\alpha_i&=2\begin{pmatrix}x_i & 0 & 0 & y_i & z_i & 0 & 1 & 0 & 0 & 0\end{pmatrix}^\top,\\\nabla^{\mathrm{a}}\beta_i&=2\begin{pmatrix}0 & y_i & 0 & x_i & 0 & z_i & 0 & 1 & 0 & 0\end{pmatrix}^\top,\\\nabla^{\mathrm{a}}\gamma_i&=2\begin{pmatrix}0 & 0 & z_i & 0 & x_i & y_i & 0 & 0 & 1 & 0\end{pmatrix}^\top.\end{aligned}\tag{4.24}$$

Each of these vectors has only four non-zero coordinates. Plugging (4.24) into (4.23) and substituting into (4.16) yields

$$\nabla^{\mathrm{a}}\hat{\mathrm{F}}\left(\vec{a}\right)=\sum_{i=1}^{n}4\ \hat{\mathrm{w}}_{\mathrm{i}}\begin{pmatrix}x_i\ (\alpha_i-\hat{x}_i)\\y_i\ (\beta_i-\hat{y}_i)\\z_i\ (\gamma_i-\hat{z}_i)\\(\alpha_i-\hat{x}_i)\,y_i+(\beta_i-\hat{y}_i)\,x_i\\(\alpha_i-\hat{x}_i)\,z_i+(\gamma_i-\hat{z}_i)\,x_i\\(\beta_i-\hat{y}_i)\,z_i+(\gamma_i-\hat{z}_i)\,y_i\\\alpha_i-\hat{x}_i\\\beta_i-\hat{y}_i\\\gamma_i-\hat{z}_i\\0\end{pmatrix}=4\sum_{i=1}^{n}\hat{\mathrm{w}}_{\mathrm{i}}\left(2\,\Psi_i\,\vec{a}-\Omega_i\right),\tag{4.25}$$

where

$$\Psi_i\coloneqq\begin{pmatrix}x_i^2 & 0 & 0 & x_i\,y_i & x_i\,z_i & 0 & x_i & 0 & 0 & 0\\0 & y_i^2 & 0 & x_i\,y_i & 0 & y_i\,z_i & 0 & y_i & 0 & 0\\0 & 0 & z_i^2 & 0 & x_i\,z_i & y_i\,z_i & 0 & 0 & z_i & 0\\x_i\,y_i & x_i\,y_i & 0 & x_i^2+y_i^2 & y_i\,z_i & x_i\,z_i & y_i & x_i & 0 & 0\\x_i\,z_i & 0 & x_i\,z_i & y_i\,z_i & x_i^2+z_i^2 & x_i\,y_i & z_i & 0 & x_i & 0\\0 & y_i\,z_i & y_i\,z_i & x_i\,z_i & x_i\,y_i & y_i^2+z_i^2 & 0 & z_i & y_i & 0\\x_i & 0 & 0 & y_i & z_i & 0 & 1 & 0 & 0 & 0\\0 & y_i & 0 & x_i & 0 & z_i & 0 & 1 & 0 & 0\\0 & 0 & z_i & 0 & x_i & y_i & 0 & 0 & 1 & 0\\0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{pmatrix}\tag{4.26}$$

and

$$\Omega_i \coloneqq \begin{pmatrix} x_i\hat{x}_i & y_i\hat{y}_i & z_i\hat{z}_i & x_i\hat{y}_i + y_i\hat{x}_i & x_i\hat{z}_i + z_i\hat{x}_i & y_i\hat{z}_i + z_i\hat{y}_i & \hat{x}_i & \hat{y}_i & \hat{z}_i & 0 \end{pmatrix}^\top. \quad (4.27)$$

Introducing the notation

$$\Psi \coloneqq \sum_{i=1}^{n} \hat{\mathrm{w}}_\mathrm{i}\ \Psi_i,\ \Omega \coloneqq \sum_{i=1}^{n} \hat{\mathrm{w}}_\mathrm{i}\ \Omega_i, \quad (4.28)$$

equation (4.25) becomes

$$\nabla^\mathrm{a} \hat{\mathrm{F}}\,(\vec{a}) = 8\Psi\,\vec{a} - 4\,\Omega. \quad (4.29)$$

Using the equations (4.13), (4.19) and (4.29), we obtain the desired system of linear equations in matrix form

$$0 = \nabla^\mathrm{a} \mathrm{F}\,(\vec{a}) = 2\,\Phi\,\vec{a} + 8\Psi\,\vec{a} - 4\,\Omega = (2\Phi + 8\Psi)\,\vec{a} - 4\,\Omega = \Gamma\vec{a} - 4\Omega \quad (4.30)$$

with its explicit solution

$$\vec{a}_{\min} = 4\,\Gamma^{-1}\,\Omega, \quad (4.31)$$

provided $\Gamma = 2\Phi + 8\Psi$ is an invertible matrix.

The conditions of invertibility of matrix Γ are not clear yet. In our experiments, the problems arise in linear case, that is when all vertices $\mathbf{p}_i \in \mathcal{N}_\mathrm{T}$ lie in a plane. We plan to find a more exact description of such cases.

4.4 Picking the new vertex

After the quadric $\mathcal{Q}$ has been found by solving the system (4.30), we are able to compute the coordinates of the new vertex $\mathbf{v}$. Denote

$$\mathbf{b}_T \coloneqq \frac{\mathbf{v}_0 + \mathbf{v}_1 + \mathbf{v}_2}{3} \quad (4.32)$$

to be the barycenter of the triangle T. Now we introduce two methods for picking the new vertex. Both methods are visualised on figure 4.3.

4.4.1 Intersection of normal line and quadric

Our first approach is to find the position of $\mathbf{v}$ as the intersection of quadric $\mathcal{Q}$ and the normal line $\bar{n}$ of T, defined parametrically as

$$\bar{n} \equiv \mathbf{b}_T + t\vec{\mathbf{n}}_T\ ,\ t \in \mathbb{R}, \quad (4.33)$$

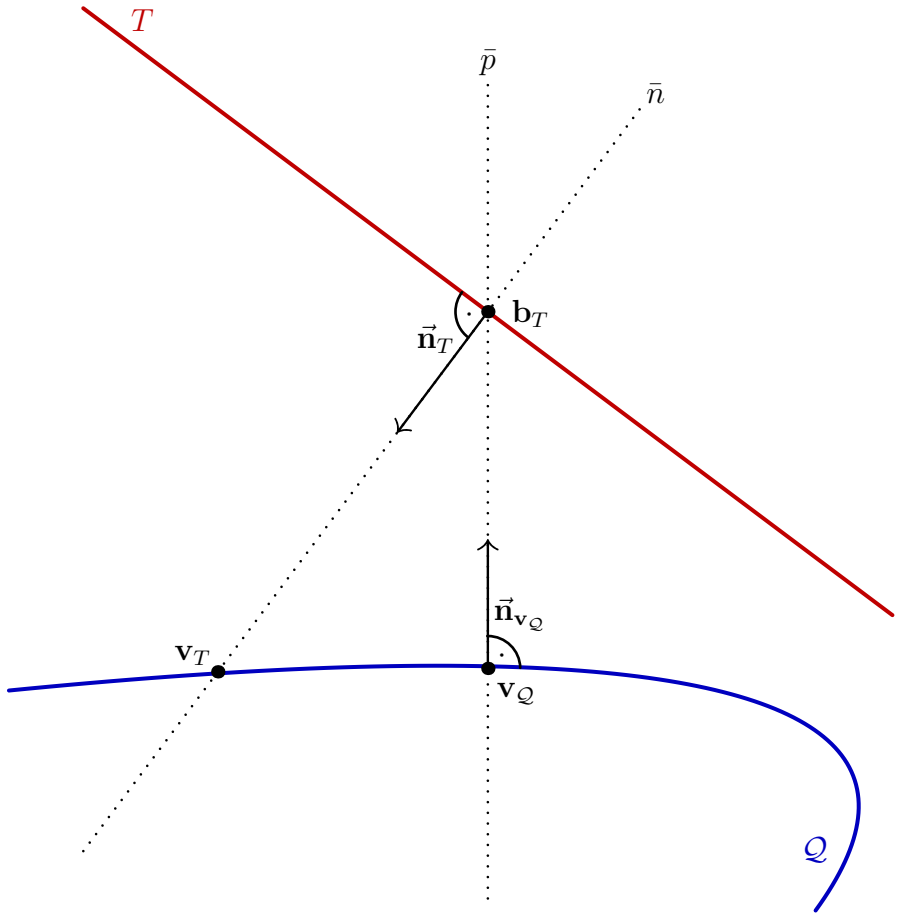


Figure 4.3: Schematic comparison of the two approaches for picking the new vertex, associated with triangle T from the quadric $\mathcal{Q}$. The technique described in section 4.4.1 yields $\mathbf{v}_T$, while the technique from 4.4.2 yields $\mathbf{v}_\mathcal{Q}$.

where $\vec{\mathbf{n}}_T$ is the normal vector of triangle T. The vector $\vec{\mathbf{n}}_T$ is defined as the unit normal of the plane determined by the vertices of T (up to signum), see figure 4.3. Denote the intersection of $\mathcal{Q}$ and $\bar{n}$, if it exists, as

$$\mathbf{v}_T = \mathbf{b}_T + t_0 \, \vec{\mathbf{n}}_T, \tag{4.34}$$

or, coordinate-wise,

$$\begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} + t_0 \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} \tag{4.35}$$

for some t_0. Plugging (4.34) into (4.4), (4.5), we get

$$a_{11}x_v^2 + a_{22}y_v^2 + \cdots + a_{34}z_v + a_{44} = 0\,, \tag{4.36}$$

which leads to the quadratic equation in t_0 of the form

$$At_0^2 + 2Bt_0 + C = 0\,, \tag{4.37}$$

where the coefficients $A, B, C \in \mathbb{R}$ are

$$\begin{aligned} A = {} & x_n\left(a_{11}x_n + a_{12}y_n + a_{13}z_n\right) + \\ & y_n\left(a_{12}x_n + a_{22}y_n + a_{23}z_n\right) + \\ & z_n\left(a_{13}x_n + a_{23}y_n + a_{33}z_n\right), \end{aligned} \tag{4.38}$$

$$\begin{aligned} B = {} & x_n\left(a_{11}x_b + a_{12}y_b + a_{13}z_b + a_{14}\right) + \\ & y_n\left(a_{12}x_b + a_{22}y_b + a_{23}z_b + a_{24}\right) + \\ & z_n\left(a_{13}x_b + a_{23}y_b + a_{33}z_b + a_{34}\right), \end{aligned} \tag{4.39}$$

$$\begin{aligned} C = {} & x_b\left(a_{11}x_b + a_{12}y_b + a_{13}z_b + a_{14}\right) + \\ & y_b\left(a_{12}x_b + a_{22}y_b + a_{23}z_b + a_{24}\right) + \\ & z_b\left(a_{13}x_b + a_{23}y_b + a_{33}z_b + a_{34}\right) + a_{44} \\ = {} & f\left(x_b, y_b, z_b\right). \end{aligned} \tag{4.40}$$

The roots of the quadratic equation (4.37) are

$$t_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{4.41}$$

and the parameter t_0 is chosen as

$$t_0 = \begin{cases} 0, & \text{if } B^2 - 4AC < 0\,; \\ t_1, & \text{if } |t_1| < |t_2|\,; \\ t_2, & \text{otherwise.} \end{cases} \tag{4.42}$$

If the parameters t_1, t_2 are real, they determine two points on $\bar{n}$. We pick the point which is closer to $\mathbf{b}_T$. If t_1, t_2 are complex and the intersection point does not exist, the barycenter $\mathbf{b}_T$ is picked as the new point.

