

**University of Oslo
Department of Informatics**

**The Simplified
Surface Spline**

**Erik Christopher
Dyken**

**Candidatus
Scientiarum Thesis**

January 24, 2003



Abstract

This thesis studies the problem of using surface splines as an approximation basis. The simplified surface spline basis, a slight simplification of the C^1 -surface spline construction of Jörg Peters, is presented. The simplified surface does not guarantee a C^1 surface, but has explicit formulas, explicit basis functions and known dimension. In addition, it resides close to the C^1 -surface spline. The simplified surface spline basis is employed in interpolation, least squares approximation and faired least squares approximation. In addition, the mesh notation, a notation for writing algorithms on polyhedral meshes in a concise and mathematical way, is presented.

Foreword

This thesis is part of my *Candidatus Scientiarum* studies at the Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo. These graduate studies are composed of one semester of courses and two semesters of scientific research. This thesis is the result of my scientific research.

Initially, the scope of my scientific research was to do an approximation using surface splines. This included triangle surface generation, control mesh generation, parameterisation of the fine triangle mesh over the control mesh, blend ratio estimation, and finally approximation. I quickly realized that in order to overcome this ordeal, I had to treat all aspects of this process rather superficially. Thus, I decided to pick just one aspect of this process instead, and study this in some detail.

It was not particularly difficult to choose which of these subjects to pick. Triangle surface generation is a well-populated scene, and I doubt I can give a significant contribution to this topic. Control mesh generation, on the other hand, is a more open question. This involves finding a base mesh for, among other uses, approximation. Blend ratio estimation and surface spline approximation methods are both very open questions. With inspiration from the approximation theory of B-splines presented in “Spline Methods” (Mørken & Lyche 2002), a compendium for a graduate course in spline methods, I decided to try doing something similar with surface splines. Thus, my topic becomes approximation with surface splines.

My intention with this thesis is to give a respectable account of the scheme and forms of evaluation, finding expressions for the basis functions, finding the dimension of this basis, and finding out if in fact this basis always was of full rank. The reader may be amused by my ambition. However, my intention was never to create a surface spline theory comparable to spline theory. The spline theory was only a source of inspiration.

I find it a yielding practice to do things simple first, and when the simple thing works, expand it to the full thing. Moreover, this was the strategy employed on the surface spline. The simple thing is the simplified surface spline. However, the study of approximation with the simplified surface spline basis grew into a topic large enough to be covered alone in this thesis.

Thus, extending my work to the “full” surface spline scheme must be an ordeal left to future generations of inquisitive minds — if anyone cares to bother.

I use the opportunity here to thank some of the people that have helped me with this thesis, ranging from general inspiration to the trade of mathematical writing.

I wish to thank Professor Knut Mørken, my supervisor, for giving me the freedom to mould this thesis into precisely what I wanted, as well as numerous valuable comments, discussions tips, and constructive criticism.

I also wish to thank Martin Reimers, a Ph.D. fellow here at the department, for friendship as well as countless discussions, comments and criticism — none necessarily limited to surface splines.

I also wish to thank my parents for encouragement and proofreading.

I also wish to thank the employees at Sim Surgery A/S, especially Jan Sigurd Røtnes and Geir Westgaard. Røtnes initially posed the problem of my thesis, and Westgaard has given me a lot of useful information on the surface spline schemes, as well as discussions, comments and constructive criticism.

Finally, I wish to thank my fellow students at “Parken” study room for the good social environment.

Erik Christopher Dyken

Oslo, January 24, 2003

Contents

Abstract	iii
Foreword	v
1 Introduction and background	1
1.1 About surface splines and meshes	1
1.1.1 Surface splines in general	1
1.1.2 The simplified surface spline	2
1.1.3 About mesh notation	2
1.2 Organisation of thesis	2
1.3 Literature review	3
1.4 Background material	4
1.4.1 Convex combinations	4
1.4.2 Bézier splines	6
1.4.3 Composite Bézier curves and surfaces	8
1.4.4 B-splines	9
1.5 Summary	10
2 Meshes	11
2.1 Some notes on sets	11
2.2 The mesh definition	12
2.2.1 The vertices of a face	14
2.2.2 Positions in the mesh	14
2.2.3 The number of mesh elements	16
2.3 Mesh queries	17
2.3.1 Relational concepts	17
2.3.2 The structure	19
2.3.3 Dart selection and projection	21
2.4 Mesh modification and refinement	24
2.4.1 Dual mesh	24
2.4.2 Face split (polyhedral split)	25
2.4.3 Vertex split (mid-edge subdivision)	27
2.4.4 The Doo-Sabin split	28
2.4.5 Planar cut polyhedra	28
2.5 Summary	29

3	The surface spline curve	31
3.1	The scheme	31
3.1.1	Refinement	32
3.1.2	Sub-interval midpoints	32
3.1.3	Bézier coefficients	33
3.1.4	Defining the Bézier curve segments	34
3.1.5	Summary of construction	36
3.2	Explicit formulas	39
3.3	Exists a subdivision formula?	39
3.3.1	Bézier segment subdivision	39
3.3.2	Surface spline subdivision	41
3.3.3	Futile attempt no. 1: inserting one point	41
3.3.4	Futile attempt no. 2: uniform refinement	43
3.4	Summary	44
4	Surface spline schemes	45
4.1	The C^1 -surface spline	45
4.1.1	The scheme	47
4.2	Two other surface spline schemes	50
4.2.1	Surfaces . . . bicubics	50
4.2.2	Localised-hierarchy surface splines	52
4.3	Summary	53
5	The simplified surface spline	55
5.1	Overview	55
5.1.1	Topological terms of the simplified surface spline	56
5.1.2	The three meshes	57
5.1.3	The intrinsic parameterisation	58
5.2	Layer I: The control mesh vertices	58
5.3	Layer II: The quad mesh points	58
5.4	Layer III: Quad midpoints (inner mesh vertices)	60
5.5	Layer IV: Bézier coefficients	62
5.6	Layer V: Bézier evaluation	65
6	The simplified surface spline basis	67
6.1	Overview	68
6.2	Layer II: Quad mesh points	70
6.3	Layer III: Quad midpoints	71
6.4	Layer IV: Bézier coefficients	75
6.5	Layer V: The surface in terms of basis functions	77
6.6	The simplified surface spline space	79
6.6.1	The space	79
6.6.2	Linear independence	80
6.6.3	A local criterium	81
6.6.4	The local quad midpoint matrices	82
6.6.5	The simplified surface spline space and its basis	84
6.7	Summary	86

7	Practical considerations	87
7.1	Evaluation of layer based schemes	87
7.1.1	Evaluation in matrix guise	87
7.1.2	Direct and indirect evaluation	88
7.1.3	Retained Bézier evaluation	88
7.2	OpenGL rendering of surfaces with local support	89
7.2.1	Efficient rendering of Bézier patches	89
7.2.2	Frustum culling	90
7.2.3	Occlusion tests	92
7.3	Benefits of using composite polynomial patches	93
7.3.1	Ray tracing	93
7.3.2	Mass, centre of gravity and moment of inertia tensor	94
8	Approximation experiment	95
8.1	Our choice of parameterisation	95
8.2	The original models	96
8.3	Error, approximation order and stability	97
8.4	Four approximation methods	97
8.4.1	Lazy approximation	97
8.4.2	Vertex interpolation	98
8.4.3	Least squares	98
8.4.4	Faired least squares	99
8.5	Application	100
8.5.1	Overview	100
8.5.2	Nitty-gritty details	101
8.6	Results	104
9	Conclusions and further work	121
	References	122
	Index	124

List of Figures

1.1	Lay-out of Bézier control points	8
2.1	A mesh	13
2.2	Positions in a mesh	15
2.3	Overview of dart queries	21
2.4	A consecutive set of corners	24
2.5	Face split and vertex split	25
3.1	Layout of surface spline curve construction	35
3.2	Example of a surface spline curve	36
3.3	A surface spline curve and a cubic Bézier segment	38
3.4	Inserting one control point.	40
4.1	Different blend ratios	46
4.2	The control mesh, the quad mesh, and the inner mesh	47
4.3	C^1 -surface spline mesh refinement	48
4.4	C^1 -surface spline edge cutting	49
4.5	C^1 -surface spline quadratic meshing	50
4.6	Construction of “surfaces . . . bicubics”	51
5.1	The simplified surface spline concepts	57
5.2	The simplified surface spline control mesh	59
5.3	The simplified surface spline quad mesh	59
5.4	The simplified surface spline quad midpoints	61
5.5	The simplified surface spline Bézier coefficients	63
5.6	The simplified surface spline boundary coefficients	64
5.7	The simplified surface spline Bézier coefficient layout	65
6.1	Deduction of basis support	74
6.2	Construction of the local quad midpoint matrix.	81
6.3	The inequality of lemma 6.6.10	85
7.1	The view frustum.	90
7.2	The vertex transformation pipeline of OpenGL.	91
8.1	Class diagram of application	102
8.2	Approximation: cube0	106
8.3	Approximation: cube1	107
8.4	Approximation: cube2	108
8.5	Approximation: cube3	109

8.6	Approximation: cube4	110
8.7	Approximation: sphere0	111
8.8	Approximation: sphere1	112
8.9	Approximation: sphere2	113
8.10	Approximation: sphere3	114
8.11	Approximation: sphere4	115
8.12	Approximation: jittercube	116
8.13	Approximation: bunny	117

List of Tables

5.1	The simplified surface spline terms	56
8.1	Data on original models	96
8.2	Results from lazy approximation	118
8.3	Results from vertex interpolation	118
8.4	Results from least squares approximation	118
8.5	Results from faired approximation	119

Chapter 1

Introduction and background

This is a thesis about surface splines. We present *the simplified surface spline*, a simplification of the “ C^1 surface splines”-scheme by Jörg Peters (Peters 1995a). The thesis can be divided into four subjects:

1. The mesh notation, a clean, concise and mathematical method of defining subsets of connectivity. This notation is not restricted to analysis of surface spline schemes.
2. A thorough investigation of the simplified surface spline scheme. Much of the structure of this investigation can be applied to other surface spline schemes. In addition, a few approximation experiments are performed with the simplified surface spline basis.
3. An overview of a few surface spline schemes.
4. A thorough discussion of practical advantages for using surface spline schemes. Much of this discussion is applicable for any scheme based on convex combinations.

The contribution of this thesis with originality are items (1) and (2). Items (3) and (4) are more of an overview of existing literature and knowledge.

1.1 About surface splines and meshes

We give a quick overview of surface spline schemes in general, and then present the motivation of the simplified scheme.

1.1.1 Surface splines in general

Surface splines may be regarded as a macro spline, similar to the Clough-Tocher and Powell-Sabin macro splines. The surface spline tries to have the same “conceptual” framework as B-splines, where the connectivity and the blend ratios play the same role as knot vectors do for B-splines. Given a control mesh, this mesh is refined and weights called blend ratios are used to tune the location of the refined vertices. The refined mesh and vertices are then used to construct a connected patchwork of tensor product B-spline patches. These patches abut with the desired smoothness.

The surface spline scheme generates a surface from control meshes of rather arbitrary topologies. The surface does satisfy standard smoothness criteria. The number of patches output is usually kept to a minimum, and a closed form of the surface exists. The parameterisation is simple and based on standard patches. Thus the surface is not difficult to evaluate, render, differentiate, and integrate.

With this said, the surface spline does not possess the subdivision property essential for wavelets. Further, the construction is not particularly simple and elegant — many of the schemes are in fact rather hard to implement. In addition, the schemes are non-trivial to generalise for arbitrary degree and smoothness, and there exists little approximation theory for these schemes.

1.1.2 The simplified surface spline

The reason of this simplification is to make a simple prototype for experimenting with methods of finding basis functions and properties.

When such a method is found, it can be applied to the full scheme. Investigating the scheme in full is not in the scope of this thesis. However, we will examine the simplified surface spline in detail: various forms of construction, the basis functions, and how to obtain them, as well as some properties of the scheme.

The focus of the thesis is approximation properties of the simplified surface spline. To shed light on this, we present four kinds of approximation schemes, and put these to practice with some experiments.

1.1.3 About mesh notation

Surface splines are defined over arbitrary topology meshes. We have found no useful notation for dealing with arbitrary meshes that both is suitable for mathematical analysis as well as describing a potential interface between algorithms and the connectivity kernel¹. We call this notation *mesh notation*. Mesh notation is the result of applying the relational database paradigm on geometrical objects. Using relational databases for geometric modelling per se is nothing new, relational databases are sometimes used in GIS-systems.

However, we have never encountered the use of this paradigm to define a notation to extract certain subsets of the connectivity. Connectivity information in the guise of relational algebra, combined with mathematical set operations, yields some interesting results. We will deal with this in chapter 2.

1.2 Organisation of thesis

We give a small overview of the chapters:

1. **Introduction and background** gives a rough overview of the problem and the contents of this thesis, as well as a literature review. We give an overview of convex combinations and some properties of these operations. Convex combinations are used to prove some properties of the simplified surface spline. The rest of the chapter is a quick overview, to set the a context, of Bézier splines and B-splines.

¹By connectivity kernel we mean whatever data structure containing the connectivity of the object as well as interface function

2. **Meshes** defines the concepts of meshes and mesh queries. Mesh queries are used extensively in the following chapters.
3. **The surface spline curve** serves as an introduction to surface spline schemes. In addition, the boundary curve of the simplified surface spline is such a curve.
4. **Surface spline schemes** gives an overview of three existing surface spline schemes.
5. **The simplified surface spline** gives an account of the simplified surface spline.
6. **The simplified surface spline basis** gives a detailed account of how the basis functions were found, and a few properties are deduced.
7. **Practical considerations** gives a discussion about practical employment of surface splines, with substantial weight on methods of evaluation and rendering.
8. **Approximation experiment** is an experiment to test the simplified surface spline basis deduced in chapter 6.
9. **Conclusions** tries to summarise this thesis. In addition, a road map for further research on approximation with surface splines is given.

1.3 Literature review

A relatively small amount of literature on the topic of surface splines exists. A few books mention the scheme, in the fourth edition of (Farin 1996) a small presentation of surface splines is given, and a few articles employ these schemes.

The Primary source of information on surface splines is found in the articles of Jörg Peters. The earliest article from Peters on surface splines is (Peters 1994), and this scheme is discussed in section 4.2.1. This scheme is improved in (Peters 1995a), which is discussed in section 4.1. This is the scheme that the simplified surface spline is based upon. In addition to a presentation of the construction, proofs for continuity and the convex hull property are given in the paper.

In (Gonzalez & Peters 1999) the C^1 surface spline is extended to incorporate local refinement and topology changes. However, the refinement procedure does not give the *exact* same surface. Finally, in (Peters 1996) a G^2 surface spline construction is given.

On application of surface splines, (Eck & Hoppe 1996) gives an account of employing surface splines when reconstructing B-splines from point clouds.

The article (Gonzalez-Ochoa, McCammon & Peters 1998) gives an account of how, by using the great theorem of Gauss, to compute various physical properties of objects defined by piecewise polynomials.

Very little is written on the topic of planar cut polyhedra. Planar cut polyhedra is mentioned in some of Peters' papers, especially in (Peters 1998) where an algorithm to convert planar cut polyhedra to trimmed NURBS patches is given.

In (Farin 1996), a thorough account on continuity of composite spline patches is given. Another account, probably more aimed at surface splines, are given in (Peters 2001). This paper reveals some of the motivation behind the steps of surface spline constructions.

The mesh notation is based on G-maps. The articles (Halbwachs & Hjelle 1999) and (Halbwachs, Courriouz, Renaud & Repusseau 1996) both give a good description of the concept of G-maps, as well as relevant references.

1.4 Background material

In this section we will look at Convex combinations, Bézier splines and B-splines and surface splines. Convex combinations are used extensively in this text, and Bézier patches are used in the surface spline construction.

1.4.1 Convex combinations

We assume that all points mentioned are points in a linear space. Thus, this will not be stated explicitly, but nevertheless it is an important requirement. We begin with the formal definition of a convex set.

Definition 1.4.1. A set \mathbb{S} is *convex* if

$$\alpha \mathbf{p}_0 + (1 - \alpha) \mathbf{p}_1 \in \mathbb{S}, \quad \forall \mathbf{p}_0, \mathbf{p}_1 \in \mathbb{S}, \quad \text{and} \quad \alpha \in [0, 1] \quad (1.1)$$

A convex set is a set where any line segment between two elements of the set is contained in the set as well. We also define the convex hull.