4.4.2 Foot point of barycenter

Although the procedure described in section 4.4.1 is easy to implement, it does not generate optimal choice of the new vertex. In some cases, the intersection of $\mathcal{Q}$ and $\bar{n}$ does not exist or it is relatively distant from the mesh. The distant vertices create unwanted local sharp spikes. Moreover, if the initial mesh is not closed, the spikes occur around the boundary.

These issues are resolved by picking the new vertex as a foot point of perpendicular

line $\bar{p}$ from $\mathbf{b}_T$ onto $\mathcal{Q}$, see figure 4.3. To find the foot point numerically, we use algorithm 4.2 taken from [Har99, section 5.1.2], which is a proper adaptation of the Newton's method.

The input of the algorithm is the function f, defining implicit surface $\mathcal{S} \equiv f(\mathbf{x}) = 0$, along with some point $\mathbf{p}$ in the vicinity of $\mathcal{S}$. Procedure FootPoint iteratively finds the foot point of $\mathbf{p}$ on $\mathcal{S}$. In our case,

$$\mathbf{v}_{\mathcal{Q}} = \textsc{FootPoint}(\mathbf{b}_T). \tag{4.43}$$

procedure SurfacePoint($\mathbf{p}$)
  $\mathbf{q}_0 = \mathbf{p}$;
  do
    $\mathbf{q}_{i+1} = \mathbf{q}_i - \dfrac{f(\mathbf{q}_i)}{\|\nabla f(\mathbf{q}_i)\|^2} f(\mathbf{q}_i)$
  while $\|\mathbf{q}_{i+1} - \mathbf{q}_i\| > \epsilon$;
  return $\mathbf{q}_{i+1}$;

Algorithm 4.1: Procedure for calculating the surface point

procedure FootPoint($\mathbf{p}$)
  $\mathbf{p}_0 = \textsc{SurfacePoint}(\mathbf{p})$;
  do
    $\mathbf{q}_i = \mathbf{p} - \dfrac{(\mathbf{p} - \mathbf{p}_i) \cdot \nabla f(\mathbf{p}_i)}{\|\nabla f(\mathbf{p}_i)\|^2} \nabla f(\mathbf{p}_i)$;
    $\mathbf{p}_{i+1} = \textsc{SurfacePoint}(\mathbf{q}_i)$;
    $\mathbf{f}_1 \coloneqq \mathbf{q}_i - \mathbf{p}_i$; $\mathbf{f}_2 \coloneqq \mathbf{p}_{i+1} - \mathbf{q}_i$;
    if $\|\mathbf{q}_i - \mathbf{p}_i\| > \epsilon$ **then**
      $a_0 \coloneqq (\mathbf{p} - \mathbf{p}_i) \cdot \mathbf{f}_1$;
      $a_1 \coloneqq 2\mathbf{f}_2 \cdot (\mathbf{p} - \mathbf{p}_i) - \|\mathbf{f}_1\|^2$;
      $a_2 \coloneqq -3\mathbf{f}_1 \cdot \mathbf{f}_2$;
      $a_3 \coloneqq -\|\mathbf{f}_2\|^2$;
      $\alpha \coloneqq 1 - \dfrac{a_0 + a_1 + a_2 + a_3}{a_1 + 2a_2 + 3a_3}$;
      if $0 < \alpha < \alpha_{\max}$ **then**
        $\mathbf{q}_i = \mathbf{p}_i + \alpha\mathbf{f}_1 + \alpha^2\mathbf{f}_2$;
        $\mathbf{p}_{i+1} = \textsc{SurfacePoint}(\mathbf{q}_i)$;
  while $\|\mathbf{p}_{i+1} - \mathbf{p}_i\| > \epsilon$;
  return $\mathbf{p}_{i+1}$;

Algorithm 4.2: Foot point algorithm for implicit surface $f(\mathbf{x}) = 0$ from [Har99]. Procedure SurfacePoint is defined in algorithm 4.1.

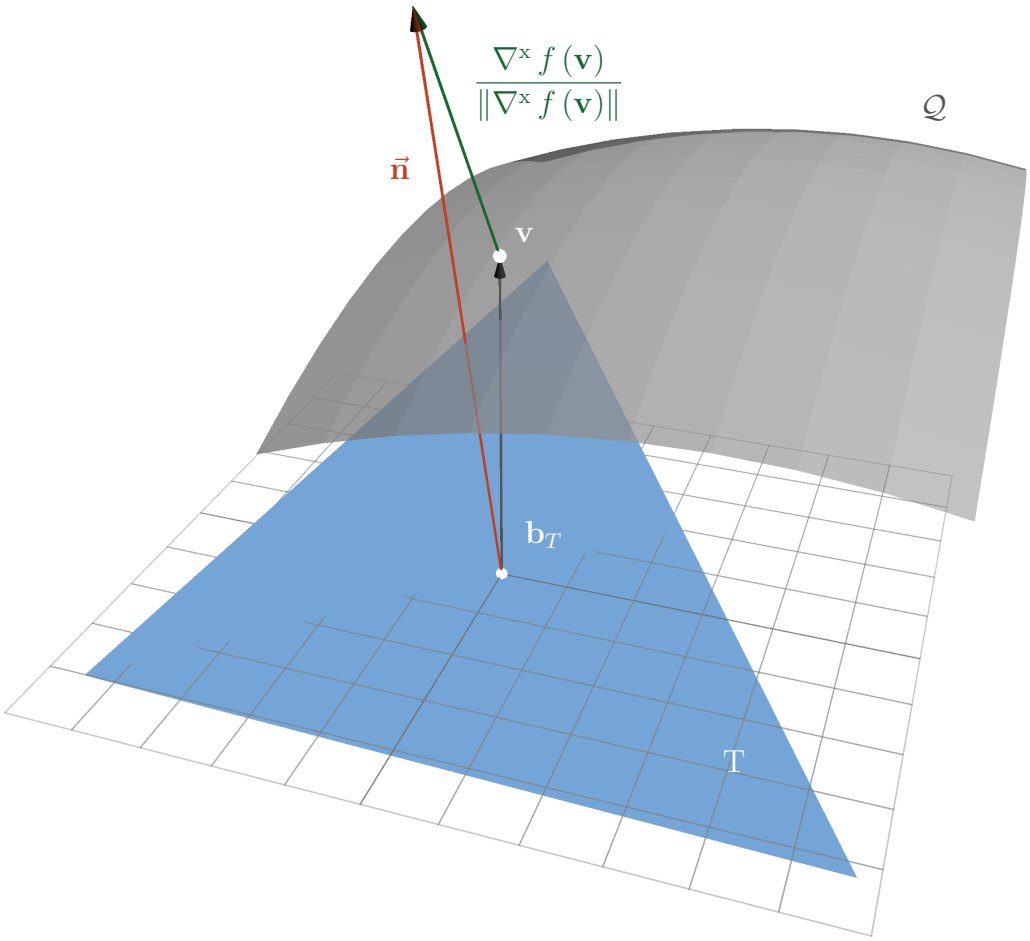


Figure 4.4: Choosing the normal vector $\vec{\mathbf{n}}$ at the newly introduced vertex $\mathbf{v}$.

4.5 Choosing the normal vector

The last step of our algorithm is the choice of the normal vector $\vec{\mathbf{n}}$ at newly inserted vertex $\mathbf{v}$. This is done by computing the normal vector of the quadric $\mathcal{Q}$ at $\mathbf{v}$ as a normalized gradient of f and adding the correction term $(\mathbf{v} - \mathbf{b}_T)$,

$$\vec{\mathbf{n}} = \frac{\nabla^{\mathrm{x}} f(\mathbf{v})}{\|\nabla^{\mathrm{x}} f(\mathbf{v})\|} + (\mathbf{v} - \mathbf{b}_T)\,. \tag{4.44}$$

We then check if the angle between $\vec{\mathbf{n}}$ and the face normal $\vec{\mathbf{n}}_T$ of T is less than or equal to 90 degrees. If the opposite is true, meaning

$$\vec{\mathbf{n}} \cdot \vec{\mathbf{n}}_T < 0, \tag{4.45}$$

the normal vector at $\mathbf{v}$ is taken as the opposite of $\vec{\mathbf{n}}$.

The resulting vector does not need to be normalized as our algorithm does not require unit vectors. Our implementation allows the user to specify whether he requires the normalization. The two approaches produce different results in general. In our tests, we have used strictly normalized vectors.

4.6 Computing the weights

In our current setup, the weights $\tilde{w}_i, \hat{w}_i$ in the objective function (4.11) need to be adjusted case-by-case. One of the possible future improvements of the QFR is to compute the weights algorithmically. The local geometry of the mesh (vertex angles, triangle areas) could be used for this purpose.

Chapter 5

Implementation

The implementation of our method was done in C++ and compiled under GCC 4.8.1. Both techniques for picking the new vertex from the quadric were implemented. All of the results in chapter 6 were obtained using the foot point algorithm exclusively, see section 4.4.2 and algorithm 4.2.

5.1 OpenMesh

For the mesh manipulation, we have decided to use an open-source C++ library *OpenMesh* [BSBK02]. OpenMesh is developed by the working group of Leif Kobbelt at RWTH Aachen University. We chose OpenMesh for several reasons.

First, OpenMesh represents a mesh via the halfedge data structure, also known as the *doubly connected edge list*. DCEL allows fast performance of mesh operations, such as iterating over all faces, iterating over all neighbours of a vertex, splitting a face, flipping an edge. These operations are also implemented in OpenMesh and can be performed by simply calling the corresponding function, see code excerpt 5.1.

```
Mesh mesh;
Mesh::VertexHandle vertex;
Mesh::EdgeHandle edge;
Mesh::FaceHandle face;

mesh.split( face, vertex );
mesh.flip( edge );
```

Code Excerpt 5.1: With OpenMesh, it is fairly easy to split the `face` at the given `vertex`, or just flip the selected `edge`.

Second, the OpenMesh library includes the application *Subdivider*, with built-in framework for subdivision surfaces. By overloading abstract base class `SubdividerT`, Subdivider implements various linear subdivision schemes such as the Loop scheme, the $\sqrt{3}$-subdivision and the Modified Butterfly.

The existence of subdivision framework in Subdivider facilitated the implementation of the quadric fitting refinement greatly. At the same time, the framework provided full control over the implementation with no limitations. The functionality of Subdivider consists of four basic operations:

1. Load a mesh from the file.
2. Refine the mesh using the selected refinement scheme.
3. Save the mesh to a file.
4. Reset the refined mesh into its initial state.

Loading and saving of the mesh supports various popular mesh data formats, such as Wavefront OBJ (.obj), Stanford Polygon File Format (.ply) and Object File Format (.off). This functionality is handled by the `OpenMesh::IO` namespace.