Definition 1.4.2. The *convex hull* of the set of points \mathbb{P} , $\text{CH}(\mathbb{P})$, is the smallest convex sets containing all of the points of \mathbb{P} .

The linear interpolation of definition 1.4.1 is in fact a convex combination. We give this concept a formal definition.

Definition 1.4.3. A *convex combination* of the points $\mathbf{c}_1, \dots, \mathbf{c}_n$ is a construction

$$\mathbf{c} = \sum_{i=1}^n \alpha_i \mathbf{c}_i, \quad (1.2)$$

where the weights α_i satisfy the two requirements

1. $\alpha_i \geq 0 \quad i = 1, \dots, n$
2. $\sum_{i=1}^n \alpha_i = 1.$

If we can write an expression as a convex combination, the result has some useful properties, in particular, the convex hull property.

Property 1.4.4. A convex combination always resides inside the convex hull of the points used in the convex combination.

This follows from the definitions 1.4.1 and 1.4.2. Convex combinations of convex combinations are in turn convex combinations themselves. This is useful when proving that a construction is a convex combination.

Property 1.4.5. Given a set of points \mathbb{P} , let \mathbb{Q} be a set of points defined as convex combinations of the points in \mathbb{P} . Let the set \mathbb{R} be a set of points defined as convex combinations of the points in \mathbb{Q} . Then, the points in \mathbb{R} are convex combinations of the points in \mathbb{P} .

Proof. Let a convex combination of a convex combination be

$$\mathbf{c} = \sum_{i=1}^{n_\alpha} \alpha_i \left(\sum_{j=1}^{n_\beta} \beta_{ij} \mathbf{c}_j \right) \quad (1.3)$$

Since both combinations are convex, all weights are positive. Any multiplication of two of these weights, $\alpha_i \cdot \beta_j$ will be positive as well; hence requirement (1) of definition 1.4.3 is satisfied. Further,

$$\sum_{i=1}^{n_\alpha} \alpha_i \left(\sum_{j=1}^{n_\beta} \beta_j \right) = \sum_{i=1}^{n_\alpha} \alpha_i \cdot 1 = 1, \quad (1.4)$$

which fulfils requirement (2) of definition 1.4.3. \square

Instead of writing convex combinations as a sum, we can write it as a matrix product between a row vector of weights and a column vector of the objects to be combined. If we stack several of these convex combinations on top of each other, we get a convex combination matrix.

Definition 1.4.6. A *convex combination in matrix form* is the construction

$$\begin{aligned} \mathbf{P} &= \mathbf{AC} \\ \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_m \end{bmatrix} &= \begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & \ddots & \vdots \\ \alpha_{m1} & \cdots & \alpha_{mn} \end{bmatrix} \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_n \end{bmatrix} \end{aligned} \quad (1.5)$$

where the elements of the weight matrix \mathbf{A} satisfy

1. $\alpha_{ij} \geq 0$, $1 \leq i \leq m$, $1 \leq j \leq n$, and
2. $\sum_{j=1}^n \alpha_{ij} = 1$, $1 \leq i \leq m$.

Each row of \mathbf{A} consists of the weights of one convex combination. We will prove one situation where the weight matrix is non-singular. In order to do so, we must define diagonally dominant matrices.

Definition 1.4.7. If the entries a_{ij} of a matrix \mathbf{A} possess the following property,

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}| \quad 1 \leq i \leq n, \quad (1.6)$$

then \mathbf{A} is a diagonally dominant matrix.

Diagonally dominant matrices possess two useful properties:

Property 1.4.8. If the matrix \mathbf{A} is diagonally dominant, the following is true:

1. The matrix \mathbf{A} is non-singular and has an LU-factorisation.
2. The system $\mathbf{Ax} = \mathbf{b}$ can be solved by Gaussian elimination without pivoting.

Proofs for both properties can be found in (Cheney & Kincaid 1996, pp.188-190). We will now show when the weight matrix of a convex combination is diagonally dominant, and thus, possesses properties 1.4.8.

Property 1.4.9. The weight matrix of a convex combination in matrix form is diagonally dominant if

1. $m = n$, and

2. $\alpha_{ii} > 1/2$, for $1 \leq i \leq n$,

that is, it is square and the diagonal entries are larger than $1/2$.

Proof. Each row sum to one and each entry in the matrix is non-negative. Thus, for each row, if the diagonal element is larger than $1/2$, then the sum of the other entries of the row will be smaller than $1/2$, and therefore the matrix is diagonally dominant according to definition 1.4.7. \square

We think it is good practice to define terms before using them. Thus, we boldly ignore the potential peril of being regarded as pedantic, and start with convex combinations and Bernstein polynomials.

1.4.2 Bézier splines

Bézier splines are closely related to B-splines, in fact it is possible to transform any B-spline to a Bézier spline by knot insertion, and thus it is always possible to write a Bézier spline as a B-spline, it is just a matter of choosing a representation. The Bézier curves and surfaces represent a fundamental building block for surface spline schemes.

Bernstein polynomials

Before defining Bézier curves and surfaces, we define the Bernstein polynomials.

Definition 1.4.10. *The Bernstein polynomials of degree d are defined as*

$$B_i^d(u) = \binom{d}{i} u^i (1-u)^{d-i}, \quad u \in [0, 1] \quad (1.7)$$

where $i = 0, \dots, d$.

The Bernstein polynomials exhibit some useful properties.

Property 1.4.11. *The Bernstein polynomials are*

1. a partition of unity,
2. all non-negative on the interval $[0, 1]$, and
3. a basis for polynomials.

Proofs are found in (Farin 1996). From (1) and (2) we see that the set of Bernstein polynomials for a given degree can be used as weights for a convex combination.

Bézier curves

The Bernstein polynomials form the basis for the Bézier curve:

Definition 1.4.12. *The Bézier curve segment of degree d is defined as*

$$\mathbf{f}(u) = \sum_{i=0}^d B_i^d(u) \mathbf{c}_i, \quad \mathbf{c}_i \in \mathbb{R}^n \quad (1.8)$$

where $B_j^d(u)$ are the Bernstein polynomials and $\mathbf{c}_0, \dots, \mathbf{c}_d$ are the $d+1$ control points of the curve.

We will give an example of two Bézier curves, the linear interpolation and the quadratic Bézier blend. These two curves will be used later in the text.

Example 1.4.1. *The linear interpolation is a Bézier curve of degree 1,*

$$\begin{aligned} \mathbf{f}(u) &= \sum_{i=0}^1 B_i^1(u) \mathbf{c}_i \\ &= B_0^1(u) \mathbf{c}_0 + B_1^1(u) \mathbf{c}_1 \\ &= (1-u) \mathbf{c}_0 + u \mathbf{c}_1. \end{aligned} \quad (1.9)$$

The quadratic Bézier blend is a Bézier curve of degree 2,

$$\begin{aligned} \mathbf{f}(u) &= \sum_{i=0}^2 B_i^2(u) \mathbf{c}_i \\ &= B_0^2(u) \mathbf{c}_0 + B_1^2(u) \mathbf{c}_1 + B_2^2(u) \mathbf{c}_2 \\ &= (1-u)^2 \mathbf{c}_0 + 2(1-u)u \mathbf{c}_1 + u^2 \mathbf{c}_2. \end{aligned} \quad (1.10)$$

Tensor product Bézier surfaces

The tensor product of two Bézier curve bases creates a bivariate tensor product Bézier basis. For details, see (Farin 1996) or (Gallier 2000).

Definition 1.4.13. *A Bézier tensor product surface $\mathbf{s}(u, v)$ is defined as*

$$\mathbf{s}(u, v) = \sum_{i=0}^{d_u} \sum_{j=0}^{d_v} \mathbf{c}_{i,j} B_i^{d_u}(u) B_j^{d_v}(v), \quad \mathbf{c}_{i,j} \in \mathbb{R}^n, \quad (1.11)$$

where $B_i^{d_u}(u)$ and $B_j^{d_v}(v)$ are Bernstein polynomials and $\mathbf{c}_{i,j}$ are elements of a $d_u \times d_v$ grid of control points.

We give an example of two tensor product surfaces (both will be used in the simplified surface spline construction).

Example 1.4.2. *The bilinear interpolation is a tensor product of two linear interpolations, thus the degrees are both 1 in the u and v directions.*

$$\begin{aligned} \mathbf{s}(u, v) &= \sum_{i=0}^1 \sum_{j=0}^1 \mathbf{c}_{i,j} B_i^1(u) B_j^1(v) \\ &= (1-u)(1-v) \mathbf{c}_0 + (1-u)v \mathbf{c}_1 + u(1-v) \mathbf{c}_2 + uv \mathbf{c}_3. \end{aligned} \quad (1.12)$$

The biquadratic blend is a tensor product of two quadratic Bézier blends. In this case the degrees are 2 in both the u and v directions.

$$\begin{aligned} \mathbf{s}(u, v) &= \sum_{i=0}^2 \sum_{j=0}^2 \mathbf{c}_{i,j} B_i^2(u) B_j^2(v) \\ &= (1-u)^2(1-v)^2 \mathbf{c}_{0,0} + 2(1-u)^2(1-v)v \mathbf{c}_{0,1} + (1-u)^2 v^2 \mathbf{c}_{0,2} + \\ &\quad 2(1-u)u(1-v)^2 \mathbf{c}_{1,0} + 4(1-u)u(1-v)v \mathbf{c}_{1,1} + 2(1-u)uv^2 \mathbf{c}_{1,2} + \\ &\quad u^2(1-v)^2 \mathbf{c}_{2,0} + 2u^2(1-v)v \mathbf{c}_{2,1} + u^2 v^2 \mathbf{c}_{2,2}. \end{aligned} \quad (1.13)$$

Figure 1.1 gives an idea of how the control points relate to the parameter space.

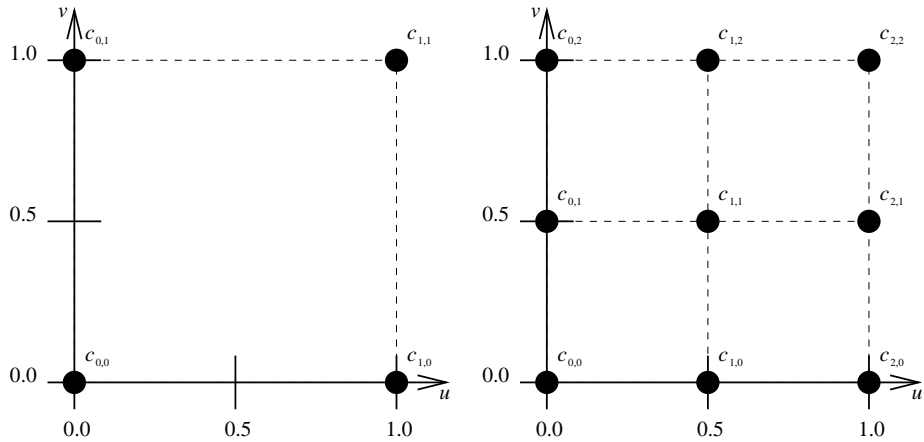


Figure 1.1: Lay-out of the control points for the bi-linear interpolation (left) and the bi-quadratic Bézier interpolation (right).

Properties of Bézier curves and surfaces

Bézier curves and surfaces have numerous useful properties, most of which stem from the fact that these shapes are convex combinations of the control points. We will present some of them here, though without motivation and proofs — the reader is encouraged to consult texts like (Farin 1996), (Gallier 2000) and (Foley, van Dam, Feiner & Hughes 1996).

Property 1.4.14. *Bézier curves and surfaces exhibit the following properties:*

1. *The derivative of a Bézier curve or surface is another Bézier curve or surface of lower degree, where the coefficients are differences of the original coefficients.*
2. *The Bézier curve or surface reside inside the convex hull of the control points.*
3. *Bézier curves or surfaces of degree 1 or higher exhibit linear precision (reproduces a line or plane).*
4. *Bézier curves interpolate its control points at the end, and the Bézier surfaces interpolate the four corner control points.*

1.4.3 Composite Bézier curves and surfaces

Polynomials are infinitely smooth; hence the continuity properties of schemes producing piecewise polynomials are only interesting at the joints. We will not treat this subject with great rigour since it is not in the major scope of this thesis. The reader is referred to (Farin 1996) and (Peters 2001) for discussions of continuity and geometric continuity, where (Peters 2001) is very focused on requirements for surface spline schemes. We define the concepts of continuity at a joint *very loosely*:

Definition 1.4.15. *A composite polynomial has the following continuity properties at a joint if the following properties match for all the pieces meeting at the joint:*

1. C^0 or G^0 if the positions match.

2. C^1 if all of the derivatives match.
3. G^1 if the tangent planes match.

In general, C^n -continuity is dependent on the parametrisations: The speed in parameter space must match across the joint. G^n -continuity requires only such a reparametrisation to exist. Thus, there is a difference between C^0 and G^0 . For a surface to be C^0 , the parameter spaces must match.

Contrary to intuition, G^1 can be stricter than C^1 . G^1 requires the tangent planes to exist, and thus, the derivatives of the parametrisation can not drop to zero. We call such a parameterisation a *regular parameterisation*. C^n -continuity does not have such a restriction. However, an irregular parameterisation where the “speed” of the parameterisation drops to zero can be desirable; being a mean to achieve sharp corners, as is the case with surface splines.

Bézier curves and surfaces are polynomials, and in order to make interesting shapes with them, they must be “glued” together. This “glue” is smoothness constraints. The derivatives of the Bézier splines are very simple, so these smoothness constraints can often be reduced to co-linearity of coefficients etc. However, it is difficult to create a single vector space for the composite curve. Surface splines, and B-splines in particular, solve this problem.

1.4.4 B-splines

A B-spline curve is composed of three elements: A *degree*, a *knot vector* and a set of *coefficients*. The degree is the degree of the polynomial pieces, and provides an upper bound for the smoothness at the knots. The knot vector is a vector of non-decreasing values and defines the parameter space of the B-spline. The relative spacing between the knots governs the tightness of the curve. If the spacing is small, the curve becomes sharper and closer to the control point, and if two knots coincide, the curve loses one degree of continuity at that parameter value. Thus, the knot vector provides a mean to control the *local shape* of the curve.

The degree and the knot vector together form a spline space. The coefficients define a spline of this spline space. The great advantage of B-splines is that smoothness of segment joints are automatically taken care of, and thus no peripheral constraints are needed.

Similar to Bézier surfaces, we can create a bivariate B-spline surface as the tensor product of two univariate B-spline curves. With this construction, we have the flexibility of two knot vectors. In addition, it is required that the control points is organised in a grid (and thus is not of arbitrary topology). To model a shape of a more arbitrary topology, we must slice the surface into pieces, and by performing this slicing, the unified behaviour gained by using B-splines rather than Bézier splines is lost since we yet again have some peripheral smoothness constraints.

There have been attempts to generalise the tensor product B-spline construction over arbitrary topologies (arbitrary is probably a bit of an over-statement, since all schemes do have *some* restrictions on the parameter domain. However, they are *more* arbitrary than the parameter domain of B-splines). This is not new, two papers were published in 1978 on this topic, (Catmull & Clark 1978) and (Doo & Sabin 1978). *Subdivision surfaces* and *surface splines* are offsprings of the strategies presented in these articles.

1.5 Summary

This initial part of this chapter gave a very brief overview of what surface splines are, as well as a literature review.

The latter part of this chapter gave some background information. The well known convex combinations are used to prove miscellaneous properties of the simplified surface spline scheme. The treatment of Bézier splines and B-splines are superficial. These topics are included to give a setting for the surface spline schemes in this text.

Chapter 2

Meshes

This chapter defines the mesh and mesh queries. A mesh is simply put the connectivity of an object defined by polygons. The interesting part is the mesh queries, the result of applying a relational database paradigm to mesh connectivity, and which enables us to specify parts of the mesh in a concise way.

We borrow some terms and concepts from G-maps. Information on G-maps can be found in (Halbwachs & Hjelle 1999) and (Halbwachs et al. 1996).