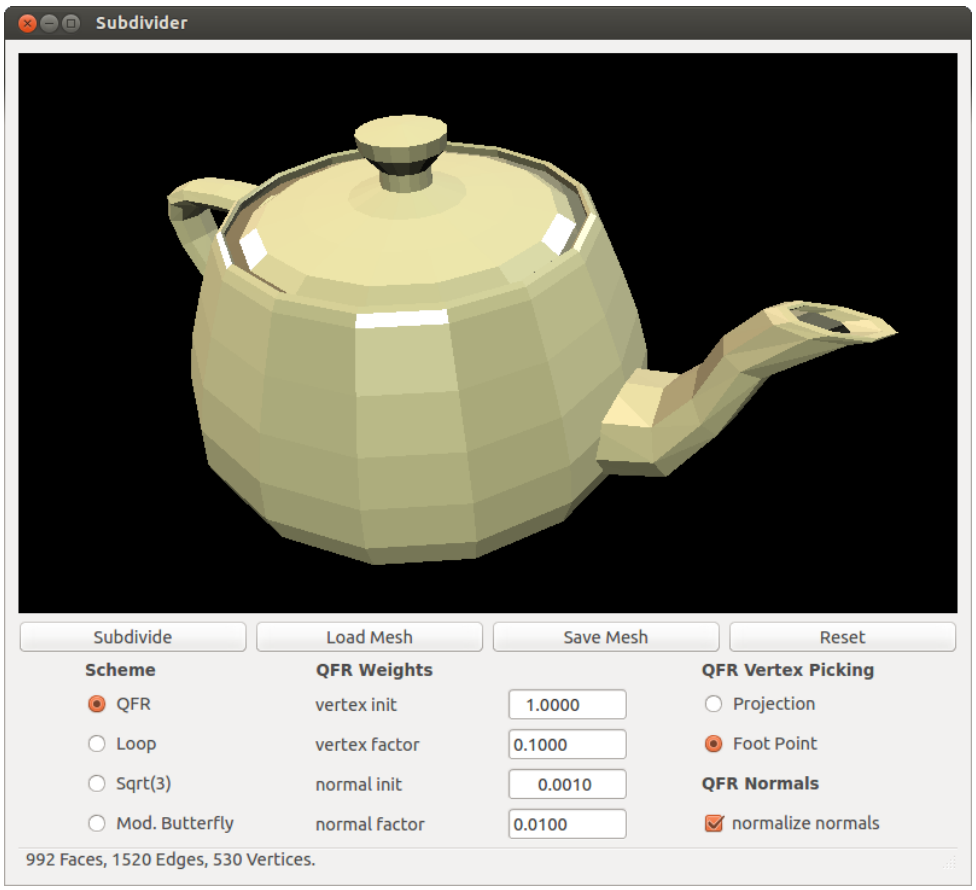


Figure 5.1: The Utah teapot in the application *Subdivider*, with the implementation of quadric fitting refinement.

The graphical user interface in Subdivider is provided by the Qt library, meshes are rendered using the OpenGL Utility Toolkit (GLUT), see figure 5.1. The four functions mentioned above are displayed as four buttons, the refinement methods are displayed as radio buttons.

When refining with QFR, the user can specify the weights $\tilde{w}_i, \hat{w}_i$ to be used in the minimization of the objective function, see equations (4.11), (4.19) and (4.25). The input of weights is implemented as a set of input text fields. The input is constrained to meet the specified numerical pattern. This validation prevents the application crashes, resulting from the input of malformed weights.

OpenMesh provides the set of iterators and circulators to simplify the navigation around the mesh. The iterators allow iterating over all faces, edges, halfedges and vertices of the mesh. The circulators are designed for enumerating the elements adjacent to the so-called central element. For instance, they provide means for enumerating all vertices adjacent to the face F, all faces adjacent to the face F, all faces adjacent to vertex V and so on. Iterators and circulators are implemented in the `OpenMesh::Iterators` namespace and the `OpenMesh::PolyConnectivity` class. Example of their use for getting the vertices in the face neighbourhood is shown in the code excerpt 5.2.

```
Mesh mesh;
Mesh::FaceHandle face;
std::set< Mesh::VertexHandle > face_neighbours;
std::set< Mesh::VertexHandle >::iterator v_it;

// first, insert the face vertices into the set of neighbours
Mesh::FaceVertexIter fv_it = mesh.fv_iter( face );
for( fv_it; fv_it.is_valid(); ++fv_it )
    face_neighbours->insert( *fv_it );

// loop until the set of neighbours is large enough
while( face_neighbours->size() < 9 ) {
    v_it = face_neighbours->begin();
    for( v_it; v_it != face_neighbours->end(); ++v_it ) {
        Mesh::VertexVertexIter vv_it = mesh.vv_iter( *v_it );
        for ( vv_it; vv_it.is_valid(); ++vv_it )
                face_neighbours->insert( *vv_it );
}}
```

Code Excerpt 5.2: Computing the set of face neighbourhood using iterators in OpenMesh. Here, the face–vertex circulator (`FaceVertIter`) and the vertex–vertex circulator (`VertVertIter`) are used. The former iterates over all vertices adjacent to `face` , the latter iterates over all vertices adjacent to vertex `*v_it` .

5.2 Armadillo

To solve the system (4.30), we used open-source C++ linear algebra library *Armadillo* [San10]. Armadillo uses simple Matlab-like syntax for matrix input. Solving the system of linear equations is very straightforward too, see code excerpt 5.3.

```
// specify the matrix
arma::mat Psi_i;
Psi_i
<< xx << 0  << 0  << xy    << xz    << 0     << x << 0 << 0 << 0 << arma::endr
<< 0  << yy << 0  << xy    << 0     << yz    << 0 << y << 0 << 0 << arma::endr
<< 0  << 0  << zz << 0     << xz    << yz    << 0 << 0 << z << 0 << arma::endr
<< xy << xy << 0  << xx+yy << yz    << xz    << y << x << 0 << 0 << arma::endr
<< xz << 0  << xz << yz    << xx+zz << xy    << z << 0 << x << 0 << arma::endr
<< 0  << yz << yz << xz    << xy    << yy+zz << 0 << z << y << 0 << arma::endr
<< x  << 0  << 0  << y     << z     << 0     << 1 << 0 << 0 << 0 << arma::endr
<< 0  << y  << 0  << x     << 0     << z     << 0 << 1 << 0 << 0 << arma::endr
<< 0  << 0  << z  << 0     << x     << y     << 0 << 0 << 1 << 0 << arma::endr
<< 0  << 0  << 0  << 0     << 0     << 0     << 0 << 0 << 0 << 0 << arma::endr;
// solve the system
arma::vec solution = arma::solve(Gamma, Omega);
```

Code Excerpt 5.3: Input of matrix Ψ_i from (4.26) and computation of the solution to the linear system (4.30) in Armadillo.

5.3 Time complexity

Computational complexity of the quadric fitting refinement is $\mathcal{O}(F)$, where F is the number of processed faces. Figure 5.2 shows the graph of the relation between time needed for one iteration of our algorithm and the number of triangles in the refined mesh. All measurements were performed on the PC with Intel Core i7 3517 Ivy Bridge processor running Ubuntu 13.10 Saucy Salamander.

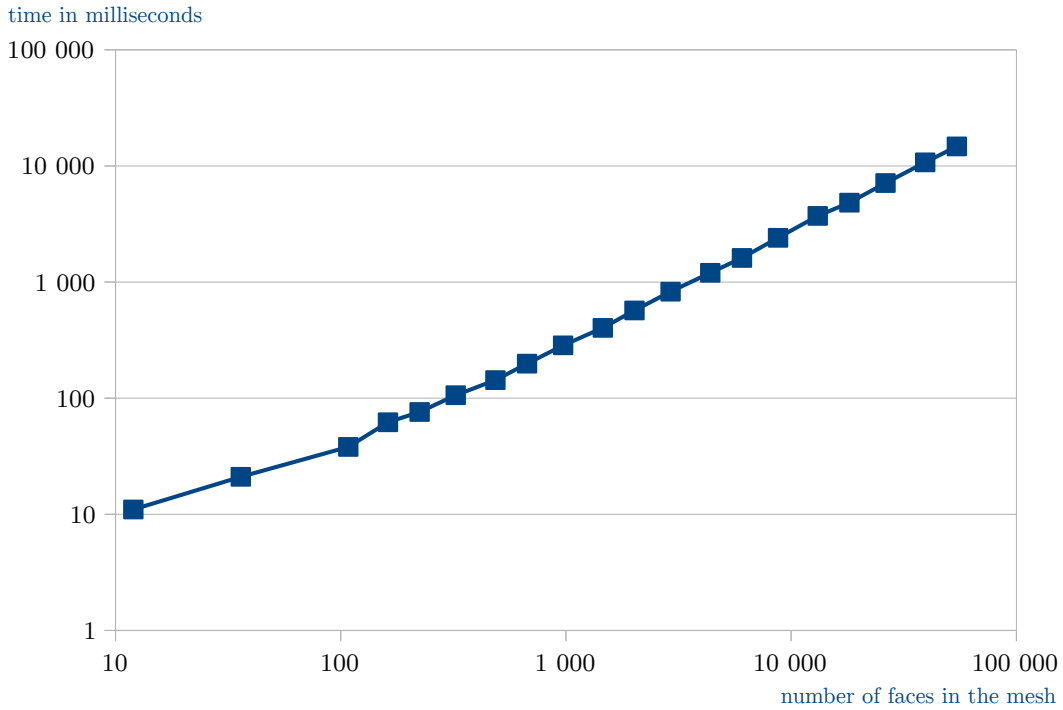


Figure 5.2: Experimental measurements of the linear time complexity of our algorithm.

Chapter 6

Results

The properties of the quadric fitting refinement were tested from multiple points of view. In our experiments, three major questions were addressed:

1. *What is the best choice of the weights* $\tilde{w}_i, \hat{w}_i$ *for given input mesh* $\mathcal{M}$*?*
2. *Given the decimation* $\mathcal{M}_D$ *of the mesh* $\mathcal{M}$*, how well is* $\mathcal{M}$ *approximated after* $\mathcal{M}_D$*is refined* k *times?*
3. *How good is the scheme's ability to reconstruct quadratic surfaces?*

In this chapter, we demonstrate the experimental answers to these questions.

6.1 Testing the scheme

6.1.1 Input meshes

Various meshes were used as an input for the presented algorithm. Here, we attempted to choose the meshes that would provide a proper demonstration of various properties.

The effect of use of different weights in our algorithm is demonstrated on the well-known Stanford bunny mesh. The results are summarized in the section 6.2. The bunny mesh was also used to demonstrate the influence of the normal vectors on the limit surface.

One group of meshes we worked with are the detailed meshes of the *Venus of Dolní Věstonice* (figure 6.1) and of the *bear-shaped animal* (figure 6.2). These meshes are the digitalized versions of the small statuettes found in Moravia south of Brno. Dated to 29,000 – 25,000 BCE, they are considered one of the oldest known pieces of ceramic in the world.