2.1 Some notes on sets

The mesh definition is based on sets. A set is a collection of *elements*. An element could be any entity, be it a vertex, an edge or another set. The size of a set \mathbb{A} , denoted $\#\mathbb{A}$, is the number of elements the set contains. Thus, if $\mathbb{A} = \{a, b, c\}$, then $\#\mathbb{A} = 3$.

An *offset invariant ordered set* is just as an ordered set, however with a different equivalence relation attached such that only the relative ordering of elements is considered.

Definition 2.1.1. *An offset invariant ordered set is an ordered set. Two offset invariant ordered sets are considered equal if one can be made identical to the other by cyclic shifting of the set's elements.*

Thus, only the contained elements and their relative order is of importance. If $\mathbb{A} = \{a, b, c, d\}$ and $\mathbb{B} = \{b, c, d, a\}$ are offset invariant ordered sets, they're considered equal since \mathbb{B} can be made identical to \mathbb{A} by applying one cyclic shift to the right on the set. However, the set $\mathbb{C} = \{a, c, b, d\}$ is equal to neither, since the relative order between the elements is disturbed.

We use pairs to define edges.

Definition 2.1.2. *A pair is a set of two distinct elements.*

Circuits are used to define faces.

Definition 2.1.3. *A circuit is a shift invariant ordered set of pairs where:*

1. *Each pair is used at most once.*
2. *Adjacent pairs contain one common element.*
3. *The elements of the pairs are used exactly twice by the pairs of the circuit.*

The fact that a circuit is shift invariant and the requirement (2) of definition 2.1.3 implies that the first and the last pair of a circuit contains the same element. The *length* of a circuit is the number of pairs from which the circuit is defined. This number happens to be the number of distinct elements used by the pairs of the circuit as well.

2.2 The mesh definition

Usually, a vertex has a geometrical position, and we define faces by connecting the vertices. In this context, the vertex serves two purposes. First of all, it is a “connectivity thingy” from which we can define faces. In addition, each vertex has as mentioned a position.

However, in the mesh paradigm we separate these two roles. A mesh is the *connectivity* of a shape. There is no geometric information in the mesh. However, we can *embed* (much in the same way as embeddings of G-maps) the mesh into a space by associating geometry with mesh elements (e.g. the vertices). If this embedded mesh is to define a surface, we must also associate a *scheme*. The scheme describes how to define a surface from the geometry and the connectivity.

There is an advantage of this. With surface splines, we let the mesh define a surface spline space, and the geometry attached to the vertices define a specific surface in this space. This is complementary to knot vectors and coefficients of B-splines.

We define the vertex, the edge and the face.

Definition 2.2.1. A *vertex* is a fundamental element, and has no properties. An *edge* is a pair of two vertices. A *face* is a circuit of edges.

Thus, edges are the elements of faces, and vertices are the elements of edges. We use the notational convention of letting v_i denote vertex i , e_j edge j , and f_k face k . Then we are ready to define the mesh.

Definition 2.2.2. A *mesh* is the collection of the three sets

$$\mathfrak{M} = (\mathbb{V}, \mathbb{E}, \mathbb{F}), \quad (2.1)$$

where \mathbb{V} is the set of vertices, \mathbb{E} is the set of edges and \mathbb{F} is the set of faces defining the mesh. The elements of the three sets must collectively satisfy:

1. Every vertex in \mathbb{V} must be used at least twice by edges in \mathbb{E} .
2. The edges in \mathbb{E} must be used at least once and at most twice by faces in \mathbb{F} .
3. If an edge is used by two faces, the faces must visit the vertices of the edge in opposing order.
4. The vertices in \mathbb{V} must be used by exactly zero or two boundary edges.

Requirement (1) of definition 2.2.2 makes sure that no vertices are left unused. This simplifies things when building approximation matrices. Further, requirement (2) assures that edges are used at least once, and not used by three or more faces. Three or more faces sharing a common edge cannot define a two-manifold surface. Requirement (3) assures a consistent ordering of the faces. Thus, a Möbius strip cannot be represented as a mesh. Requirement (4) assures a simple boundary. Faces are always of at least length three due to the circuit definition (the reason why is left as a little puzzle for the reader).

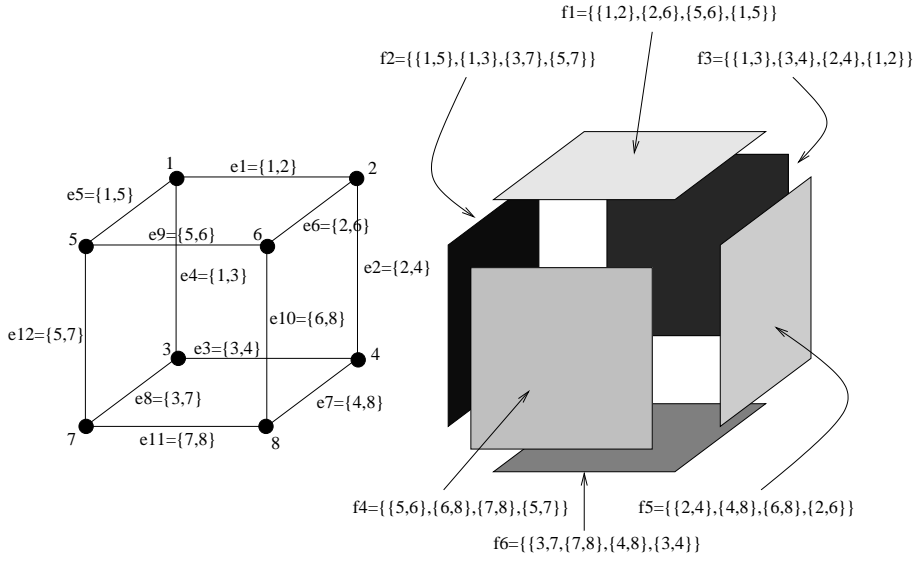


Figure 2.1: Example mesh. The numbers denote vertex numbers.

Example 2.2.1. We will now define the mesh of the unit cube (refer to figure 2.1). The unit cube is composed of eight vertices which we simply enumerate from 1 through 8,

$$\mathbb{V} = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}. \quad (2.2)$$

The unit cube has twelve edges, each connecting a pair of vertices:

$$\begin{aligned} \mathbb{E} &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\} \\ &= \{\{v_1, v_2\}, \{v_2, v_4\}, \{v_3, v_4\}, \{v_1, v_3\}, \{v_1, v_5\}, \{v_2, v_6\}, \\ &\quad \{v_4, v_8\}, \{v_3, v_7\}, \{v_5, v_6\}, \{v_6, v_8\}, \{v_7, v_8\}, \{v_5, v_7\}\}. \end{aligned} \quad (2.3)$$

And there are six faces:

$$\begin{aligned} \mathbb{F} &= \{f_1, f_2, f_3, f_4, f_5, f_6\} \\ &= \{\{e_1, e_6, e_9, e_5\}, \{e_5, e_4, e_8, e_{12}\}, \{e_4, e_3, e_2, e_1\} \\ &\quad \{e_9, e_{10}, e_{11}, e_{12}\}, \{e_2, e_7, e_{10}, e_6\}, \{e_8, e_{11}, e_7, e_3\}\} \\ &= \{\{\{v_1, v_2\}, \{v_2, v_6\}, \{v_5, v_6\}, \{v_1, v_5\}\} \\ &\quad \{v_1, v_5\}, \{v_1, v_3\}, \{v_3, v_7\}, \{v_5, v_7\}\} \\ &\quad \{v_1, v_3\}, \{v_3, v_4\}, \{v_2, v_4\}, \{v_1, v_2\}\} \\ &\quad \{v_5, v_6\}, \{v_6, v_8\}, \{v_7, v_8\}, \{v_5, v_7\}\} \\ &\quad \{v_2, v_4\}, \{v_4, v_8\}, \{v_6, v_8\}, \{v_2, v_6\}\} \\ &\quad \{v_3, v_7\}, \{v_7, v_8\}, \{v_4, v_8\}, \{v_3, v_4\}\}\} \end{aligned} \quad (2.4)$$

Note that the set of edges is in fact a set of sets of vertices, and the set of faces is a set of sets of sets of vertices. From these three sets we define the mesh:

$$\mathfrak{M} = (\mathbb{V}, \mathbb{E}, \mathbb{F}). \quad (2.5)$$

A mesh can contain most kinds of polygons. However, if the mesh only contains triangles, the mesh is called a *triangular mesh*, and if the mesh only contains quadrilaterals the mesh is called a *quad mesh*.

2.2.1 The vertices of a face

It is often useful to represent a face as a set of vertices instead of a circuit (which is a set of edges). We use the “hat” to denote when we use the face as a set of vertices instead of a set of edges.

Definition 2.2.3. *The vertices of a face are*

$$\hat{f} = \bigcup_{e_i \in f} e_i. \quad (2.6)$$

Thus \hat{f} is the union of all the edges of f , and since the edges are sets of vertices, the result is a large single set containing the vertices of the face.

Example 2.2.2. *Using the mesh defined in example 2.2.1 we have*

$$\begin{aligned} \hat{f}_1 &= \{v_2, v_6, v_5, v_1\} \\ \hat{f}_2 &= \{v_1, v_3, v_7, v_5\} \\ \hat{f}_3 &= \{v_1, v_3, v_4, v_2\} \\ \hat{f}_4 &= \{v_6, v_8, v_7, v_5\} \\ \hat{f}_5 &= \{v_4, v_8, v_6, v_2\} \\ \hat{f}_6 &= \{v_3, v_7, v_8, v_4\} \end{aligned} \quad (2.7)$$

2.2.2 Positions in the mesh

We use some terms to specify a position in the mesh. Figure 2.2 gives an illustration of some of them. *These terms specifies positions in the connectivity (mesh) and does not specify a geometrical position.* The first two of them, *vertices* and *edges*, are rather obvious. These two are the same concepts known from graph theory. In addition, a mesh has *faces*.

Then we have some more uncommon terms. The *corners* is where two faces of an edge connect.

Definition 2.2.4. *A corner is a vertex-face pair (v_i, f_k) where $v_i \in \hat{f}_k$. A corner represents a position between two edges in the face description. These two edges are named e_{in} or the incoming edge, which is the edge in f_k that connects **to** v_i , and the outgoing edge e_{out} which connects **from** v_i in the circuit.*

A face that is an n -gon has n corners. Each corner resides in the position where two edges meet (at a vertex) in this face. Since the circuit definition enforces a vertex to be visited either by none or by two edges, the corner definition causes no ambiguities. A

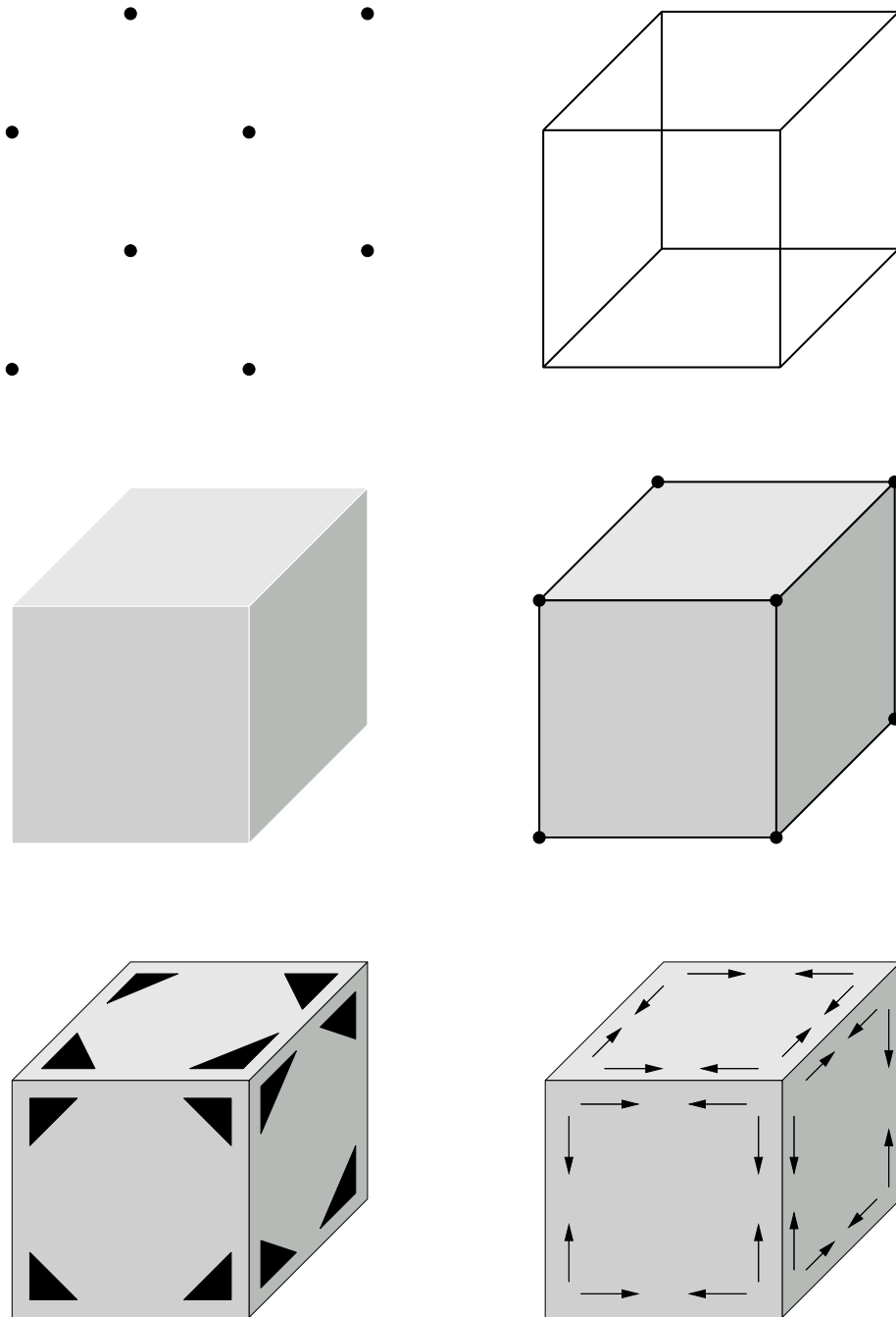


Figure 2.2: Positions in a mesh. The nodes (upper left), the edges (upper right) and the faces (middle left) forms a mesh (middle right). In this mesh we find corners, the black triangles, (lower left) and darts, the pointed arrows (lower right).

corner “has” one vertex (the vertex where the corner resides) as well as two edges, the incoming edge and the outgoing edge. Which edge is incoming and which is outgoing depends on the orientation of the faces.

At each end of each edge we have an edge-end.

Definition 2.2.5. A *edge-end* is a vertex-edge pair (v_i, e_j) where $v_i \in e_h$. The edge-ends are located on the end of edges.

Obviously, each edge has two edge-ends. Each edge also have a number of half-edges. The half-edges are along the sides of the edges.

Definition 2.2.6. A *half-edge* is a edge-face pair (e_j, f_k) where $e_j \in f_k$. The half-edges are located along an edge where a face abuts.

In this way, if m faces abut an edge, the edge has m half-edges. Since only one or two faces may abut an edge, consequently an edge only has one or two half-edges depending on whether or not the edge is a boundary edge. Boundary edges has only one half-edge, while non-boundary edges has two.

A concept borrowed from G-maps are the darts. Each half-edge has two darts. In contrast to the darts of the G-maps, our darts do not have any involutions defined. Our darts is a simple vertex-edge-face triple.

Definition 2.2.7. A *dart* is a vertex-edge-face triple (v_i, e_j, f_k) where $v_i \in e_j$ and $e_j \in f_k$. Darts are located on the end of half-edges.

From the definition of darts we see that the dart is not an arbitrary triple. This requirement put a lot of topological information of the mesh into the darts.

We can look at these positions as cursors in the connectivity at different “resolutions”. We let $v_i \in \mathbb{V}$, $e_j \in \mathbb{E}$, $f_k \in \mathbb{F}$, and $\mathfrak{M} = (\mathbb{V}, \mathbb{E}, \mathbb{F})$. We then have some *deduced sets* (they are not explicitly defined). The set \mathbb{C} is the set of all possible corner specifications in \mathfrak{M} , the set \mathbb{D} as the set of all possible darts, the set \mathbb{HE} as the set of all possible half-edges and \mathbb{EE} as the set of all possible edge-ends.

Example 2.2.3. The mesh defined in example 2.2.1 has 24 corners, and we define the set of corners \mathbb{C} :

$$\begin{aligned} \mathbb{C} = \{ & \{v_2, f_1\}, \{v_6, f_1\}, \{v_5, f_1\}, \{v_1, f_1\}, \{v_1, f_2\}, \{v_3, f_2\}, \\ & \{v_7, f_2\}, \{v_5, f_2\}, \{v_1, f_3\}, \{v_3, f_3\}, \{v_4, f_3\}, \{v_2, f_3\}, \\ & \{v_6, f_4\}, \{v_8, f_4\}, \{v_7, f_4\}, \{v_5, f_4\}, \{v_4, f_5\}, \{v_8, f_5\}, \\ & \{v_6, f_5\}, \{v_2, f_5\}, \{v_3, f_6\}, \{v_7, f_6\}, \{v_8, f_6\}, \{v_4, f_6\} \} \end{aligned} \quad (2.8)$$

2.2.3 The number of mesh elements

Some observations can be made about the different kind of mesh elements. However, we must first define some classifications of the mesh elements. We classify each edge to be a boundary or a non-boundary edge.

Definition 2.2.8. An edge is *boundary* if only one face abuts the edge. Otherwise, two faces abut the edge and the edge is *non-boundary*.

We let the two sets \mathbb{BE} and \mathbb{NE} be the sets of boundary edges and non-boundary edges respectively.

A vertex can have both a *edge degree* or *face degree*. The edge degree is the number of edges connected to the vertex, and the face degree is the number of faces abutting the vertex. If the vertex is interior (all connecting edges are non-boundary edges), the edge and face degree is the same.

Then to the observations. If the mesh is purely triangular, the number of corners $\#\mathbb{C} = 3\#\mathbb{F}$. Likewise, if the mesh is purely quadrilateral, then $\#\mathbb{C} = 4\#\mathbb{F}$. If the mesh is mixed with different kinds of polygons, the number of corners must be found by summation. Each triangle contributes with three corners, each quadrilateral with four, etc. In addition we have some invariants of the sizes of the sets of corners, edges and darts.

Property 2.2.9. *From the mesh \mathfrak{M} the set of corners \mathbb{C} and the set of darts \mathbb{D} may be deduced. We partition the set of edges in two, into the set of boundary edges \mathbb{BE} and the set of internal (non-boundary) edges \mathbb{IE} . The following is true:*

1. $\#\mathbb{C} = \#\mathbb{BE} + 2\#\mathbb{IE}$
2. $\#\mathbb{D} = 2\#\mathbb{C} = 2\#\mathbb{BE} + 4\#\mathbb{IE}$

Proof. Each internal edge is surrounded by four corners, and each boundary edge is surrounded incompletely by two corners. A corner abuts two edges, and this gives (1). Each corner contains two unique darts (one along the incoming edge and one along the outgoing edge), and this gives (2). \square

Sometimes, in sub- and superscripts of sums, the number of vertices in a mesh are denoted N_v , the number of edges are denoted N_e , the number of faces N_f , and the number of corners N_c , etc. The reason for this is purely aesthetically reasons. It is also worth noticing that even though both circuits and shift invariant ordered sets are in fact sets, we do not use the set font when dealing with edges and faces. The reader may think that this inconsistency can be a cause of confusion, however, in our humble opinion, the opposite is the case. Therefore, vertices, edges and faces are denoted in the same font.

2.3 Mesh queries

This sections presents the results from applying the paradigm of relational algebra (databases) on connectivity of geometrical models. This is indeed an unorthodox approach. However, we will see that this approach let us reap useful concepts.

In computational geometry we often need to do operations “for each abutting face”, “for each face defined by four vertices where one vertex is boundary” and so on. These verbose descriptions are (sometimes) intuitive, however, they lack the conciseness and compactness of a mathematical notation. Ambiguities is especially a problem when implementing algorithms. Mesh queries is supposed to be a notation to write algorithms of computational geometry in a compact and concise form.

2.3.1 Relational concepts

The idea is very based on relational algebra (used in database design). Relational algebra in a database setting *is not the same as a mathematical binary relation*. For

comprehensive information on relational algebra the reader is referred to one of the numerous books dealing with this topic, for example (Elmasri & Navathe 1994). However we will give a extremely brief overview of the concept used in this section.

A relation can be viewed as a table. Each column of this table represent a property, and is called an *attribute*. The class of data that can appear in a column is called the *domain* of that attribute. Each row represents a collection of related data values and is called a tuple. The tuple relates specific occurrences of the attributes together.

At least one attribute of an relation should be a *primary* key. If an attribute is the primary key, it is an unique identifier for tuples of the relation. An attribute can be a *foreign* key as well. This restricts the domain of the attributes in such a way that only tuples with a value in the attribute field that already exists in a referenced attribute in another relation are allowed.

The CREATE TABLE operation in SQL is used for relation creation:

```
CREATE TABLE <name of relation>
  ( <attribute, domain>, ...
    PRIMARY KEY (<attribute>),
    FOREIGN KEY (<attribute>)
      REFERENCES <other relation> (<attribute>)
```

We will use three fundamental operations on the relations. The result of these operations are a new relation:

1. The *select*, denoted $\sigma_{\langle \text{condition} \rangle}(R)$, extracts tuples (rows) from the relation R . The result is all tuples satisfying the condition, and this is a subset of the population of R .
2. The *project*, denoted $\pi_{\langle \text{attributes} \rangle}(R)$, extracts attributes (columns) of the relation R . All tuples remain, but with fewer attributes.
3. The *Cartesian product*, denoted $R_1 \times R_2$, of the two relations R_1 and R_2 results a composite relation with all the attributes from both relations and all possible combinations of the tuples from the two relations.

The SELECT operation in SQL is a composition of these three operations. The syntax is:

```
SELECT <attributes>
FROM <relations>
WHERE <conditions>
```

Conceptually, the Cartesian product of all the relations combined are made. All tuples not satisfying <condition> are discarded, as well as attributes not in <attributes>. In relational algebra this expression is:

$$\sigma_{\langle \text{condition} \rangle} (\pi_{\langle \text{attributes} \rangle} (R_1 \times R_2 \times \cdots \times R_m)). \quad (2.9)$$

with all possible combinations of the tuples from the two relations. thus if relation A has n_A tuples, and relation B has n_B tuples, the Cartesian product has $n_A \times n_B$ tuples. Join operations are Cartesian products with a selection applied (thus, some of the tuple combinations are discarded).

2.3.2 The structure

The idea is to represent a mesh as a relational database, and then we can use select and project to extract different subsets of the mesh. In this way, we define exact a great variety of useful subsets. In fact, the mesh used for mesh queries does not be as strict as definition 2.2.2. In this way, mesh queries could be used to define the restrictions of this definition. However, this would result in a chicken-and-egg-issue (a mesh must be defined for the queries to be defined).

If the application is divided in a geometry kernel (which deals with handling the geometry) and an algorithms which use this library. The mesh query is a suitable candidate for defining a concise and clear interface between the kernel and the algorithms. G-maps is another such interface. Neither G-maps nor mesh queries impose a particular design for the kernel implementation, only a interface the kernel must satisfy.

The domains

We define three domains, the vertices, the edges and the faces. These domains are used to identify a vertex, edge or face. We use three relations (notice that these relations does not relate anything since they have only one single attribute each). The corresponding SQL-statements can be:

```
CREATE TABLE Vertices
  (vertex, int NOT NULL,
   PRIMARY KEY(vertex));

CREATE TABLE Edges
  (edge, int NOT NULL,
   PRIMARY KEY(edge));

CREATE TABLE Faces
  (faces, int NOT NULL,
   PRIMARY KEY(face));
```

These three relations define only three lists, the list of vertices, the list of edges and the list of faces:

Vertices	Edges	Faces
vertex	edge	face
v_1	e_1	f_1
\vdots	\vdots	\vdots

We will not dwell on how to populate these relations.

Relating elements of the domains

We use two relations to relate elements in the domains defined in the previous section. One relation, VertexEdge, relates vertices and edges, and the other, EdgeFace, relates edges and faces.

The VertexEdge relation has two attributes: vertex and edge. These two attributes are foreign keys of the single attributes of the Vertices and the Edges relations respectively. Each tuple in this relation is a “vertex is in edge” (or conversely, “edge has vertex”) statement.

The EdgeFace relation has two attributes: edge and face. These two attributes are foreign keys of the the single attribute in the Edges and Faces relations respectively. Each tuple in this relation is an “edge is in face” (or conversely, “face has edge”) statement. SQL-statements for the two relations can be:

```
CREATE TABLE VertexEdge
(vertex, int NOT NULL,
 edge,   int NOT NULL,
 PRIMARY KEY(vertex, edge),
 FOREIGN KEY(vertex) REFERENCES Vertices(vertex),
 FOREIGN KEY(edge) REFERENCES Edges(edge));

CREATE TABLE EdgeFace
(edge, int NOT NULL,
 face, int NOT NULL,
 PRIMARY KEY(edge, face),
 FOREIGN KEY(edge) REFERENCES Edges(edge),
 FOREIGN KEY(face) REFERENCES Faces(face));
```

Which creates the following structure:

VertexEdge		EdgeFace	
vertex	edge	edge	face
v_i	e_j	e_j	f_k
\vdots	\vdots	\vdots	\vdots

Here we see that v_i is one of the two edges of e_j , and e_j is one of the edges defining f_k . When combining the two relations we know that v_i is a vertex of f_k . These two relations contain all the connectivity information needed.

The virtual dart relation

As indicated in the previous section, we can combine the two relations. In this way we define the darts. The darts are a *virtual* relation. The population of this relation is deduced from other relations.

We find the darts as the Cartesian product of the VertexEdge and EdgeFace relations, followed by an select and project operation:

```
CREATE VIEW Darts
AS SELECT VertexEdge.vertex,
         VertexEdge.edge,
         EdgeFace.face
FROM   VertexEdge, EdgeFace
WHERE  VertexEdge.edge = EdgeFace.edge
```

A virtual relation (called “view” in SQL) can be used as a regular relation in queries. The SQL-statements produces the following structure:

Darts = $\pi_a(\sigma(\text{VertexEdge} \times \text{EdgeFace})_c)$		
vertex	edge	face
v_i	e_j	f_k
\vdots	\vdots	\vdots

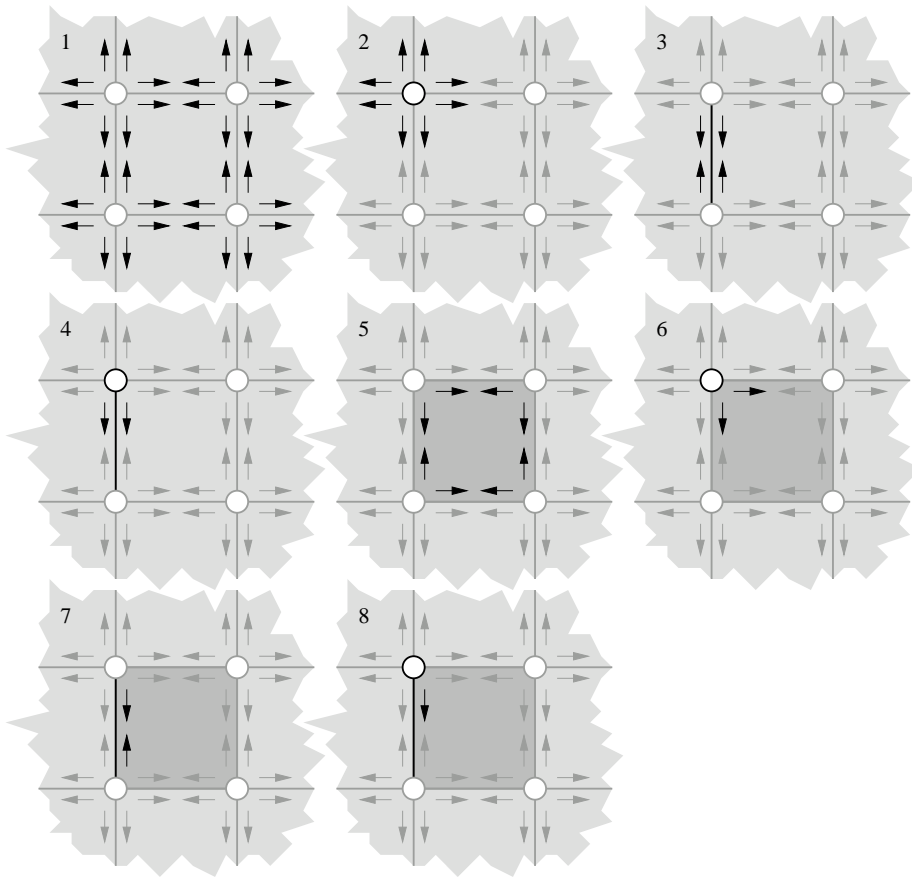


Figure 2.3: Dart queries. Enumeration refers to list given on page 22. A black circle is a selected vertex, a black edge is a selected edge and a dark face is a selected face.

where a are the attributes from the `SELECT` statement and c is the condition from the `WHERE` statement. The triple (v_i, e_j, f_k) is a dart in the mesh. The virtual dart relation is the core of the mesh queries.

2.3.3 Dart selection and projection

A mesh query is the result from a dart selection and a dart projection on the dart virtual relation.

Dart selection

We start with the selection part, which we call *dart queries*. The dart query defines the condition for the selection, and the query picks out the matching darts.

Definition 2.3.1. A *dart query* (v_i, e_j, f_k) is a triple of a vertex, an edge and a face where none, some or all of these three elements can be replaced by a wildcard denoted “*”. The dart query matches all darts which is composed of v_i , e_j and f_k or all vertices, edges or faces in the case of a wildcard.

In SQL the dart query (v_i, e_j, f_k) becomes

```
SELECT *
FROM   Darts
WHERE  vertex = v_i
      AND edge = e_j
      AND face = f_k;
```

If some elements of the query are wildcards, the corresponding WHERE statements are removed. Thus, the dart query $(v_i, *, *)$ becomes

```
SELECT *
FROM   Darts
WHERE  vertex = v_i;
```

There is in fact eight classes of these queries (refer to figure 2.3):

1. $(*, *, *)$ selects all darts of a mesh.
2. $(v_i, *, *)$ selects all darts around vertex v_i (it is the 0-orbit of a G-map).
3. $(*, e_j, *)$ selects all darts around edge e_j (it is the 1-orbit of a G-map).
4. $(v_i, e_j, *) \equiv (v_i, *, *) \cap (*, e_j, *)$ selects all darts around edge e_j at the side of v_i .
5. $(*, *, f_k)$ selects all darts of face f_k (it is the 2-orbit of a G-map).
6. $(v_i, *, f_k) \equiv (v_i, *, *) \cap (*, *, f_k)$ selects all darts of face f_k at v_i .
7. $(*, e_j, f_k) \equiv (*, e_j, *) \cap (*, *, f_k)$ selects all darts of face f_k along e_j .
8. $(v_i, e_j, f_k) \equiv (v_i, *, *) \cap (*, e_j, *) \cap (*, *, f_k)$ selects a single dart or none.

The queries 1,2,3 and 5 always match something. The other queries are called composite (since they can be written as an intersection) does not necessarily match something. For example, the query $(v_i, e_j, *)$ does not match anything unless v_i is one of the two edges of e_j . Thus, these tests can be used to check connectivity relations (if a vertex belongs to a face, an edge connects a vertex etc.).

Projection

A projection picks out certain attributes, and removes duplicate tuples in the result. In this case, the tuples are a vertex index, an edge index and a face index.

Definition 2.3.2. A *dart projection* is defined by a subset of the set $\{v, e, f\}$. The projection extracts some (or all) of the three attributes of a dart and removes duplicate tuples.

There exist seven projections of darts (in fact there are eight, but the void projection is completely useless):

1. $\{v\}$ picks vertices.
2. $\{e\}$ picks edges

3. $\{v, e\}$ picks vertex-edge combinations (known as edge-ends).
4. $\{f\}$ picks faces.
5. $\{v, f\}$ picks vertex-face combinations (known as corners).
6. $\{e, f\}$ picks edge-face combinations (known as half-edges)
7. $\{v, e, f\}$ picks vertex-edge-face combinations (known as darts).