In order to test the scheme's ability to reconstruct arbitrary object from its coarse approximation, the detailed Venus and bear meshes were decimated using quadric

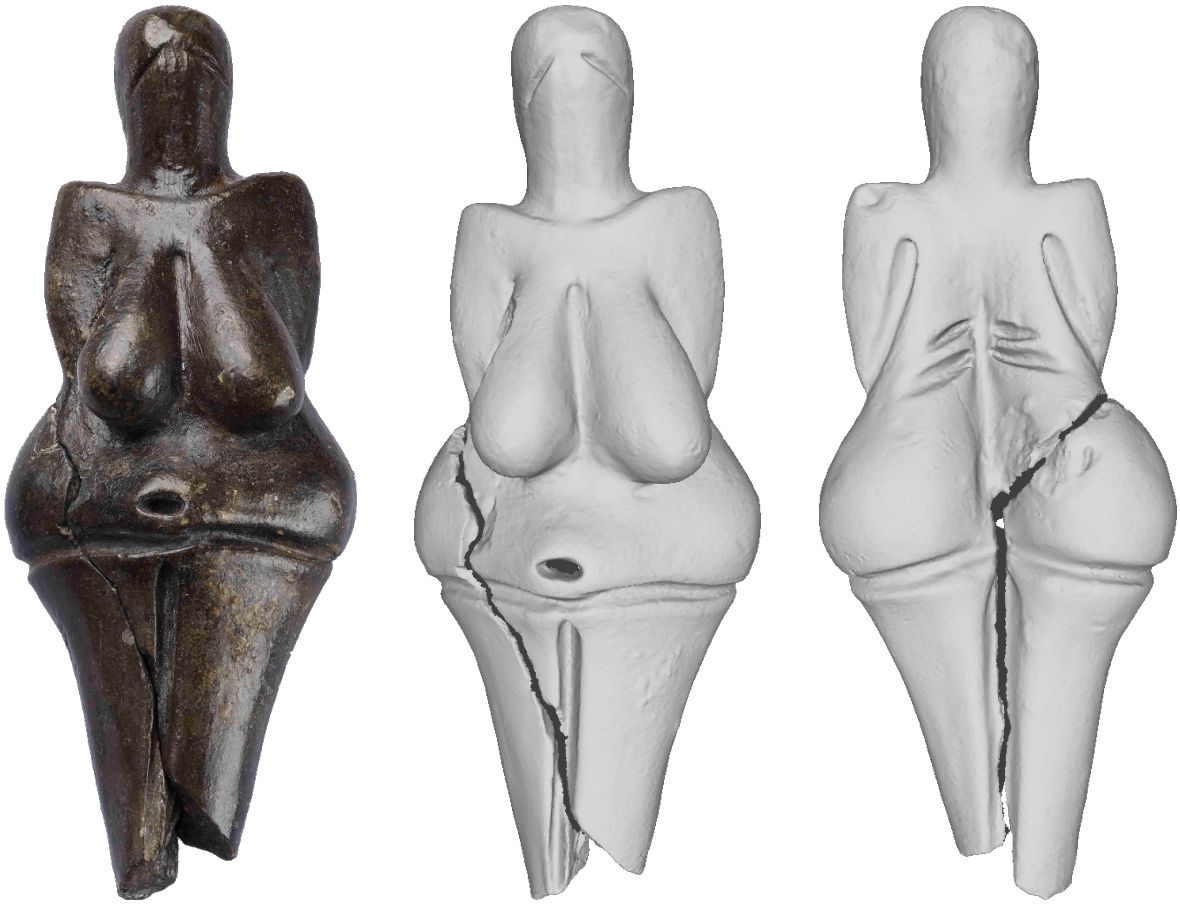


Figure 6.1: Venus of Dolní Věstonice, photo of the original statuette (courtesy of [Arc13]) and two renderings of the laser-scanned mesh. The mesh has 131 114 vertices, 391 596 edges and 260 482 faces.

based edge collapse strategy. For the decimation method details, see [GH97]. We shall remark the vertices of decimated mesh form a subset of vertices of the original mesh.

The two meshses were decimated with approximately 99% compression rate. The original Venus mesh has 131 114 vertices, its decimation has 1 356 vertices. The original Bear mesh has 119 519 vertices, its decimation has 1 202 vertices.

6.1.2 Note on the weights

So far, we have not specified the real weights from the objective function F, see equation (4.11). When computing the weights $\tilde{w}_i, \hat{w}_i$, we usually consider the graph distance $\mathcal{D}_T(\mathbf{p}_i) =: \mathcal{D}_T^i$ of the vertex $\mathbf{p}_i \in \mathcal{N}_T$ from the triangle T, see equation (4.2). Moreover, the weights are taken as functions, exponential in the additive inverse of $\mathcal{D}_T$.

Often in this chapter, the weights $\tilde{w}_i, \hat{w}_i$ are specified by the pairs $(v_i, v_f), (n_i, n_f)$. The scalars $v_i, n_i \in \mathbb{R}$ are called the *initial values*, the scalars $v_f, n_f \in (0, 1]$ form the bases of the exponential functions and are called the *factors*. Given the initial values

Figure 6.2: Bear statuette from the archaeological site in Dolní Věstonice, photo of the original statuette (courtesy of [Hit14]), the laser-scanned mesh and its decimation. The original mesh has 119 519 vertices, 358 551 edges and 239 034 faces.

and factors, the weights for $\mathbf{p}_i$ are computed as

$$\tilde{w}_i = v_i \, v_f^{\mathcal{D}_T^i}, \tag{6.1a}$$

$$\hat{w}_i = n_i \, n_f^{\mathcal{D}_T^i}. \tag{6.1b}$$

This means the weights of the vertices from the triangle T are equal to initial values v_i, n_i, the weights of vertices next to triangle T are smaller by factor v_f, n_f and so on. The values of the initial values and the factors need to be adjusted experimentally for a specific case. However, we give some analysis on how to choose the right weights in section 6.2.

6.1.3 Measuring the error

To compare refined models, the one-sided Hausdorff distance of the vertices of the original mesh $\mathcal{M}$ from the refined mesh $\bar{\mathcal{M}}$ is used as an error function. If $\mathbf{v} \in \mathcal{V}(\mathcal{M})$ and d is the Euclidean distance, then

$$H(\mathbf{v}) = \min_{\bar{\mathbf{v}} \in \bar{\mathcal{M}}} d(\mathbf{v}, \bar{\mathbf{v}}) \tag{6.2}$$

is the one-sided Hausdorff distance of vertex $\mathbf{v}$ from the refined mesh. Technically, H is not a metric; it does not satisfy the metric axioms. (For instance, it is not symmetric.) However, such one-sided distance is effective for estimating the error between refined and original mesh in practical cases. The error function H is evaluated for each vertex of the original mesh $\mathcal{M}$ to determine

- **maximal error**: $e_{\max} = \max_{\mathbf{v}\in\mathcal{V}(\mathcal{M})} H(\mathbf{v})$,
- **mean error**: $e_{\text{mean}} = \frac{1}{|\mathcal{V}(\mathcal{M})|} \sum_{\mathbf{v}\in\mathcal{V}(\mathcal{M})} H(\mathbf{v})$,
- **root mean square (RMS) error**: $e_{\text{RMS}} = \sqrt{\frac{1}{|\mathcal{V}(\mathcal{M})|} \sum_{\mathbf{v}\in\mathcal{V}(\mathcal{M})} H^2(\mathbf{v})}$.

All measurements in this chapter are in mesh units. For reference, the lengths of the sides of the bounding box of the Venus mesh are $108.4, 31.8$, and 42.8 units, the diagonal of the bounding box is 120.8 units long. The bounding box lengths for the bear mesh are $67.3, 43.2$ and 45.7 units, the diagonal is 89.2 units long. For error computation and visualisation, we have used the software *MeshLab*.

6.2 Choosing the weights

Theoretically, any positive real number can be used as a weight $\tilde{w}_i$ of the vertex $\mathbf{p}_i$ or as a weight $\hat{w}_i$ of the normal $\vec{\mathbf{n}}_i$. In practice, the weights have to be chosen carefully as their choice influences the result significantly.

We demonstrate the effect of different sets of weights on the Stanford bunny, see figure 6.3. The bunny mesh was decimated to 3000 faces and refined using QFR with the initial values and factors

$$v_i = 1, \quad v_f = 1, \quad n_i = 1, \quad n_f = 1; \tag{6.3a}$$
$$v_i = 1000, \quad v_f = 1, \quad n_i = 0.0001, \quad n_f = 0.0001; \tag{6.3b}$$
$$v_i = 1000, \quad v_f = 0.0001, \quad n_i = 1, \quad n_f = 1; \tag{6.3c}$$
$$v_i = 1000, \quad v_f = 0.0001, \quad n_i = 0.0001, \quad n_f = 0.0001. \tag{6.3d}$$

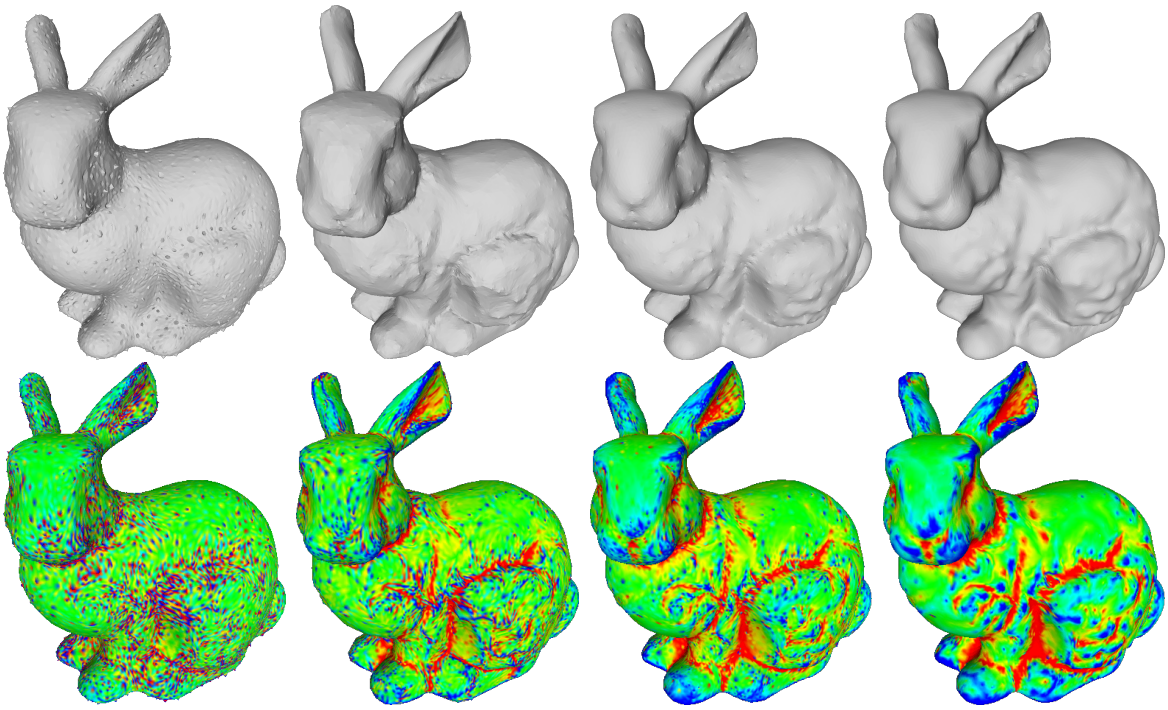
Top row of figure 6.3 shows the decimated bunny mesh after three iterations of QFR with the above sets of weights. Bottom row shows the visualisation of the discrete mean curvature of the refined meshes, which is the discrete version of the *mean curvature*

$$\kappa_H = \frac{1}{2}(\kappa_1 + \kappa_2), \tag{6.4}$$

where κ_1 and κ_2 are the *principal curvatures*, see [HLS93]. For the definition of the discrete mean curvature, see [MDSB03].

Here, we are not interested in the particular values corresponding to individual colors. More importantly, we study the variation in the discrete curvature. For this purpose, it is sufficient to know the red color corresponds to positive mean curvature, values close to zero are coloured green, while the blue color maps to negative values.

This example allows us to observe various interesting facts concerning the weights in QFR. It is clear the strategy of taking all data with the same weights – as in (a) – does not produce fine results for an irregular mesh such as the Stanford bunny. This is



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

Figure 6.3: The Stanford bunny refined with four different sets of weights from equations (6.3a-d). Bottom part shows the visualisation of the discrete mean curvature.