Each of these projections has a name (like corner or half-edge). We use these terms when indexing the mesh (“for each corner in the mesh”, “for each half-edge”, “for each vertex”).

Combining projection and selection

We combine a dart selection with a successive dart projection on the dart relation to create a mesh query.

Definition 2.3.3. We *query* the mesh \mathfrak{M} with the following notation

$$\mathfrak{M}[\text{projection}; \text{selection}], \quad (2.10)$$

where *selection* is a dart selection (defined in definition 2.3.1) selecting a subset of darts and *projection* is a dart projection (defined in definition 2.3.2) to extract the desired kind of unique tuples.

The result is a set of tuples satisfying the requirement of the selection, with the attributes specified by the projection. In SQL the mesh query is the following select statement:

```
SELECT DISTINCT <projection>
      FROM Darts
      WHERE <selection>;
```

Mesh query example

We conclude this chapter with some examples of mesh queries.

Example 2.3.1. We use the mesh defined in example 2.2.1. The sets of vertices (equation 2.2), edges (equation 2.3) and faces (equation 2.4) can be found by the following mesh queries:

$$\begin{aligned} \mathbb{V} &= \mathfrak{M}[v; *, *, *], \\ \mathbb{E} &= \mathfrak{M}[e; *, *, *] \text{ and} \\ \mathbb{F} &= \mathfrak{M}[f; *, *, *]. \end{aligned} \quad (2.11)$$

Further, we may find the set of all the corners \mathbb{C} (equation 2.8):

$$\mathbb{C} = \mathfrak{M}[v, f; *, *, *]. \quad (2.12)$$

The faces abutting v_1 :

$$\{f_1, f_2, f_3\} = \mathfrak{M}[f; v_1, *, *]. \quad (2.13)$$

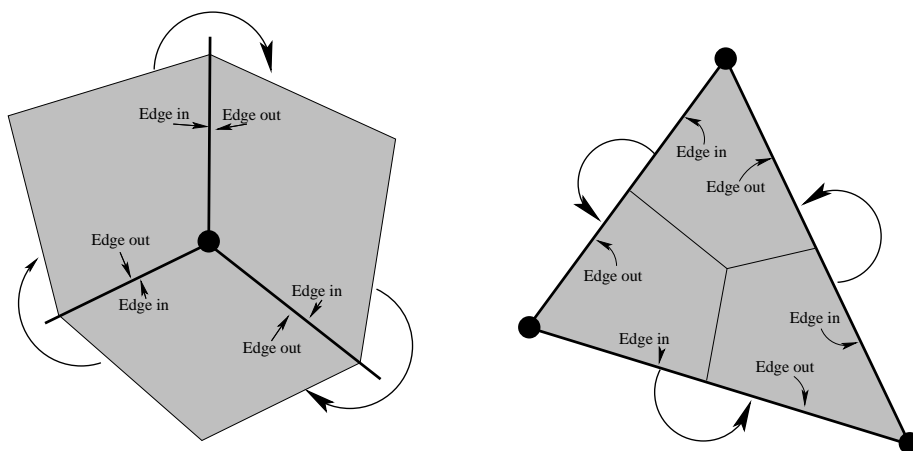


Figure 2.4: Consecutive ordering of corners around a vertex (left) and of a face (right). The edge in of one corner must be equal to the edge out of the next corner.

The vertices of edge e_1 :

$$\{v_1, v_2\} = \mathfrak{M}[v; *, e_1, *]. \quad (2.14)$$

We can define integrity constraints. The condition

$$0 < \#\mathfrak{M}[f; *, e_j, *] \leq 2, \quad \forall e_j \in \mathbb{E} \quad (2.15)$$

requires all edges to be used, and by two faces at most. Further,

$$0 \leq \#\mathfrak{M}[e; v_i, *, *] - \#\mathfrak{M}[f; v_i, *, *] \leq 1, \quad \forall v_i \in \mathbb{V} \quad (2.16)$$

requires a simple boundary.

2.4 Mesh modification and refinement

A refinement procedure is always employed in subdivision surfaces or surface splines. The two basic operations are known under various names, but we refer to them as the *face split* and the *vertex split*. Before diving into the details of the different schemes, we define a property, which we call *consecutive ordering of corners* (see figure 2.4). This definition is just convenient when describing the splitting procedures.

Definition 2.4.1. A *consecutive set of corners* is an offset invariant ordered set where the edge out of one corner is equal to the incoming edge of the next corner.

In addition, we assume that all meshes are *closed* in this section. Thus, we do not present any rules for dealing with boundary elements.

2.4.1 Dual mesh

A mesh has a dual where faces and vertices occupy complimentary locations — analogous to duals of polyhedra. A point can be defined by the intersection of some planes, and a plane can be defined by some points. In this sense, the point and the plane have

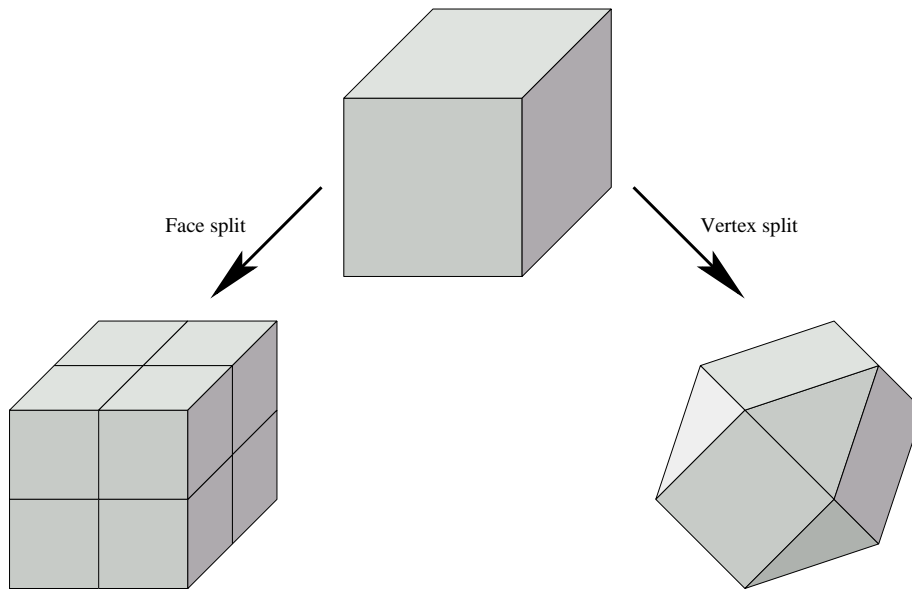


Figure 2.5: The two major splitting approaches. Face split (lower left): Each n -gon is split into n quadrilaterals. Vertex split (lower right): Each vertex of n -degree is removed and replaced with an n -gon.

a relationship of duality. The operations of defining these two kinds of objects are regarded as complementary dual operations. The dual of a polyhedron is defined by the exchange of these two dual elements.

Definition 2.4.2. *Vertices and faces are dual elements. Let a vertex be defined by abutting faces, and a face defined by vertices along the perimeter. These two operations are dual operations. The dual of a mesh is a mesh built with the dual elements and dual operations of this mesh.*

In this way, the vertices of one mesh corresponds to the faces of it's dual and vice versa. We can find the dual B of a mesh A by defining a face for each vertex and defining a vertex for each face: For each face in A , we define a vertex in B . For each vertex in A , we extract the fan of abutting corners in consecutive order, and use the vertices in B corresponding to the face of the corners to define a new face in B . Each pair of adjacent corners in this ordering defines an edge between the vertices of their respective faces.

We see from the definition that the corner of a mesh represents a kind of "pivot" between the duals. *The number of corners remains constant.* The dual mesh of the dual mesh is the mesh itself. We use the dual mesh of the quad mesh to define the *inner mesh* for the simplified surface spline.

2.4.2 Face split (polyhedral split)

Most subdivision schemes can be split into two operations: a splitting step and a smoothing step. The splitting step refines the mesh, that is, it produces a new mesh with more and smaller faces. Ideally, the splitting step should not change the geometry of the shape. This is the case if the embedded mesh is a polyhedron (which has planar

faces). The smoothing step generates a new set of points, where each point is an average of some of the points produced by the splitting step. We ignore the smoothing step for now, and focus on the splitting step. For more information on both steps, (Stollnitz, DeRose & Salesin 1996) is a good source.

We begin with a subdivision scheme known as both the “face split” and the “polyhedral split”. This scheme splits each face into several new faces. We call the original mesh \mathfrak{M}^i and the refined mesh \mathfrak{M}^{i+1} .

Vertex and point generation First, we define the vertices of \mathfrak{M}^{i+1} and their positions.

1. For each vertex in \mathfrak{M}^i , create a vertex in \mathfrak{M}^{i+1} with the same location.
2. For each edge in \mathfrak{M}^i , create a vertex in \mathfrak{M}^{i+1} with the average of the endpoints of the edge as location.
3. For each face in \mathfrak{M}^i , create a new vertex in \mathfrak{M}^{i+1} with the average of the vertices of the face as location.

The number of vertices in the new mesh is equal to the sum of vertices, edges, and faces of the old mesh.

Defining the new edges The edges are defined implicitly by the face descriptions, but for completeness, we define them here.

1. For each edge in \mathfrak{M}^i , create two edges in \mathfrak{M}^{i+1} . These connect to the vertices in \mathfrak{M}^{i+1} corresponding to the end points of the edge as well as the edge midpoint.
2. For each edge in \mathfrak{M}^i , create an edge in \mathfrak{M}^{i+1} for each abutting face. This edge connects the edge midpoint to the face midpoint.

The number of edges is equal to the total sum of corners in the old mesh plus two times the number of edges in the old mesh. If the mesh is closed, as we assume, the number of corners are $2 \cdot \#\mathbb{E}$, and thus the total number of new edges are $4 \cdot \#\mathbb{E}$.

Defining the new faces For each corner in \mathfrak{M}^i , create a quadrilateral in \mathfrak{M}^{i+1} with the edges:

1. The edge from the vertex of the corner to the midpoint of the outgoing edge.
2. The edge from the midpoint of the outgoing edge to the face midpoint.
3. The edge from the face midpoint to the midpoint of the incoming edge.
4. The edge from the incoming edge to the vertex of the corner.

The number of faces in the new mesh is equal to the total sum of corners in the old mesh. The subdivision scheme is visualised in figure 2.5. Meshes generated by the face split scheme possess some useful properties:

1. All faces are quadrilaterals.
2. The vertices from \mathfrak{M}^i keep their degree.

3. The degree of a midpoint of an n -gon in \mathcal{M}^i , has a degree of n . Thus, midpoints of quadrilaterals are of degree 4.
4. The degree of edge midpoints are 4.

Thus, after one application of this scheme the quads of the refined mesh may have more than one vertex of degree not equal to 4.

If all vertices in the original mesh with a degree other than 4 are only surrounded by quadrilaterals, or the converse, every non-quadrilateral face in the mesh are defined only by vertices with degree 4, *a quad in the refined mesh have at most one vertex with degree not equal to 4*. However, if these prerequisites are not fulfilled, two successive applications of the splitting procedure always generate a refined mesh with this property.

2.4.3 Vertex split (mid-edge subdivision)

The face split splits faces, and the vertex split splits, as the name implies, vertices. The vertex split (also known as mid-edge subdivision) is very simple, and the scheme proceeds as follows:

Vertex and point generation For each edge in \mathcal{M}^i , create a vertex in \mathcal{M}^{i+1} and position this vertex at the midpoint of the edge. We call this point the *midpoint of the edge*.

Defining the new edges For each vertex in \mathcal{M}^i , find the set of corners abutting the vertex. Create an edge in \mathcal{M}^{i+1} for each of these corners, connecting the midpoint of the incoming edge to the midpoint of the outgoing edge.

Defining the new faces Each face and each vertex in \mathcal{M}^i generates a face in \mathcal{M}^{i+1} .

1. For each vertex, extract the corners abutting that vertex and organise them as a consecutive set of corners. Then define a face by the edges connecting the midpoints of the incoming and outgoing edge each corner.
2. For each face in the unrefined mesh, extract the corners and organise them and build a face as done in the previous step.

The number of vertices in the new mesh is thus equal to the number of edges in the old mesh, the number of edges in the new mesh is equal to the number of corners in the old mesh, and the number of faces in the new mesh is the sum of the faces in the old mesh plus the number of vertices in the old mesh. The refined mesh has some interesting properties:

1. All vertices of the refined mesh have a degree of 4 (two faces and two vertices abut an edge).
2. An n -gon in the unrefined mesh becomes an n -gon in the refined mesh.
3. A vertex of degree n becomes an n -gon in the refined mesh.

2.4.4 The Doo-Sabin split

The Doo-Sabin split is a bit more elaborate than the two previous subdivision schemes. It splits both vertices and edges, while keeping faces. The algorithm proceeds as follows:

Vertex generation For each corner in \mathfrak{M}^i , create a vertex in \mathfrak{M}^{i+1} . The position of this vertex is defined by a linear interpolation between the position of the corner's vertex (in \mathfrak{M}^i) and the midpoint of the face (average of the face's vertices in \mathfrak{M}^i). The number of vertices in \mathfrak{M}^{i+1} equals the number of corners in \mathfrak{M}^i . The parameter of this linear interpolation defines the sharpness (analogous to blend ratios) when building Doo-Sabin subdivision surfaces.

Defining the new edges Each edge in \mathfrak{M}^i is abutted by four corners. We connect the midpoints (vertices in \mathfrak{M}^{i+1}) with four edges and the result is a quadrilateral centred on the edge in \mathfrak{M}^i . The number of edges in \mathfrak{M}^{i+1} is four times the number of edges in \mathfrak{M}^i .

Defining the new faces Each vertex, each edge, and each face define a face in the new mesh. The midpoint of the abutting corners define a new face, thus, a vertex in \mathfrak{M}^i of valence n defines an n -gon in \mathfrak{M}^{i+1} . Each edge will form a quadrilateral, while each face will keep their number of vertices. The number of faces in \mathfrak{M}^{i+1} equals the sum of the number of vertices, the number of edges, and the number of faces in \mathfrak{M}^i .

There is a connection between the polyhedral split and the Doo-Sabin split. Both schemes produces the same number of edges, and the Doo-Sabin split produces the same number of vertices as the number of the polyhedral split produces faces, and vice versa. This is indeed a hint that the results from the polyhedral split and the Doo-Sabin split are duals of each other.

For a more extensive exposition of splitting procedures, as well as subdivision surfaces, some suggested books are (Farin 1996), (Gallier 2000) and (Stollnitz et al. 1996).

2.4.5 Planar cut polyhedra

Planar cut polyhedra (PCP) is a central component of surface spline schemes. Often, the term “polyhedron” is used quite loosely about anything built from polygonal faces. However, for such a shape to be a polyhedron, it is required that every face is planar. This is of course not an issue when dealing with triangles, since they are planar when not degenerate. To emphasise this planarity, we call such a shape a *planar polyhedron* when this is of importance.

We get a *planar cut polyhedron* from a planar polyhedron by chopping off the shape at each vertex with a plane. The amount of the shape cut away is called the depth of the cut. The planar cut polyhedron can be viewed as a generalisation of the tensor product control structure, where each planar cut represents a subdivision step.

If an embedded mesh is equivalent regarding connectivity (and not geometry) to a planar cut polyhedron, we can sometimes make this shape into a planar cut polyhedron by utilising some local projections. This is always possible if the mesh is *local degree bounded*. This requires that at most one non-planar face abuts a vertex. If two or more offending faces abut, then we get a dependency in the projections, which are not necessarily solvable (one projection destroys the planarity of the other).

Definition 2.4.3. *A polyhedron is **local degree bounded** if all non-quadrilateral faces of a polyhedron is only defined from vertices where only four faces abut, or conversely, all vertices surrounded by more or less than four faces is only surrounded by quadrilateral faces.*

If a polyhedron is not local degree bounded, it can be made local degree bounded by refinement.

2.5 Summary

In this chapter we have defined the mesh. Meshes are a formal way to define connectivity of geometric objects. Vertices are the fundamental atom of meshes, an edge is a set of two vertices and faces are sets of edges. We also defined some terms of positions in the connectivity.

Then we gave the background and definition of the mesh queries. The qualities of mesh queries will be obvious in chapters 5 and 6, in which mesh queries are used extensively.