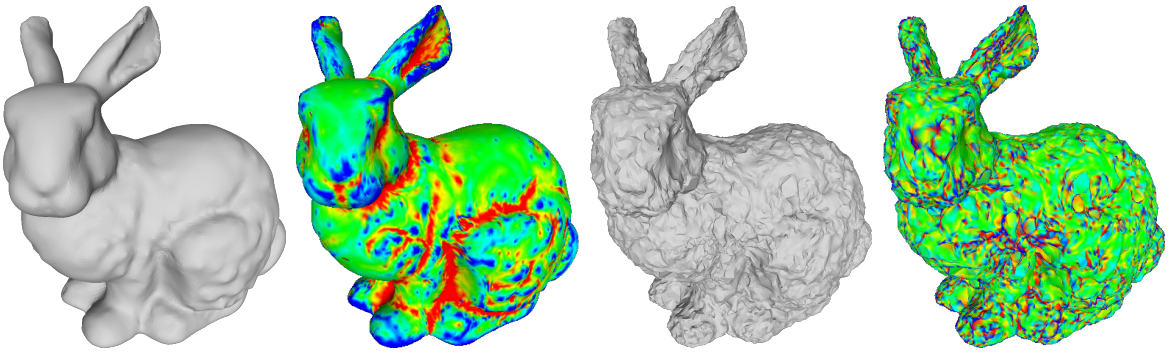
due to the fact that the information carried by normal vectors is very strong and has to be treated carefully.

Assigning the normals the smaller weights (b-d) yields much realistic result. The best results are obtained in (d), where the vertices have much higher weights than the normals (order of 10^7). Also, the more distant vertices and normals have smaller weights than the close ones (order of 10^4). However, using the weights that are independent of the distance (when $v_f = n_f = 1$) can be useful in some cases, as we see in section 6.4.

To show how the normals at mesh vertices influence the resulting limit surface, we refined two bunny meshes with the same vertex positions but different normal vectors. The bunny at figure 6.4a has the original normals, for the one at 6.4c the normals were generated randomly. The weights from equation (6.3d) were used for both meshes. The results differ dramatically, see figure 6.4 even though the weights of the normal vectors are much smaller than the weights of the vertices. The strong difference in the results implies the importance of the role played by the normal vectors.

6.3 Comparison with linear schemes

To understand how well the quadric fitting refinement performs next to linear triangular schemes, both approximating and interpolating, we refined the decimations of Venus and bear four times. Besides the quadric fitting refinement, the tested methods were



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

Figure 6.4: The Stanford bunny with original vertex normals (a,b) and randomly generated vertex normals (c,d), refined using the QFR with the weights from (6.3d). The visualisation of discrete mean curvature is shown.

the $\sqrt{3}$-subdivision, the Modified butterfly and the Loop scheme.

For the QFR, we have used the weights $(v_i, v_f) = (1, 0.1), (n_i, n_f) = (0.001, 0.01)$. Again, we have used the one-sided Hausdorff distance to measure the error and compare the refined meshes. The results are visualised on figure 6.5, the numerical results are summarized in tables 6.1, 6.2, 6.3 and 6.4.

The performance of the quadric fitting refinement is comparable to the performance of the Modified Butterfly. This is related to the fact that both QFR and Butterfly are interpolating schemes. However, the meshes produced by our method are visually smoother. The meshes generated by the approximating schemes ($\sqrt{3}$-subdivision, Loop) are also smooth, but they lack the details of the mesh produced by the QFR.

Our setup can be used for the compression of the large-scale meshes. Using only a fraction of the original vertices, the quadric fitting refinement is able to produce a close approximation of the original dense mesh while preserving some details.

scheme	**error**					
	max.		**mean**		**RMS**	
	Venus	bear	Venus	bear	Venus	bear
QFR	1.944741	0.524010	0.092007	0.047816	0.171047	0.070868
$\sqrt{3}$	1.944741	0.552654	0.150250	0.068184	0.210626	0.090188
MB	1.874044	0.490356	0.083599	0.047714	0.164202	0.066050
Loop	1.987397	0.580256	0.157413	0.071771	0.217968	0.094510

Table 6.1: Comparison of QFR with linear schemes, 1st iteration.

scheme	**error**					
	max.		**mean**		**RMS**	
	Venus	bear	Venus	bear	Venus	bear
QFR	1.944741	0.648266	0.092112	0.051493	0.173042	0.077019
$\sqrt{3}$	1.989867	0.581215	0.166759	0.078788	0.226488	0.103254
MB	1.845731	0.500278	0.082776	0.053625	0.163477	0.072219
Loop	2.001070	0.583309	0.170478	0.081101	0.229893	0.106173

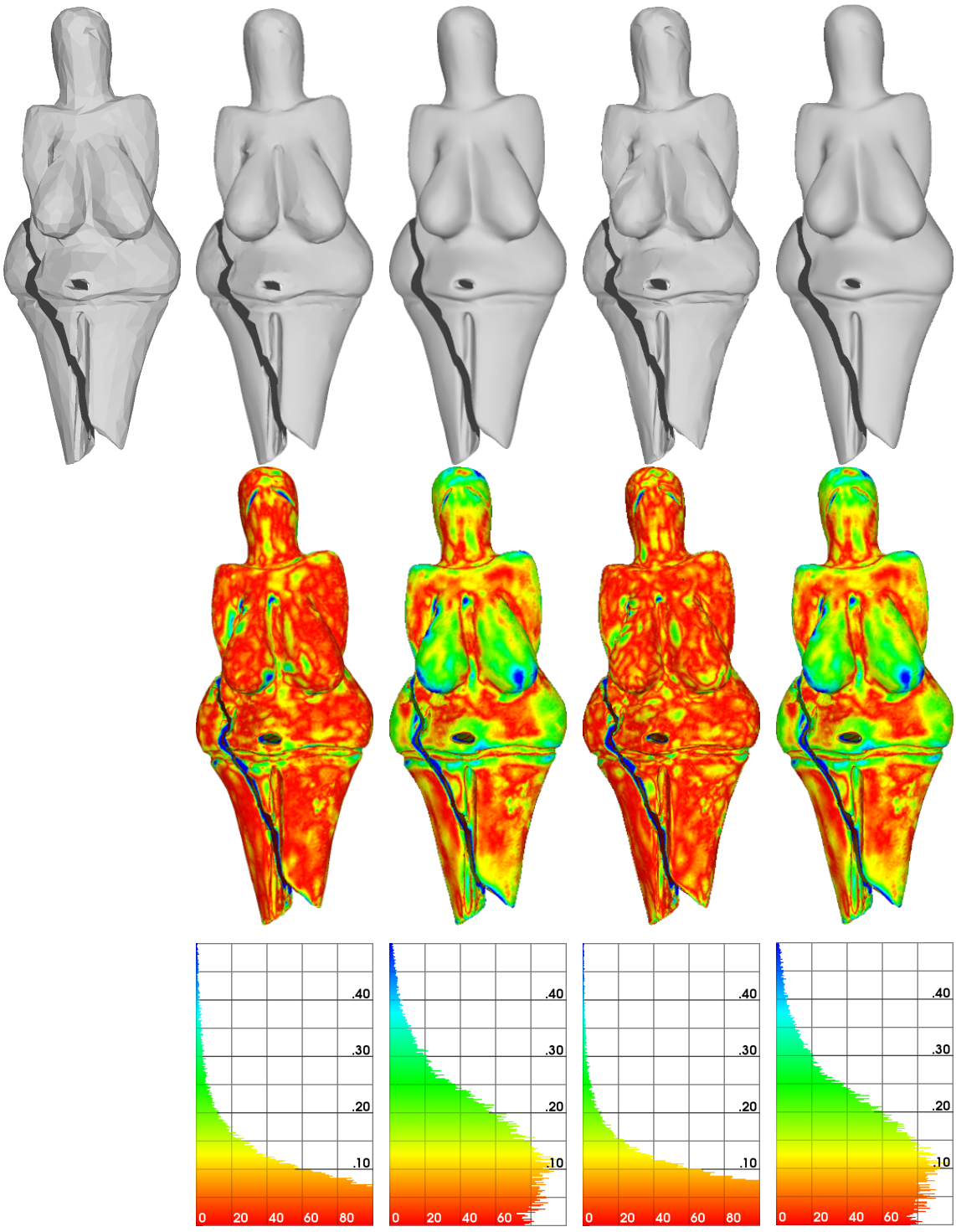
Table 6.2: Comparison of QFR with linear schemes, 2nd iteration.

scheme	**error**					
	max.		**mean**		**RMS**	
	Venus	bear	Venus	bear	Venus	bear
QFR	1.944741	0.670785	0.092703	0.053033	0.173825	0.079792
$\sqrt{3}$	1.989867	0.590496	0.171830	0.082465	0.230977	0.107827
MB	1.840539	0.503913	0.083315	0.055693	0.163611	0.074660
Loop	2.001096	0.587618	0.173752	0.083461	0.232948	0.109120

Table 6.3: Comparison of QFR with linear schemes, 3rd iteration.

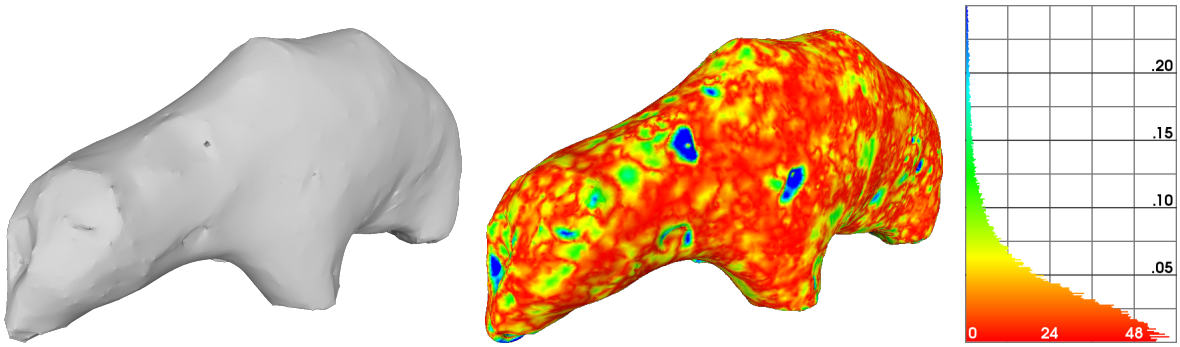
scheme	**error**					
	max.		**mean**		**RMS**	
	Venus	bear	Venus	bear	Venus	bear
QFR	1.944741	0.705566	0.092874	0.053628	0.174013	0.080872
$\sqrt{3}$	2.002892	0.589712	0.173668	0.083702	0.232844	0.109367
MB	1.839134	0.505339	0.083516	0.056289	0.163672	0.075373
Loop	2.002858	0.588562	0.174568	0.084052	0.233709	0.109860

Table 6.4: Comparison of QFR with linear schemes, 4th iteration.