We concluded the chapter with an overview of mesh duals and some splitting schemes. Both of these topics are essential in the construction of surface splines.

Chapter 3

The surface spline curve

What happens if we apply the surface spline paradigm to curves? Surface spline curves are smooth, modelled by the control polygon, can interpolate data points with some modifications, and have a simple construction. A study of the simple case of the curve is a gentle introduction to the surface spline paradigm, and in addition, this curve defines the boundary curves of a simplified surface spline surface.

3.1 The scheme

Surface splines have a peculiar parameterisation, and the same is true for surface spline curves. The curves are parameterised by two numbers: an integer index, and a scalar running from zero to one. The index specifies in which sub-interval we are in, and the scalar specifies the position in this sub-interval. For visualisation of the construction we arrange the control points along the real axis.

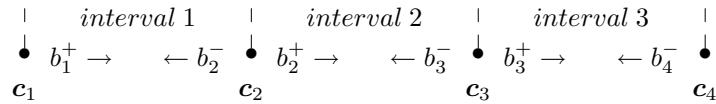
Definition 3.1.1. A surface spline curve is defined by a vector of m control points,

$$\mathbf{c} = [\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m], \quad \mathbf{c}_i \in \mathbb{R}^n, \quad (3.1)$$

and a vector of $2m - 2$ blend ratios,

$$\mathbf{b} = [b_1^+, b_2^-, b_2^+, \dots, b_{m-1}^-, b_{m-1}^+, b_m^-], \quad b_i \in [0, 1], \quad (3.2)$$

organised in the following manner:



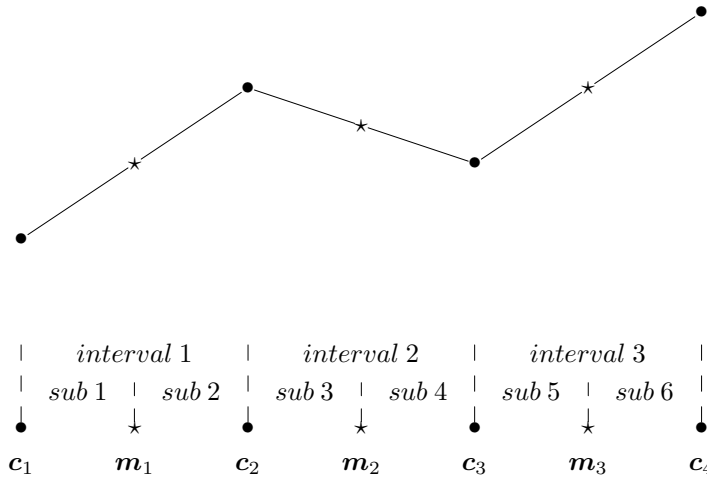
Each interior \mathbf{c}_i has two blend ratios associated with it, b_i^- and b_i^+ , and \mathbf{c}_1 and \mathbf{c}_m has one blend ratio each. The control point array defines $m - 1$ intervals, one interval between each successive pair of control points: Interval i is between \mathbf{c}_i and \mathbf{c}_{i+1} . Each interval defines two sub-intervals, one odd and one even. The odd sub-interval is the left half of the interval, and the even sub-interval is the right half. Blend ratios and sub-intervals have a one-to-one correspondence, and use the same indices.

3.1.1 Refinement

For every interval i we introduce a midpoint, m_i defined by

$$m_i = 1/2(c_i + c_{i+1}). \quad (3.3)$$

These midpoints split every interval into the two sub-intervals.



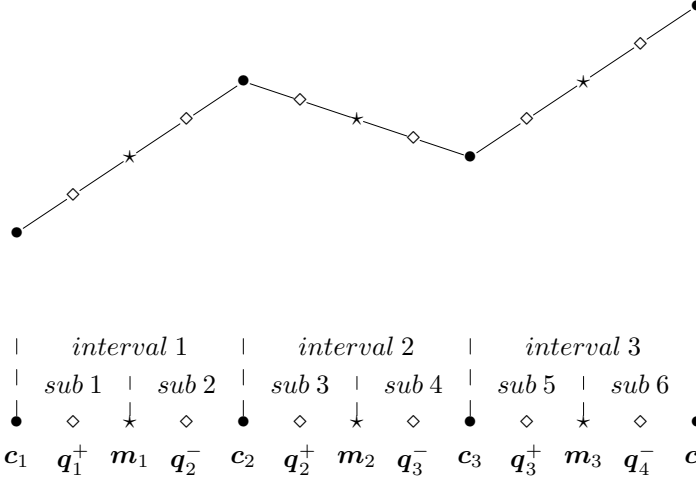
We have so far done nothing extraordinary. The midpoints are the averages of the control points at the extremities of the intervals, and the geometry of the curve is not changed.

3.1.2 Sub-interval midpoints

We introduce yet another level of midpoints, the $2m - 2$ sub-interval midpoints. These are just logical midpoints, not necessarily geometrical. For each sub-interval i we define the sub-interval midpoint q_i as

$$\begin{aligned} q_i^+ &= b_i^+ c_i + (1 - b_i^+) m_i \\ q_i^- &= b_i^- c_i + (1 - b_i^-) m_{i-1} \end{aligned} \quad (3.4)$$

Each inner control point and interval midpoint is surrounded by two sub-interval midpoints. We get a constellation looking like



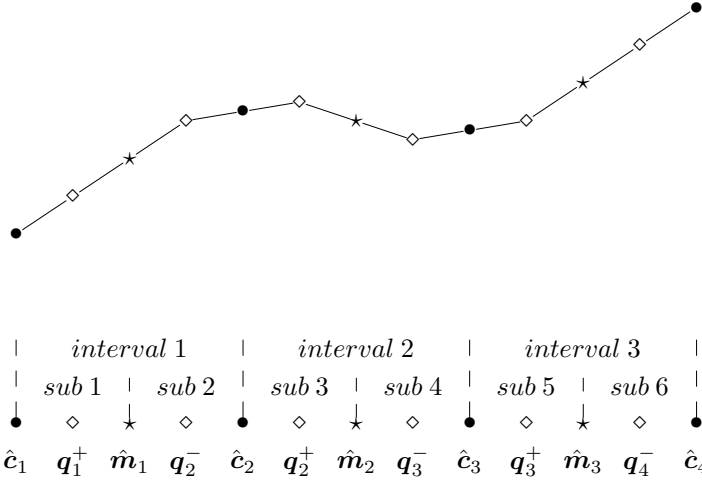
If all blend ratios are equal to $\frac{1}{2}$, the q 's are the geometric averages of their two respective neighbours. If blend ratios are small, the sub-interval midpoints are located closer to the control points, and conversely, when blend ratios are large, the sub-interval midpoints are located closer to the interval midpoints. Regardless of blend ratios, the sub-interval midpoints are just convex combinations of an interval midpoint and a control point, hence it lies on the line between these two points. The geometry is not changed in this step either.

3.1.3 Bézier coefficients

Then we change the geometry; we generate new positions for the control points and the interval midpoints. The collection of new control points, new interval midpoints and the sub-interval midpoints form the Bézier coefficients of the curve. The new control points are denoted \hat{c}_i and the new interval midpoints \hat{m}_i , and are defined by the averages

$$\begin{aligned}
 \hat{c}_1 &= c_1 \\
 \hat{c}_i &= (q_i^- + q_i^+)/2 \quad i = 2, \dots, m-1 \\
 \hat{c}_m &= c_m \\
 \hat{m}_i &= (q_i^+ + q_{i+1}^-)/2 \quad i = 1, \dots, m-1
 \end{aligned} \tag{3.5}$$

If we connect each subsequent coefficient with a line, we get



This step is important. If the last control point of one segment equals the first control point of another segment, the two segments join continuously (C^0). In addition, if the next-to-last and last control point of the first segment are coplanar with the first and second control point of the second segment, they joint tangent continuously (G^1). Even further, if the distances between the next-to-last and last control points of the first segment equals the distance between the first and second control point of the second segment, the derivatives of the two segments match, and thus the segments join C^1 .

And the equation 3.5 makes the curve satisfy these requirements. The geometry is changed in this step; we get a smoothed version of the original control curve, though it interpolates the control curve at the first and at the last control points.

3.1.4 Defining the Bézier curve segments

Each sub-interval defines a quadratic Bézier curve segment, denoted $s_i(u)$. Each of these curve segments are defined by three points.

$$s_i(u) = \begin{cases} B_0^2(u)c_j + B_1^2(u)q_i^+ + B_2^2(u)m_j, & j = 1 + (i - 1)/2 \text{ if } i \text{ is odd} \\ B_0^2(u)m_j + B_1^2(u)q_i^+ + B_2^2(u)c_{j+1}, & j = i/2 \text{ if } i \text{ is even,} \end{cases} \quad (3.6)$$

where $B_{0,1,2}^2$ are the Bernstein polynomials defined in definition 1.4.10. This produces the curve

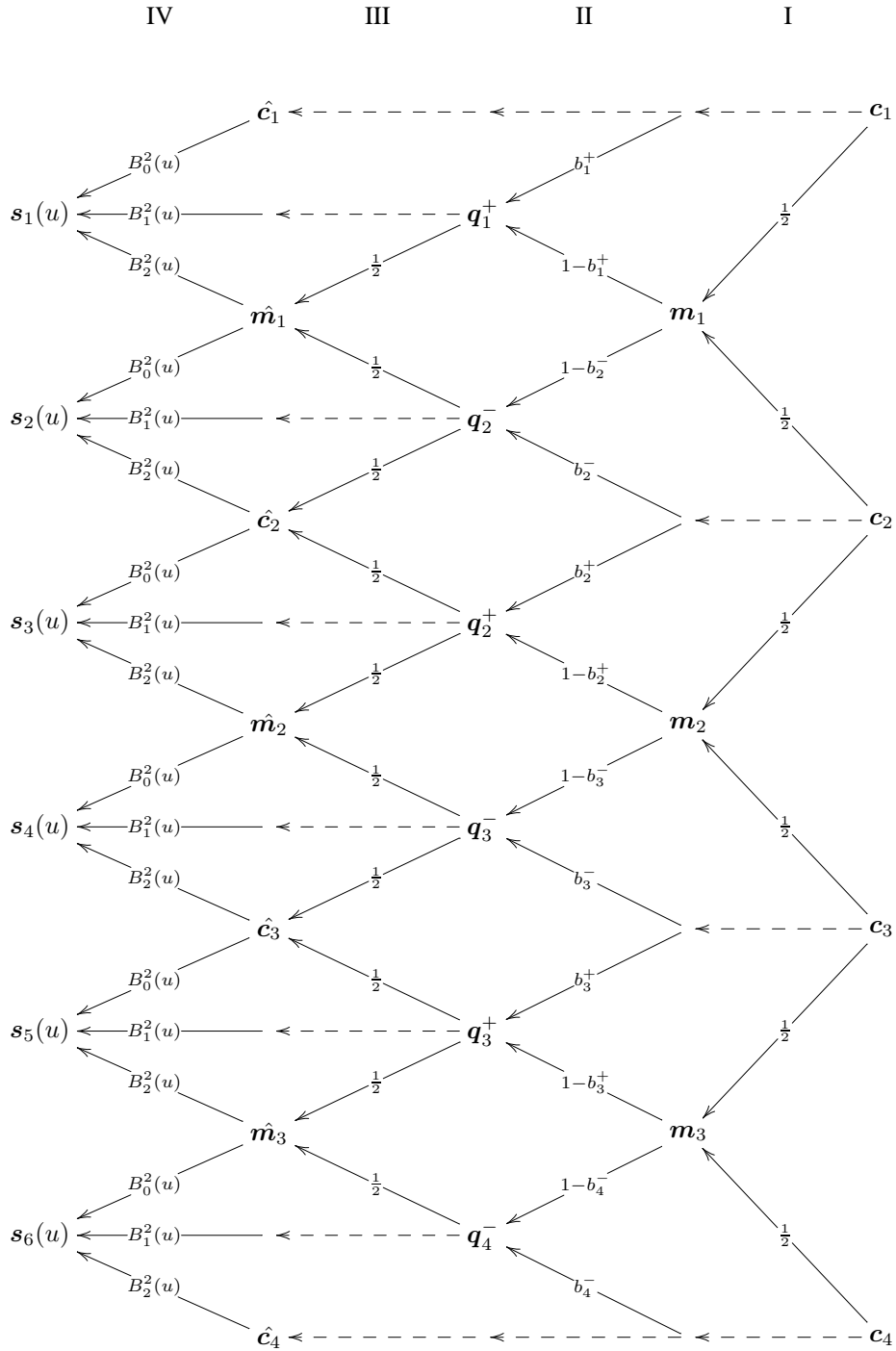


Figure 3.1: Layout of surface spline curve construction

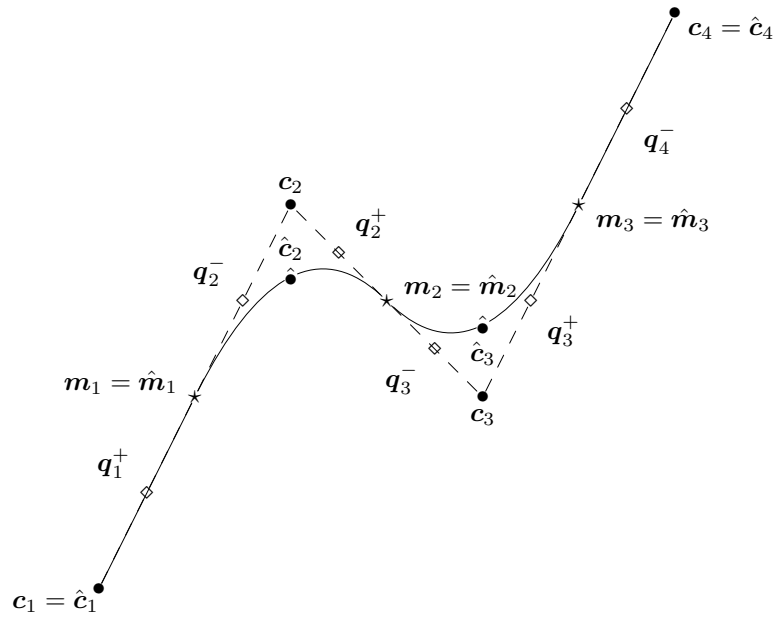
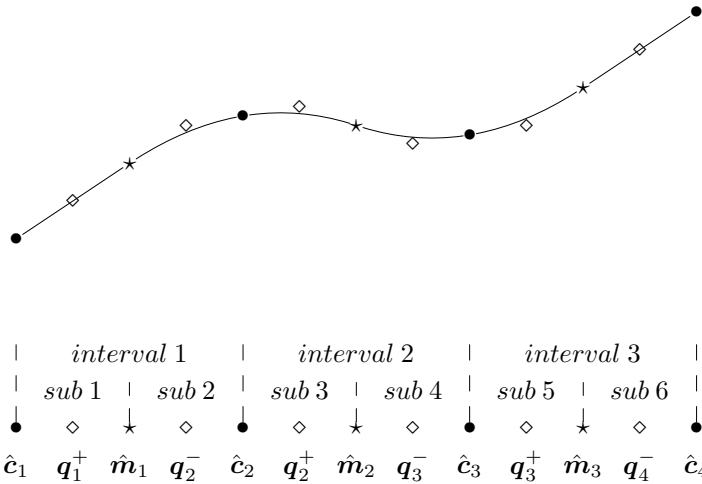


Figure 3.2: The surface spline curve of example 3.1.1.



3.1.5 Summary of construction

A systematic overview of the complete scheme is given in figure 3.1. The scheme consists of four distinct stages (from right to left). At stage I, the interval midpoints are created. At stage II the subinterval midpoints are created, utilising the blend ratios. At stage III smoothing is performed to assure the C^1 continuity requirement of composite Bézier curves, and at stage IV conventional quadratic Bézier evaluation is done. The dashed lines represent “keeping a value” to be used later in the scheme.