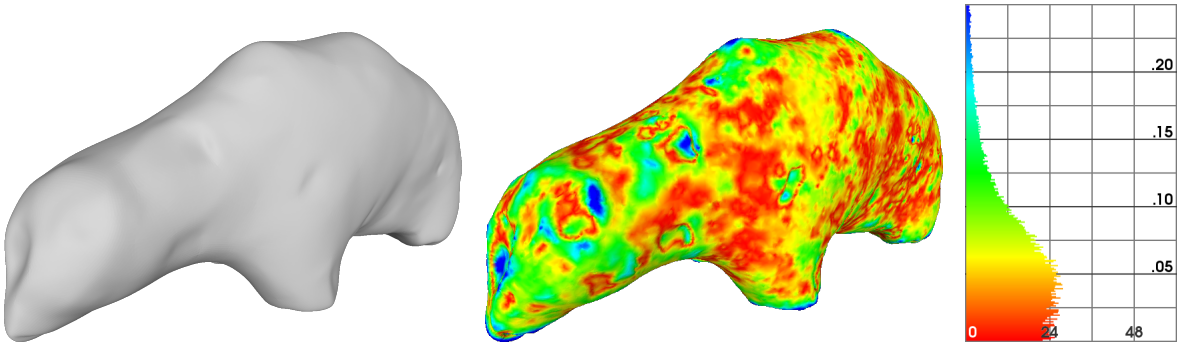


(a) (b) (c) (d) (e)

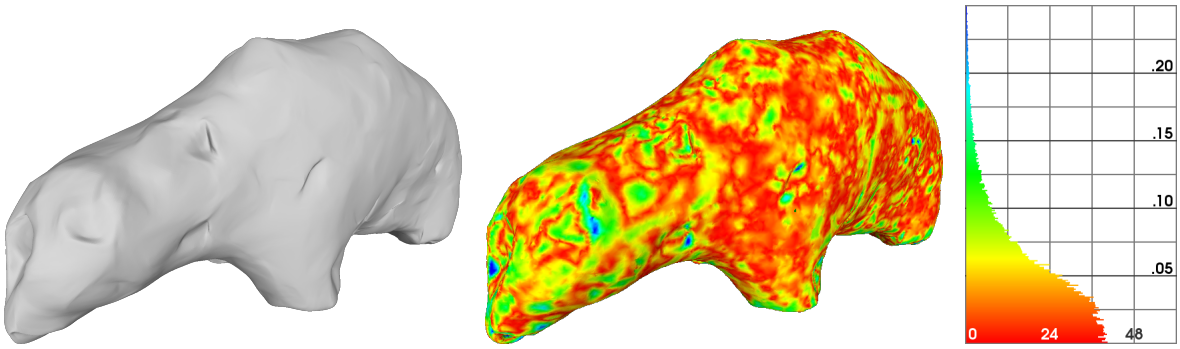
Figure 6.5: Comparison of our method with the linear triangular schemes on the Venus mesh. (a) Decimated Venus mesh, (b-e) decimated mesh refined with the QFR, the $\sqrt{3}$-subdivision, the Modified Butterfly and the Loop scheme. Top row in (b-e) shows the mesh after four iterations of given scheme, middle row shows the visualisation of the Hausdorff distance of the original mesh from the refined meshes. Bottom row displays the histograms of the Hausdorff distance.



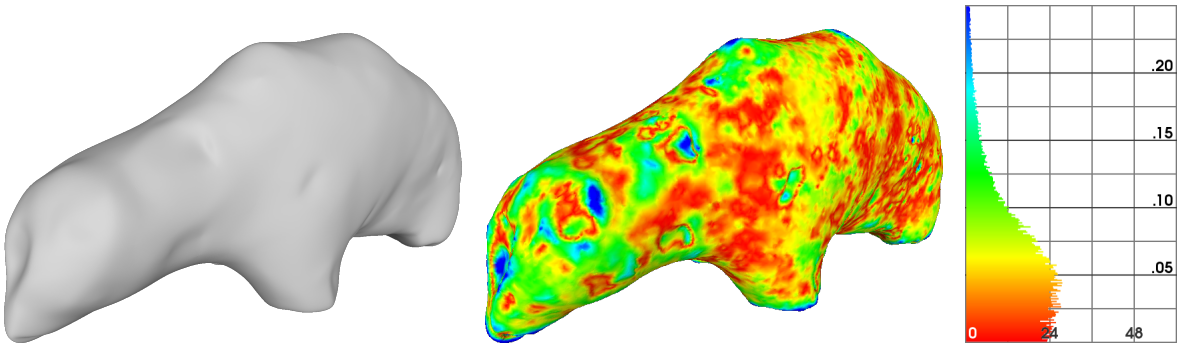
(a)



(b)



(c)



(d)

Figure 6.6: Comparison of our method with the linear triangular schemes on the bear mesh. (a-d) decimated mesh refined with the QFR, the $\sqrt{3}$-subdivision, the Modified Butterfly and the Loop scheme. Left column shows the mesh after four iterations of given scheme, middle column shows the visualisation of the Hausdorff distance of the original mesh from the refined meshes. Right column displays the histograms of the Hausdorff distance.

6.4 Reconstruction of quadratic surfaces

In the context of our refinement method, quadratic surfaces or *quadrics* are an important tool. Here, we show how the can resulting scheme be used to reconstruct quadrics from a coarse approximating mesh. Our idea was that given the nature of quadric fitting method, with the right choice of weights, the scheme should be able to produce good approximation of quadric surface after a few iterations. Obviously, there is a limit to that idea; if we take too coarse mesh on the input, the error of the refined mesh will naturally be greater.

For the purpose of testing, we use the bounded subset of quadric, which is defined via implicit equation. Namely, we have experimented with

$$\textbf{sphere} \qquad x^2 + y^2 + z^2 = 1, \tag{6.5a}$$

$$\textbf{elliptic paraboloid} \qquad x^2 + y^2 + z = 1, \tag{6.5b}$$

$$\textbf{hyperbolic paraboloid} \qquad x^2 - y^2 - z = 0, \tag{6.5c}$$

$$\textbf{cylinder} \qquad x^2 + y^2 = 1. \tag{6.5d}$$

Due to the strong symmetry in the input meshes and desired limit surfaces, we decided to use *uniform* weights. Such weights are independent of the distance of the corresponding vertex from the subdivided triangle. More precisely, we have used $\tilde{w}_i = 1000, \hat{w}_i = 0.0001$.

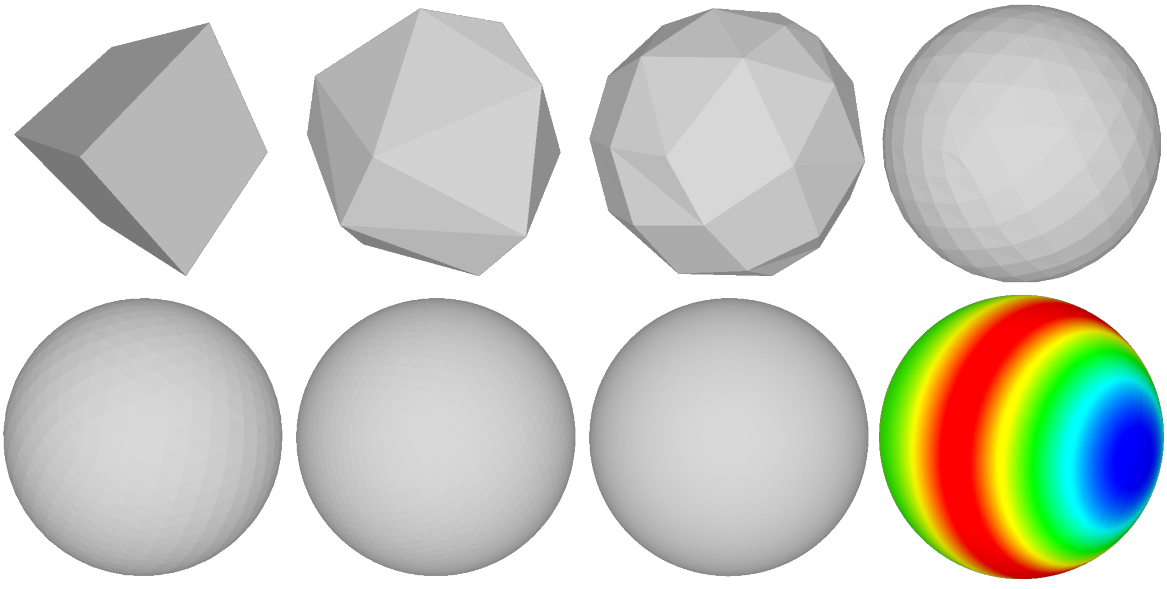


Figure 6.7: Reconstruction of the sphere $x^2 + y^2 + z^2 = 1$.

Using the given weights, the scheme is capable of reconstructing a close approximation of the sphere from the cube, see figure 6.7. The initial mesh (cube) is shown after $0, 1, 2, 3$ and 9 iterations, together with the color visualisation of distance of the densest mesh from the sphere. The red color corresponds to zero distance, blue color corre-

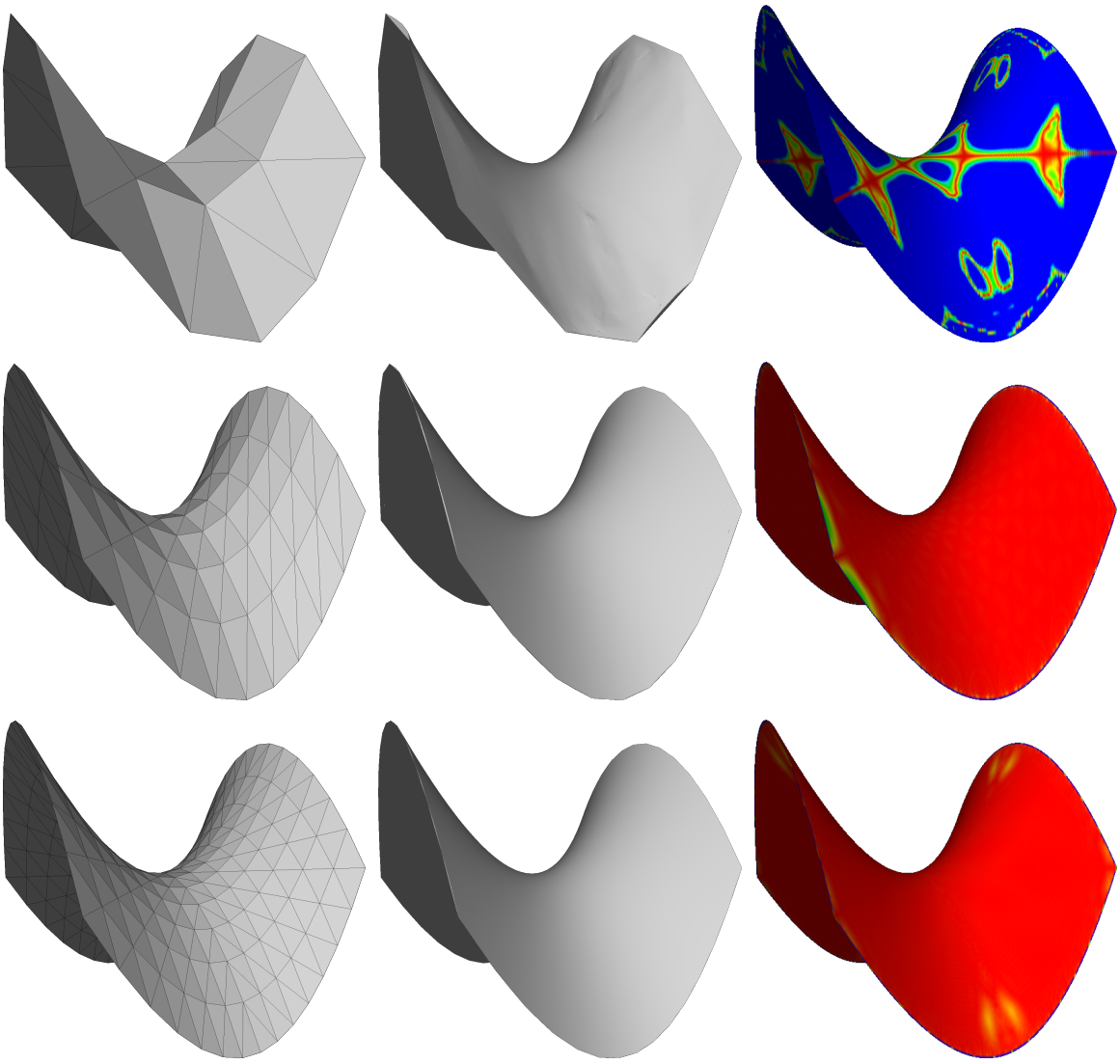


Figure 6.8: Reconstruction of the hyperbolic paraboloid $x^2 - y^2 - z = 0$.

sponds to distance ≥ 0.0025. The output is also influenced by the initial triangulation of the cube.

The other surfaces exhibit good behaviour as well, see figure 6.8 for the hyperbolic paraboloid, figure 6.9 for the elliptic paraboloid and figure 6.10 for the cylinder. First column of these figures shows the initial meshes which were obtained from the corresponding quadratic surface sampled to $25, 100$ and 225 vertices. Second column is the initial mesh after a few iterations of the QFR and the last column is the color mapping of the error on the original surface. Here, the blue color corresponds to distance ≥ 0.001.

Each initial mesh with 25 vertices was refined seven times, a mesh with 100 vertices was refined six times and a mesh with 225 vertices was refined five times. The reason why the numbers of the iterations differ is that we wanted to compare meshes with roughly the same amount of vertices.

It appears the coarsest approximations with only 25 vertices are not sufficient to re-

construct the surface appropriately. However, the error of the approximation decreases as the number of vertices in the initial mesh grows. The numerical values of Hausdorff distance from the original quadratic surfaces are summarized in table 6.5. Table 6.6 shows the values relative to the coarsest initial mesh, table 6.7 shows the values relative to the corresponding initial mesh and table 6.8 shows the values relative to the coarsest (initial or refined) mesh.

It is clear from table 6.6 the error of each initial mesh is linearly dependent on the density (number of vertices) of an initial mesh. This means the error of an initial mesh with 100 vertices is roughly four times smaller than the error of an initial mesh with 25 vertices; similarly for an initial mesh with 225 vertices. This does not hold for the refined meshes as can be observed in table 6.8. Although in some cases (paraboloids) the relative error of the refined mesh is smaller than the relative error of the initial mesh, other times it gets greater (cylinder).

Overally, the scheme's capabilities to reconstruct the tested surfaces are significant. The obtained experimental results allow us to make a hypothesis that QFR actually *reproduces* quadratic surfaces. In the future, we want to study this hypothesis from the analytic point of view.

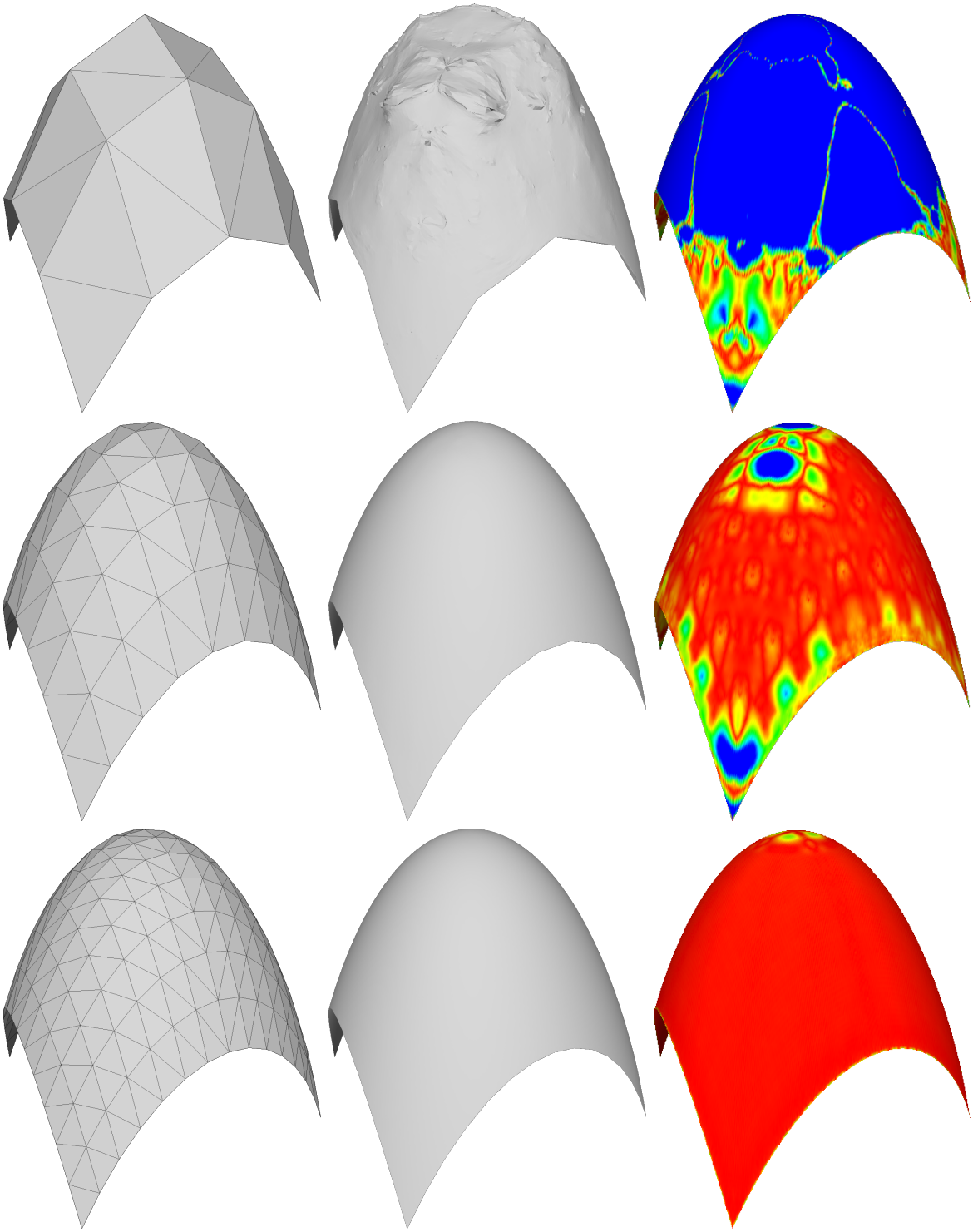


Figure 6.9: Reconstruction of the elliptic paraboloid $x^2 + y^2 + z = 1$.

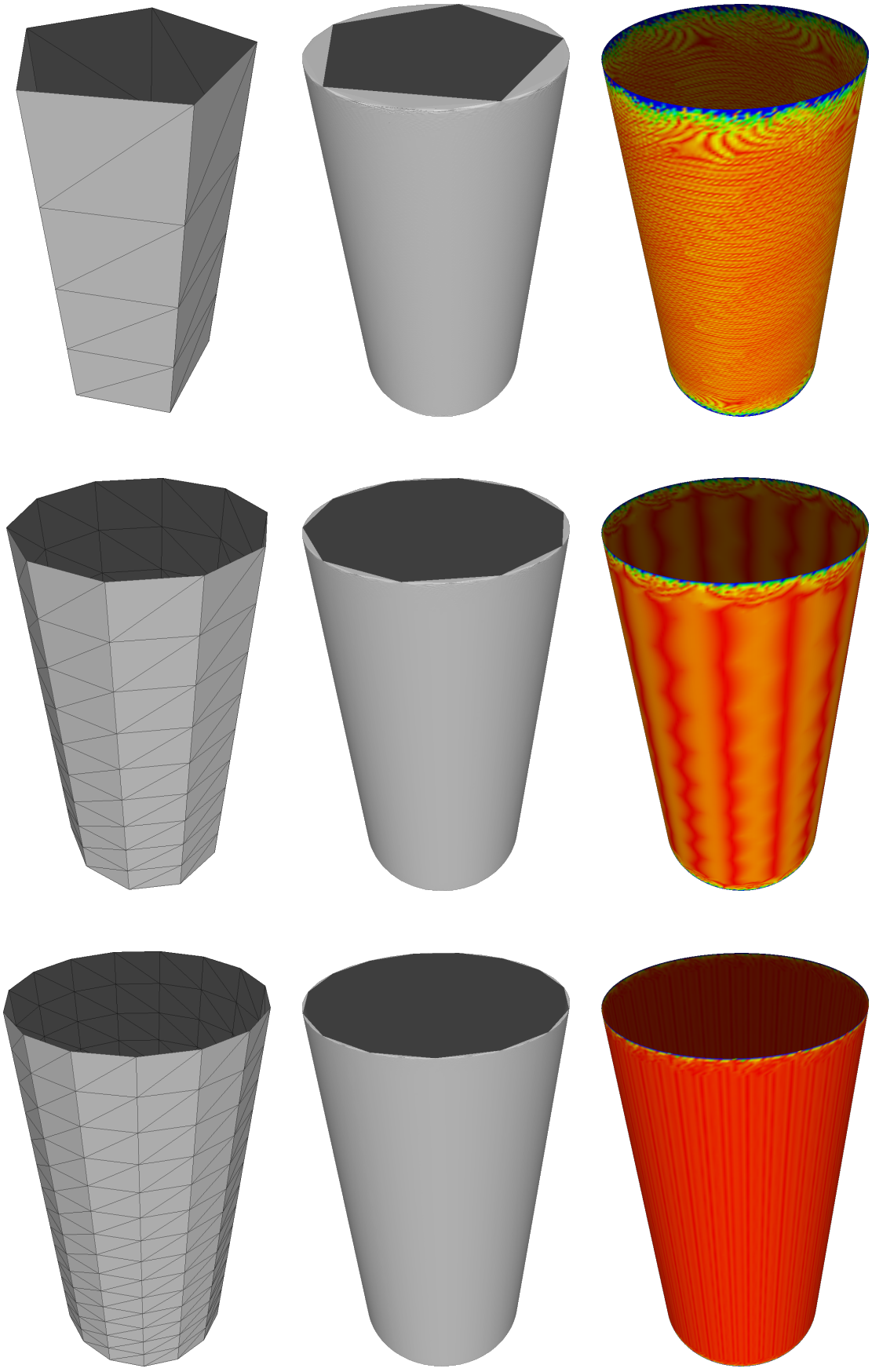


Figure 6.10: Reconstruction of the cylinder $x^2 + y^2 = 1$.

surface	**# of vertices** in the initial mesh	**iter.**	**absolute error**		
			max.	**mean**	**RMS**
sphere	8	0	0.173076	0.123827	0.129736
		1	0.161774	0.109749	0.115202
		2	0.064192	0.040658	0.042899
		3	0.022978	0.013905	0.014680
		4	0.009419	0.005125	0.005555
		5	0.004838	0.002164	0.002609
		6	0.003343	0.001461	0.001810
		7	0.002847	0.001350	0.001604
		8	0.002683	0.001322	0.001547
		9	0.002630	0.001313	0.001530
hyperbolic paraboloid	25	0	0.055894	0.012171	0.015882
		7	0.052361	0.003809	0.006712
	100	0	0.012331	0.002400	0.003154
		6	0.011568	0.000089	0.000747
	225	0	0.005041	0.000993	0.001306
		5	0.005041	0.000046	0.000341
elliptic paraboloid	25	0	0.102048	0.048118	0.052024
		7	0.088869	0.015077	0.023006
	100	0	0.024660	0.009409	0.010224
		6	0.002871	0.000111	0.000291
	225	0	0.009984	0.003882	0.004221
		5	0.000392	0.000023	0.000047
cylinder	25	0	0.190982	0.126526	0.138846
		7	0.009681	0.000125	0.000579
	100	0	0.048944	0.032588	0.035710
		6	0.004184	0.000074	0.000307
	225	0	0.021852	0.014559	0.015954
		5	0.004589	0.000057	0.000225