Example 3.1.1. Find the surface spline curve defined by the four points in the plane,

$$\mathbf{c} = [\mathbf{c}_1 \quad \mathbf{c}_2 \quad \mathbf{c}_3 \quad \mathbf{c}_4] = \left[\begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 3 \\ 3 \end{bmatrix} \right], \quad (3.7)$$

and the blend ratios

$$\begin{aligned} \mathbf{b} &= [b_1^+ \quad b_2^- \quad b_2^+ \quad b_3^- \quad b_3^+ \quad b_4^-] \\ &= \left[\frac{1}{2} \quad \frac{1}{2} \quad \frac{1}{2} \quad \frac{1}{2} \quad \frac{1}{2} \quad \frac{1}{2} \right]. \end{aligned} \quad (3.8)$$

First, we find the interval midpoints

$$\begin{aligned} \mathbf{m} &= [\mathbf{m}_1 \quad \mathbf{m}_2 \quad \mathbf{m}_3] \\ &= \left[\frac{1}{2}(\mathbf{c}_1 + \mathbf{c}_2) \quad \frac{1}{2}(\mathbf{c}_2 + \mathbf{c}_3) \quad \frac{1}{2}(\mathbf{c}_3 + \mathbf{c}_4) \right] \\ &= \left[\begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1\frac{1}{2} \\ 1\frac{1}{2} \end{bmatrix} \quad \begin{bmatrix} 2\frac{1}{2} \\ 2 \end{bmatrix} \right]. \end{aligned} \quad (3.9)$$

We then find the sub-interval midpoints, and since all blend ratios are equal to one half, these become simple averages,

$$\begin{aligned} \mathbf{q} &= [\mathbf{q}_1^+ \quad \mathbf{q}_2^- \quad \mathbf{q}_2^+ \quad \mathbf{q}_3^- \quad \mathbf{q}_3^+ \quad \mathbf{q}_4^-] \\ &= \begin{bmatrix} b_1^+ \mathbf{c}_1 + (1 - b_1^+) \mathbf{m}_1 \\ (1 - b_2^-) \mathbf{m}_1 + b_2^- \mathbf{c}_2 \\ b_2^+ \mathbf{c}_2 + (1 - b_2^+) \mathbf{m}_2 \\ (1 - b_3^-) \mathbf{m}_2 + b_3^- \mathbf{c}_3 \\ b_3^+ \mathbf{c}_3 + (1 - b_3^+) \mathbf{m}_3 \\ (1 - b_4^-) \mathbf{m}_3 + b_4^- \mathbf{c}_4 \end{bmatrix}^T \\ &= \begin{bmatrix} \frac{1}{2}(\mathbf{c}_1 + \mathbf{m}_1) \\ \frac{1}{2}(\mathbf{m}_1 + \mathbf{c}_2) \\ \frac{1}{2}(\mathbf{c}_2 + \mathbf{m}_2) \\ \frac{1}{2}(\mathbf{m}_2 + \mathbf{c}_3) \\ \frac{1}{2}(\mathbf{c}_3 + \mathbf{m}_3) \\ \frac{1}{2}(\mathbf{m}_3 + \mathbf{c}_4) \end{bmatrix}^T \\ &= \left[\begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \end{bmatrix} \quad \begin{bmatrix} \frac{3}{4} \\ 1\frac{1}{2} \end{bmatrix} \quad \begin{bmatrix} 1\frac{1}{4} \\ 1\frac{3}{4} \end{bmatrix} \quad \begin{bmatrix} 1\frac{3}{4} \\ 1\frac{1}{4} \end{bmatrix} \quad \begin{bmatrix} 2\frac{1}{4} \\ 1\frac{1}{2} \end{bmatrix} \quad \begin{bmatrix} 2\frac{3}{4} \\ 2\frac{1}{2} \end{bmatrix} \right]. \end{aligned} \quad (3.10)$$

and from this the smoothed control points,

$$\begin{aligned} \hat{\mathbf{c}} &= [\hat{\mathbf{c}}_1 \quad \hat{\mathbf{c}}_2 \quad \hat{\mathbf{c}}_3 \quad \hat{\mathbf{c}}_4] \\ &= [\mathbf{c}_1 \quad \frac{1}{2}(\mathbf{q}_2^- + \mathbf{q}_2^+) \quad \frac{1}{2}(\mathbf{q}_3^- + \mathbf{q}_3^+) \quad \mathbf{c}_4] \\ &= \left[\begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1\frac{5}{8} \end{bmatrix} \quad \begin{bmatrix} 2 \\ 1\frac{3}{8} \end{bmatrix} \quad \begin{bmatrix} 3 \\ 3 \end{bmatrix} \right], \end{aligned} \quad (3.11)$$

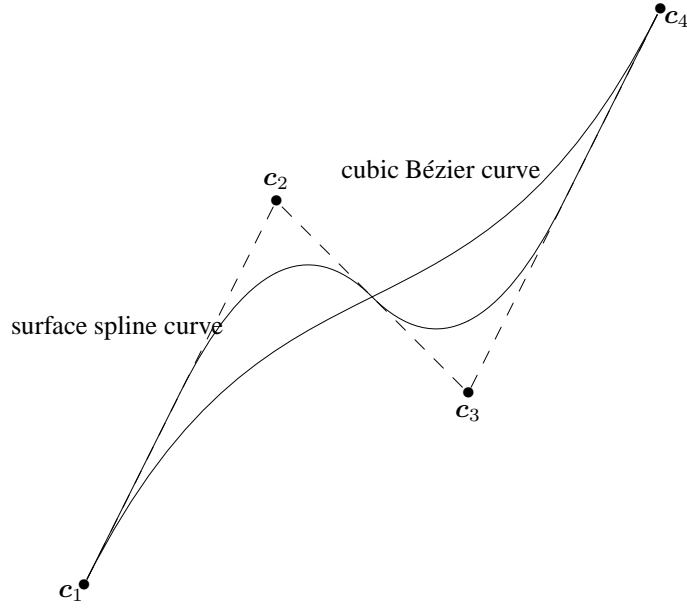


Figure 3.3: Comparison between a four-point surface spline curve and a cubic Bézier curve.

and the smoothed interval midpoints,

$$\begin{aligned}
 \hat{\mathbf{m}} &= [\hat{\mathbf{m}}_1 \quad \hat{\mathbf{m}}_2 \quad \hat{\mathbf{m}}_3] \\
 &= \left[\frac{1}{2}(\mathbf{q}_1^+ + \mathbf{q}_2^-) \quad \frac{1}{2}(\mathbf{q}_2^+ + \mathbf{q}_3^-) \quad \frac{1}{2}(\mathbf{q}_3^+ + \mathbf{q}_4^-) \right] \\
 &= \frac{1}{2} \begin{bmatrix} \left[\frac{1}{2} \right] & \left[1\frac{1}{2} \right] & \left[2\frac{1}{2} \right] \\ \left[1 \right] & \left[1\frac{1}{2} \right] & \left[2 \right] \end{bmatrix},
 \end{aligned} \tag{3.12}$$

and this defines six quadratic Bézier segments,

$$\begin{aligned}
 \mathbf{s}_1(u) &= B_0^2(u)\hat{\mathbf{c}}_1 + B_1^2(u)\mathbf{q}_1^+ + B_2^2(u)\hat{\mathbf{m}}_1 \\
 \mathbf{s}_2(u) &= B_0^2(u)\hat{\mathbf{m}}_1 + B_1^2(u)\mathbf{q}_2^+ + B_2^2(u)\hat{\mathbf{c}}_2 \\
 \mathbf{s}_3(u) &= B_0^2(u)\hat{\mathbf{c}}_2 + B_1^2(u)\mathbf{q}_3^+ + B_2^2(u)\hat{\mathbf{m}}_2 \\
 \mathbf{s}_4(u) &= B_0^2(u)\hat{\mathbf{m}}_2 + B_1^2(u)\mathbf{q}_4^+ + B_2^2(u)\hat{\mathbf{c}}_3 \\
 \mathbf{s}_5(u) &= B_0^2(u)\hat{\mathbf{c}}_3 + B_1^2(u)\mathbf{q}_5^+ + B_2^2(u)\hat{\mathbf{m}}_3 \\
 \mathbf{s}_6(u) &= B_0^2(u)\hat{\mathbf{m}}_3 + B_1^2(u)\mathbf{q}_6^+ + B_2^2(u)\hat{\mathbf{c}}_4
 \end{aligned} \tag{3.13}$$

The collection of these curve segments is the surface spline curve.

The curve is rather smooth and bears resemblance to a traditional spline. Figure 3.3 compares the surface spline curve to a cubic Bézier curve with the same coefficients. The surface spline curve is closer to the control polygon.

3.2 Explicit formulas

For such a simple scheme as the surface spline curve scheme it is easy to deduce an explicit formula for the curve.

Proposition 3.2.1. *The Bézier coefficients of the surface spline curve can be found by the expressions*

$$\begin{aligned}
\hat{c}_1 &= c_1 \\
\hat{c}_m &= c_m \\
\hat{c}_i &= \frac{1}{4}(1 - b_i^-)c_{i-1} + \frac{1}{2}\left(1 + \frac{1}{2}(b_i^+ + b_i^-)\right)c_i + \frac{1}{4}(1 - b_i^+)c_{i+1} \\
\hat{m}_i &= \frac{1}{2}\left(1 + \frac{1}{2}(b_i^+ - b_{i+1}^-)\right)c_i + \frac{1}{2}\left(2 + \frac{1}{2}(b_{i+1}^- - b_i^+)\right)c_{i+1} \\
q_i^+ &= \frac{1}{2}(1 + b_i^+)c_i + \frac{1}{2}(1 - b_i^+)c_{i+1} \\
q_i^- &= \frac{1}{2}(1 + b_i^-)c_i + \frac{1}{2}(1 - b_i^-)c_{i-1}.
\end{aligned} \tag{3.14}$$

The Bézier coefficients are convex combinations of the control points if the blend ratios are values in $[0, 1]$.

Proof. The formulas can easily be deduced from the construction (see figure 3.1). If the blend ratios are kept within $[0, 1]$, we see that none of the weights in the expressions become negative. Expanding the expressions, we see that the weights in each expression sum to one. \square

3.3 Exists a subdivision formula?

The existence of a simple subdivision formula considerably increases the value of a curve or surface algorithm. The ‘‘LeSS’’ (Gonzalez & Peters 1999) construction (see section 4.2.2) does not give nested spaces. However, it is worthwhile to see whether this is possible.

3.3.1 Bézier segment subdivision

Our starting point will be the refinement formulas for Bézier segments. From the literature (e.g. (Farin 1996) and (Gallier 2000)) we know that we can rewrite one single Bézier curve as two new ones.

We split the quadratic Bézier segment defined by the Bézier control points $\{x_1, x_2, x_3\}$ into two equal halves, the two segments defined by the Bézier control points $\{\hat{x}_{1,1}, \hat{x}_{1,2}, \hat{x}_{1,3}\}$ and $\{\hat{x}_{2,1}, \hat{x}_{2,2}, \hat{x}_{2,3}\}$

$$\begin{aligned}
\hat{x}_{1,1} &= x_1 \\
\hat{x}_{1,2} &= \frac{1}{2}x_1 + \frac{1}{2}x_2 \\
\hat{x}_{1,3} &= \frac{1}{4}x_1 + \frac{1}{2}x_2 + \frac{1}{4}x_3,
\end{aligned} \tag{3.15}$$

and

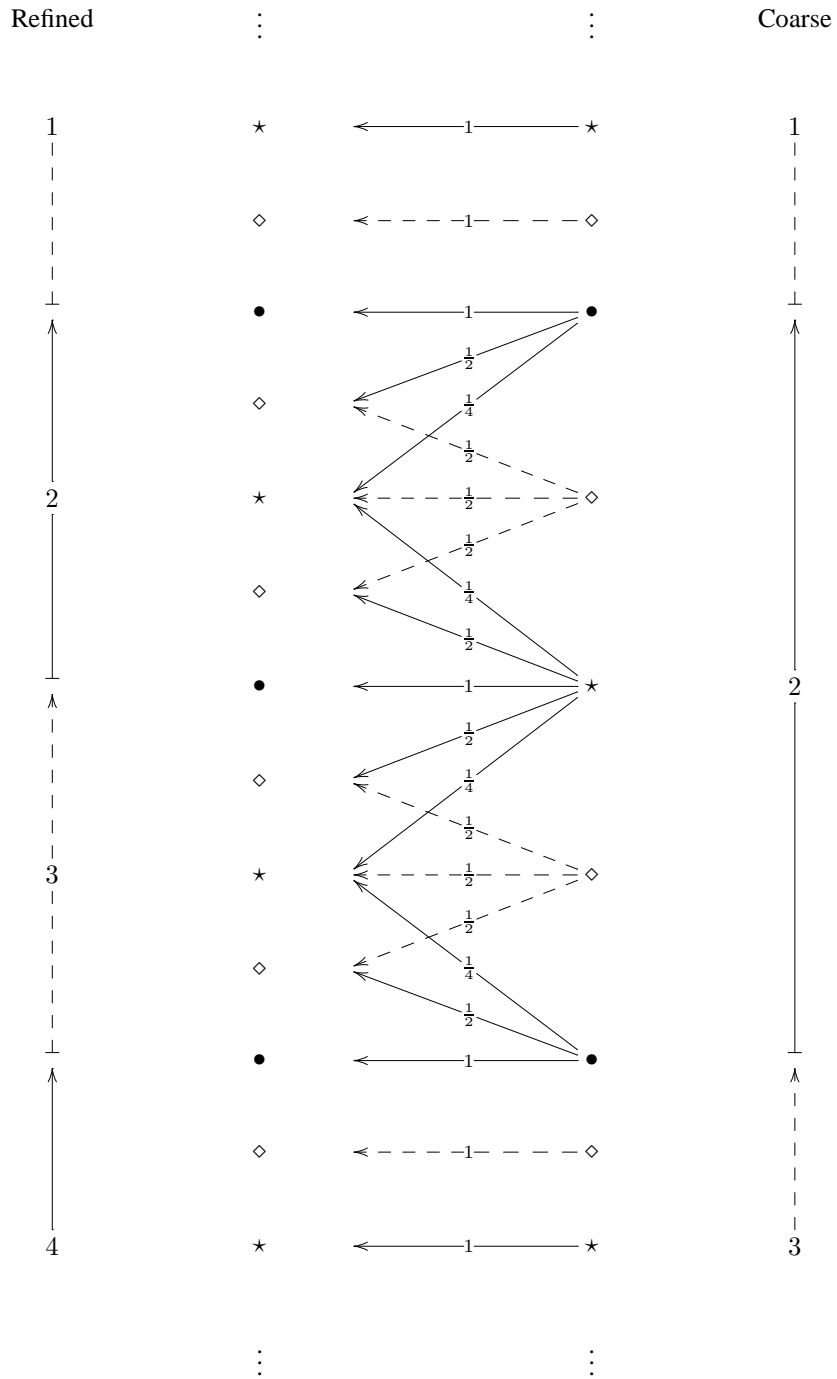


Figure 3.4: Inserting one control point.

$$\begin{aligned}
\hat{\mathbf{x}}_{2,1} &= \frac{1}{4}\mathbf{x}_1 + \frac{1}{2}\mathbf{x}_2 + \frac{1}{4}\mathbf{x}_3 \\
\hat{\mathbf{x}}_{2,2} &= \frac{1}{2}\mathbf{x}_2 + \frac{1}{2}\mathbf{x}_3 \\
\hat{\mathbf{x}}_{2,3} &= \mathbf{x}_3.
\end{aligned} \tag{3.16}$$

This will use this relation of Bézier segments to relate the coarse and the refined surface spline curve.

3.3.2 Surface spline subdivision

To keep things simple, we let the blend ratios equal one half, and we write the problem as a matrix equation. We let \mathbf{c}^i be the control points of the surface spline curve at subdivision level i , and \mathbf{c}^{i+1} be the control points at level $i + 1$.

The matrix \mathbf{A} finds the Bézier coefficients from the surface spline curve control points. Thus, \mathbf{A} represents the steps “refinement”, “sub-interval midpoints”, and “Bézier coefficients” combined. We let \mathbf{A}^i be the matrix that transforms the control points at level i to the Bézier coefficients at level i , and \mathbf{A}^{i+1} be the similar matrix at level $i + 1$.