Table 6.5: Summary of the numerical results from the reconstruction of the quadratic surfaces using the QFR.

surface	**# of vertices**		**iter.**	**relative error (in %)**		
	init.	refined		**max.**	**mean**	**RMS**
sphere	8	8	0	100.00	100.00	100.00
		20	1	93.47	88.63	88.80
		56	2	37.09	32.83	33.07
		164	3	13.28	11.23	11.32
		488	4	5.44	4.14	4.28
		1 460	5	2.80	1.75	2.01
		4 376	6	1.93	1.18	1.40
		13 124	7	1.64	1.09	1.24
		39 368	8	1.55	1.07	1.19
		118 100	9	1.52	1.06	1.18
hyperbolic paraboloid	25	25	0	100.00	100.00	100.00
		35 001	7	93.68	31.30	42.26
	100	100	0	22.06	19.72	19.86
		59 068	6	20.70	0.73	4.70
	225	225	0	9.09	8.16	8.22
		47 657	5	9.09	0.38	2.15
elliptic paraboloid	25	25	0	100.00	100.00	100.00
		35 001	7	87.09	31.33	44.22
	100	100	0	24.17	19.55	19.65
		59 068	6	2.81	0.23	0.56
	225	225	0	9.78	8.07	8.11
		47 657	5	0.38	0.05	0.09
cylinder	25	25	0	100.00	100.00	100.00
		43 745	7	5.07	0.10	0.42
	100	100	0	25.63	25.76	25.72
		65 620	6	2.19	0.06	0.22
	225	225	0	11.44	11.51	11.49
		51 045	5	2.40	0.05	0.16

Table 6.6: Reconstruction of the quadratic surfaces, numerical data from table 6.5 relative to the coarsest initial mesh of each quadratic surface.

surface	**# of vertices**		**iter.**	**relative error (in %)**		
	init.	refined		**max.**	**mean**	**RMS**
hyperbolic paraboloid	25	25	0	100.00	100.00	100.00
		35 001	7	93.68	31.30	42.26
	100	100	0	100.00	100.00	100.00
		59 068	6	93.81	3.71	23.68
	225	225	0	100.00	100.00	100.00
		47 657	5	100.00	4.63	26.11
elliptic paraboloid	25	25	0	100.00	100.00	100.00
		35 001	7	87.09	31.33	44.22
	100	100	0	100.00	100.00	100.00
		59 068	6	11.64	1.18	2.85
	225	225	0	100.00	100.00	100.00
		47 657	5	3.93	0.59	1.11
cylinder	25	25	0	100.00	100.00	100.00
		43 745	7	5.07	0.10	0.42
	100	100	0	100.00	100.00	100.00
		65 620	6	8.55	0.23	0.86
	225	225	0	100.00	100.00	100.00
		51 045	5	21.00	0.39	1.41

Table 6.7: Reconstruction of the quadratic surfaces, numerical data from table 6.5 relative to the corresponding initial mesh.

surface	**# of vertices**		**iter.**	**relative error (in %)**		
	init.	refined		**max.**	**mean**	**RMS**
hyperbolic paraboloid	25	25	0	100.00	100.00	100.00
		35 001	7	100.00	100.00	100.00
	100	100	0	22.06	19.72	19.86
		59 068	6	22.09	2.34	11.13
	225	225	0	9.09	8.16	8.22
		47 657	5	9.63	1.21	5.08
elliptic paraboloid	25	25	0	100.00	100.00	100.00
		35 001	7	100.00	100.00	100.00
	100	100	0	24.17	19.55	19.65
		59 068	6	3.23	0.73	1.26
	225	225	0	9.78	8.07	8.11
		47 657	5	0.44	0.15	0.20
cylinder	25	25	0	100.00	100.00	100.00
		43 745	7	100.00	100.00	100.00
	100	100	0	25.63	25.76	25.72
		65 620	6	43.22	43.22	53.02
	225	225	0	11.44	11.51	11.49
		51 045	5	47.40	47.40	38.86

Table 6.8: Reconstruction of the quadratic surfaces, numerical data from table 6.5 relative to the coarsest mesh. The values for the initial meshes are relative to the coarsest initial mesh, the values for the refined meshes are relative to the coarsest refined mesh.

Chapter 7

Conclusions

We have introduced a new nonlinear algorithm for refinement of triangular mesh. The scheme was successfully applied to the compression of a large mesh. An experimental demonstration of the scheme's ability to reconstruct quadratic surfaces was given.

The future work on the scheme can be divided in two parts. One part is the rigorous analysis of the scheme; the other part is the improvement of the scheme itself.

The main problem we left open is the smoothness of the limit surface, produced by the quadric fitting refinement. In the future, we want to prove the scheme is actually $\mathcal{G}^1$ smooth as implied by our experimental results.

The experiments have also shown a good behaviour when it comes to the reconstruction of the quadratic surfaces. Further investigation concerning our hypothesis from section 6.4 about the reproduction of the quadratic surfaces would be appropriate.

As we already mentioned in section 4.3, we also plan to explore the properties of the matrix Γ. The conditions of the invertibility are not clear yet, although the matrix Γ seems to be singular when all points $\mathbf{p}_i \in \mathcal{N}_{\mathrm{T}}$ lie in a plane.

One way for future improvement of the scheme is the use of alternative objective function, perhaps leading to different optimization process. For example, instead of using the gradient $\nabla^{\mathrm{x}} f$, one could use the normalized gradient of f. This would allow working with only normalized vectors. However, such approach necessarily leads to nonlinear optimization and higher time complexity – the price we need to pay for potentially improved results.

As we already mentioned in section 4.6, the scheme could be improved by computing the weights algorithmically using the local geometry of the mesh. For instance, the vertex $\mathbf{v}_\alpha$ adjacent to the angle α should be taken with greater weight than the vertex $\mathbf{v}_\beta$ adjacent to the angle $\beta < \alpha$. Similar conditions could be used for other metric properties in the mesh, such as the area of a triangle or the length of an edge. Moreover, the weights lack a thorough mathematical analysis. In this thesis, we only included an experimental (qualitative) study of their impact on the limit surface.

An interesting complement of the proposed scheme could be the design of the *dual*

decimation (if it does not exist yet). A large-scale mesh decimated using such dual decimation and then refined back using the quadric fitting refinement would be better approximated by the refined mesh than using any other decimation.

Two alterations to the scheme were suggested by [Fer14]. In our current setup, when picking the new point from the quadric, the barycenter of the triangle is used. The different setup could involve the use of another point from the interior of the triangle, such as the center of the inscribed circle. The other alteration involves picking the point from a cubic surface. Such setup might be interesting, although possibly superfluous.

Bibliography

[AEV03] Nicolas Aspert, Touradj Ebrahimi, and Pierre Vandergheynst. Non-linear subdivision using local spherical coordinates. *Computer Aided Geometric Design*, 20(3):165–187, 2003.

[Arc13] World Archaeology. Ice age art: arrival of the modern mind. `www.world-archaeology.com/more/ice-age-art-arrival-of-the-modern-mind.htm`, February 2013. Accessed: 2014-04-22.

[BPK+07] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Lévy, Stephan Bischoff, and Christian Rössl. Geometric modeling based on polygonal meshes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 Courses*, 2007.

[BSBK02] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. OpenMesh – a generic and efficient polygon mesh data structure, 2002.

[CC78] Edwin Catmull and James Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided design*, 10(6):350–355, 1978.

[CF01] Richard J Campbell and Patrick J Flynn. A survey of free-form object representation and recognition techniques. *Computer Vision and Image Understanding*, 81(2):166–210, 2001.

[Cha74] George Merrill Chaikin. An algorithm for high-speed curve generation. *Computer graphics and image processing*, 3(4):346–349, 1974.

[CJ07] Pavel Chalmovianský and Bert Jüttler. A non-linear circle-preserving subdivision scheme. *Advances in Computational Mathematics*, 27(4):375–400, 2007.

[DKT98] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 85–94. ACM, 1998.

[DLG90] Nira Dyn, David Levin, and John A Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics (TOG)*, 9(2):160–169, 1990.

[DS78] Daniel Doo and Malcolm Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.

[Far96] Gerald E Farin. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*. Academic Press, Inc., 1996.

[Fer14] Andrej Ferko. Discussion after the lecture. Diploma Thesis Seminar, FMPH, Bratislava. March 13, 2014.

[FHK02] Gerald E Farin, Josef Hoschek, and Myung-Soo Kim. *Handbook of computer aided geometric design*. Elsevier, 2002.

[GH97] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *In Proceedings of SIGGRAPH 97, Annual Conference Series*, pages 209–216, 1997.

[Har99] Erich Hartmann. On the curvature of curves and surfaces defined by normalforms. *Computer Aided Geometric Design*, 16(5):355–376, 1999.

[Hit14] Don Hitchcock. Dolni Vestonice jewellery, pottery, tools and other artifacts. `http://donsmaps.com/dolnivpottery.html`, April 2014. Accessed: 2014-04-22.

[HLS93] Josef Hoschek, Dieter Lasser, and Larry L Schumaker. *Fundamentals of computer aided geometric design*. AK Peters, Ltd., 1993.

[Kob00] Leif Kobbelt. $\sqrt{3}$-subdivision. In *Proceedings of SIGGRAPH 2000, Annual Conference Series*, pages 103–112, 2000.

[KSH00] Stefan Karbacher, Stephan Seeger, and Gerd Häusler. A non-linear subdivision scheme for triangle meshes. In *VMV*, pages 163–170, 2000.

[Loo87] Charles Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, August 1987.

[MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*, pages 35–57. Springer, 2003.

[PR08] Jörg Peters and Ulrich Reif. *Subdivision surfaces*. Springer, 2008.

[Sab10] Malcolm Sabin. *Analysis and Design of Univariate Subdivision Schemes*. Springer Berlin Heidelberg, 2010.

[San10] Conrad Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.

[SDRS96] Eric Stollnitz, Tony De Rose, and David Salesin. *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann, 1996.

[She12] Jonathan Richard Shewchuk. Unstructured mesh generation. In Uwe Naumann and Olaf Schenk, editors, *Combinatorial Scientific Computing*, chapter 10, pages 259–299. CRC Press, 2012.

[SZ00] Peter Schröder and Denis Zorin, editors. *Subdivision for modeling and animation. Course notes*. ACM SIGGRAPH, 2000.

[WW01] Joe Warren and Henrik Weimer. *Subdivision methods for geometric design: A constructive approach*. Morgan Kaufmann, 2001.

[Yan05] Xunnian Yang. Surface interpolation of meshes by geometric subdivision. *Computer-Aided Design*, 37(5):497–508, 2005.

[ZSS96] Denis Zorin, Peter Schröder, and Wim Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of SIGGRAPH 96, Annual Conference Series*, pages 189–192, 1996.

[ZSS97] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 259–268. ACM Press/Addison-Wesley Publishing Co., 1997.