The matrix \mathbf{R}^i refines the Bézier control points at level i to Bézier control points at level $i + 1$ (this expression is visualised in figure 3.4). This gives us

$$\mathbf{A}^{i+1}\mathbf{c}^{i+1} = \mathbf{R}^i\mathbf{A}^i\mathbf{c}^i, \tag{3.17}$$

which would give

$$\mathbf{c}^{i+1} = \underbrace{(\mathbf{A}^{i+1})^{-1}\mathbf{R}^i\mathbf{A}^i}_{\text{subdivision rule}}\mathbf{c}^i, \tag{3.18}$$

the subdivision rule we are looking for. However, \mathbf{A} is not square, hence this rule does not necessarily exist. We will not try to find $(\mathbf{A}^{i+1})^{-1}\mathbf{R}^i\mathbf{A}^i$, only see if this expression in fact exists.

3.3.3 Futile attempt no. 1: inserting one point

Initially, let’s try to insert *one* single control point into the coarse curve. *Given a curve with n control points, exists a curve with $n + 1$ control points having the exact same geometry?* That is, does the curve with n control points live in the space of the curve with $n + 1$ control points?

Then, let us start with the simplest scenario. Let the coarse curve consist of two control points. We want to refine this curve by inserting a new control point into the single interval of this curve. Using the relation

$$\mathbf{A}^{i+1}\mathbf{x}^{i+1} = \mathbf{R}\mathbf{A}^i\mathbf{c}^i, \tag{3.19}$$

we get

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{3}{4} & \frac{1}{4} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & \frac{3}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{c}_1^{i+1} \\ \mathbf{c}_2^{i+1} \\ \mathbf{c}_3^{i+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{3}{4} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{3}{4} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{c}_1^i \\ \mathbf{c}_2^i \end{bmatrix}. \quad (3.20)$$

We multiply together the matrices on the right-hand side

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{3}{4} & \frac{1}{4} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{3}{4} & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & \frac{3}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{c}_1^{i+1} \\ \mathbf{c}_2^{i+1} \\ \mathbf{c}_3^{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{c}_1^i \\ \frac{7}{8}\mathbf{c}_1^i + \frac{1}{8}\mathbf{c}_2^i \\ \frac{3}{4}\mathbf{c}_1^i + \frac{1}{4}\mathbf{c}_2^i \\ \frac{5}{8}\mathbf{c}_1^i + \frac{3}{8}\mathbf{c}_2^i \\ \frac{1}{2}\mathbf{c}_1^i + \frac{1}{2}\mathbf{c}_2^i \\ \frac{3}{8}\mathbf{c}_1^i + \frac{5}{8}\mathbf{c}_2^i \\ \frac{1}{4}\mathbf{c}_1^i + \frac{3}{4}\mathbf{c}_2^i \\ \frac{1}{8}\mathbf{c}_1^i + \frac{7}{8}\mathbf{c}_2^i \\ \mathbf{c}_2^i \end{bmatrix} \quad (3.21)$$

and perform Gaussian elimination,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_1^{i+1} \\ \mathbf{c}_2^{i+1} \\ \mathbf{c}_3^{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{c}_1^i \\ \frac{1}{2}\mathbf{c}_1^i + \frac{1}{2}\mathbf{c}_2^i \\ 0 \\ 0 \\ \mathbf{c}_2^i \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.22)$$

which reveals that indeed this expression has a solution! The solution is $\mathbf{c}_1^{i+1} = \mathbf{c}_1^i$, $\mathbf{c}_2^{i+1} = \frac{1}{2}\mathbf{c}_1^i + \frac{1}{2}\mathbf{c}_2^i$ and $\mathbf{c}_3^{i+1} = \mathbf{c}_2^i$. This does not really come as a surprise because of the convex hull property. A two point surface spline curve is just a line segment.

We increase the number of intervals in the coarse mesh to two, and insert a point into one of these intervals. After right-hand side multiplication and Gaussian elimina-

tion, we get

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_1^{i+1} \\ c_2^{i+1} \\ c_3^{i+1} \\ c_4^{i+1} \end{bmatrix} = \begin{bmatrix} c_1^i \\ \frac{1}{2}c_1^i + \frac{1}{2}c_2^i \\ 0 \\ 0 \\ c_2^i \\ 0 \\ c_1^i - 2c_2^i + c_3^i \\ c_1^i - 2c_2^i + c_3^i \\ \frac{1}{2}c_1^i - \frac{1}{2}c_2^i + 2c_3^i \\ c_1^i - c_2^i \\ c_1^i - c_2^i \\ c_1^i - c_2^i \\ c_1^i - c_2^i \end{bmatrix} \quad (3.23)$$

The last four equations requires c_1^i and c_2^i to be equal, and thus this system does not have a solution for arbitrary control points at level i .

We conclude that, while keeping blend ratios equal to $1/2$, we cannot insert a new control point and get the exact same geometry.

3.3.4 Futile attempt no. 2: uniform refinement

Inserting a point in one of the intervals failed, so instead we try to insert a point into every interval. *Given a surface spline curve with n control points, can we find a curve with $2n - 1$ control points with the exact same geometry?* The initial scenario, refining a curve with two control points, is exactly the same as in the previous section — the curve has only one interval to be refined. And this problem did in fact have a solution.

Then, let us try to refine a curve with three control points, that is, a curve with two intervals. We insert a new control point in the middle of each interval. After right-hand side multiplication and Gaussian elimination we get

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_1^{i+1} \\ c_2^{i+1} \\ c_3^{i+1} \\ c_4^{i+1} \\ c_5^{i+1} \end{bmatrix} = \begin{bmatrix} c_1^i \\ \frac{1}{2}c_1^i + \frac{1}{2}c_2^i \\ 0 \\ 0 \\ c_2^i \\ 0 \\ c_1^i - 2c_2^i + c_3^i \\ c_1^i - 2c_2^i + c_3^i \\ \frac{1}{2}c_1^i - \frac{1}{2}c_2^i + c_3^i \\ c_1^i - 2c_2^i + c_3^i \\ c_1^i - 2c_2^i + c_3^i \\ c_1^i - 2c_2^i + c_3^i \\ -3c_1^i + 6c_2^i - 2c_3^i \\ c_1^i - 2c_2^i + c_3^i \\ c_1^i - 2c_2^i + c_3^i \\ c_1^i - 2c_2^i + c_3^i \\ c_1^i - 2c_2^i + c_3^i \end{bmatrix} \quad (3.24)$$

which requires $2c_2^i$ to equal $c_1^i - c_3^i$, and thus we do not have a solution for arbitrary control points at level i .

We conclude therefore that, when blend ratios are equal to $1/2$, we cannot insert a new control point in between each adjacent pair of control points and get the exact same geometry. However, allowing blend ratios to change *could* possibly give us a subdivision formula.

Our conclusion is therefore that the surface spline curve does not have a general subdivision formula, and this implies that that the simplified surface spline scheme does not have a subdivision formula either.

3.4 Summary

The surface spline curve is the result of applying the surface spline paradigm on a curve. We defined the steps of the curve and investigated the steps of the construction. In addition, we tried to find a refinement rule for surface spline curves, however this gave no results.

This chapter serves two purposes; primarily it is an intuitive introduction to surface spline schemes, since the case of the curve are simpler than the case of the surface, and requires substantially less book-keeping. Secondly, it is used to define the boundary curve of the simplified surface spline curve.

Chapter 4

Surface spline schemes

Surface spline schemes are methods for obtaining a patchwork of traditional triangular or rectangular B-spline patches, with smoothness constraints fulfilled over the joints, from a control mesh of arbitrary topology. The surface spline is the resulting surface defined by this patchwork. The surface spline scheme tries to:

1. Create a surface over an arbitrary parameter domain, where the surface satisfies smoothness constraints.
2. Represent these patches as a patchwork of conventional patches, so that the scheme can be easily integrated into existing frameworks and pipelines.
3. Achieve minimal polynomial degree for each patch.
4. Have a local construction, so that no large linear system has to be solved.

Blend ratios are used to control the tightness of the surface (see figure 4.1 on page 46). The blend ratios play the same role for surface splines as knot intervals do for B-splines.

If the control mesh of the surface is a planar polyhedron (see section 2.4.5), we can usually make a planar cut polyhedron from it, and from this we can always define a C^1 surface of Bézier patches (Peters 1995*b*). Thus, the surface spline must, unless the control polyhedron is a suitable polyhedron, perform a refinement (to isolate offending vertices) and perform a local planarisation. Then we can define the surface as a composite set of patches.

4.1 The C^1 -surface spline

The C^1 -surface spline scheme (Peters 1995*a*) is a generalisation of the biquadratic tensor product B-spline. The input mesh topology gives the layout of the “knot vectors” and the blend ratios describe the knot spacing. This defines a vector space, where the points associated with the vertices of the control mesh are coefficients relative to some basis, and thus specifies a particular spline inside this space. The resulting surface is described as a collection of Bézier patches that are either rectangular, triangular or a mix. If the mesh is composed of triangular or a mix of rectangular or triangular patches, the surface is always G^1 . The input of this scheme consists of three ingredients:

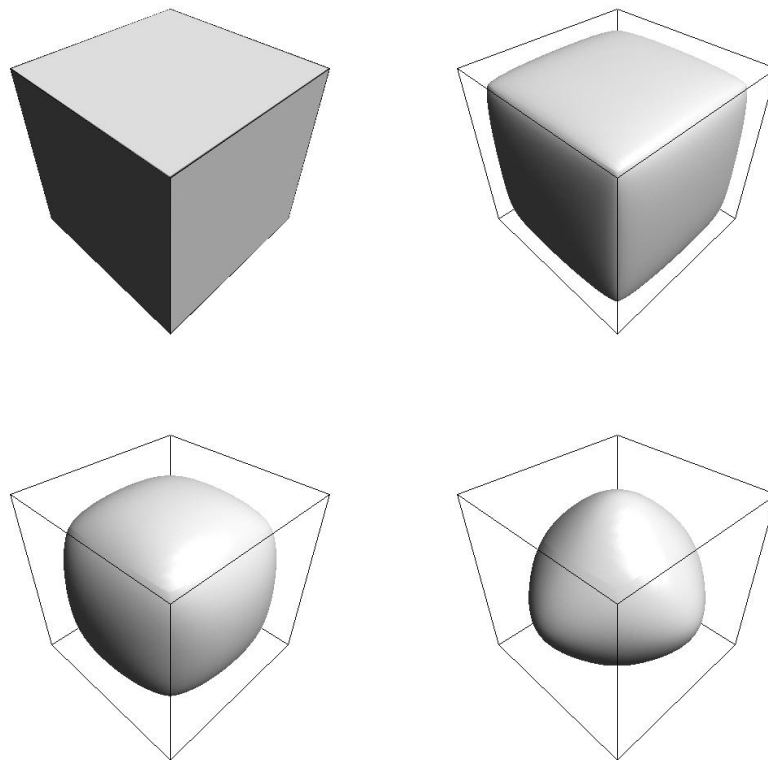


Figure 4.1: Different blend ratios. The simplified surface spline scheme was used to generate these images. In upper left the blend ratios are all equal to 0. Upper right, blend ratios are 0.25, lower left, 0.5, and lower right, 0.75.

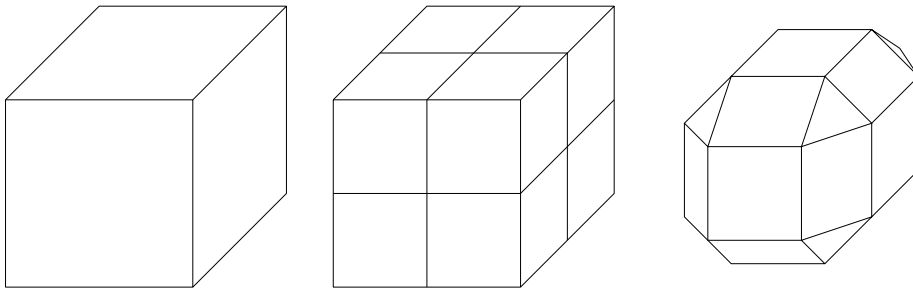


Figure 4.2: The tree meshes of the C^1 -surface spline scheme. Left: The control mesh, Middle: The quad mesh, and right: The inner mesh.

1. A control mesh. This mesh must satisfy the *local degree boundedness* property (see definition 2.4.3 on page 29).
2. One blend ratio associated with each dart of the control mesh. The blend ratios have a value in the range $[0, 1]$. The blend ratios describe a property analogous to knot spacing. Smaller blend ratios give a sharper surface, and if all blend ratios associated with a vertex is zero, we get the same effect as collapsing a knot interval; a degree of smoothness is lost (and in this case we get interpolation of that particular vertex).
3. One point in \mathbb{R}^3 associated with each vertex of the control mesh. The points must satisfy the *projective convexity* property. The projective convexity property states that for each face there exists a projection of the face such that the face is the convex hull of it's vertices. This assures that holes and bays of the mesh faces are not covered by the surface. *However, this means that some restrictions apply to the coefficients.* This makes it awkward to find a general basis. However, the simplified surface spline scheme does not this restriction.

4.1.1 The scheme

We give a quick overview of the C^1 -surface spline scheme. For more elaborate explanations and formulas, see (Peters 1995a).

Three meshes are used in the construction (see figure 4.2). The first mesh is the control mesh, the second is the quad mesh and the third is the inner mesh. The quad mesh is the result of a polyhedral split of the control mesh, and the inner mesh is the dual of the quad mesh.

The scheme is composed of six steps: Mesh refinement, edge cutting, quadratic meshing, quadratic patching, degree rising, and twist adjustment.

Mesh refinement

The initial step of this scheme refines the control mesh. There are at least two reasons for this step: to separate offending regions such that the planar cut polyhedron property is achieved, and to convert the mesh to an all-quad mesh. This refinement procedure is the polyhedral split, described in section 2.4.2. The quad mesh consists only of quadrilaterals (see figure 4.3 and figure 4.2 left and middle).

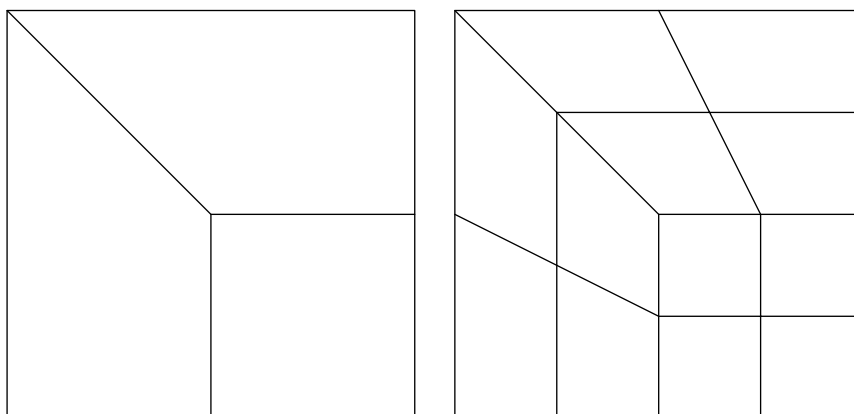


Figure 4.3: Mesh refinement of the control mesh (left) resulting in the quad mesh (right).

The refined mesh \mathfrak{M}^{i+1} is called the *quad mesh*. Each face in the quad mesh is a quadrilateral, and they are known as quads. Each quad is surrounded by four edges, and these edges are called quad edges. Two of the edges are the result from a split face, and two of the edges are the result from a split edge. There is a one-to-one correspondence between corners in \mathfrak{M}^i and faces in \mathfrak{M}^{i+1} .

An irregular vertex is a vertex where more or less than four faces abut. Each quad of the quad mesh abuts *at most* one irregular vertex. Each quad is defined from one vertex which corresponds to a vertex in the control mesh, one vertex which corresponds to a face midpoint, and two vertices corresponding to edge midpoints.

A vertex in the quad mesh is either a vertex from the control mesh, an edge midpoint or a face midpoint. The edge midpoints *always* have four abutting quads (degree four). If a face midpoint has a degree larger than four (the result of a face of more than four vertices), then all of the surrounding vertices are either edge midpoints, or vertices from the control mesh which has a degree of four or lower (due to the local degree boundedness).

Edge cutting

This step performs the planar cut. Each quad has two exclusive blend ratios associated with the two edges of the quad corresponding to edges in the control mesh (Remember the one-to-one correspondence between corners of the control mesh and the quads). For each quad in the quad mesh we create a temporary quad midpoint (see figure 4.4).

The quad midpoint is a bilinear interpolation of the four vertices of the quad, where the blend ratios are used as parameters. If the blend ratios are zero, the quad midpoint ends up at the position of the one vertex of the quad which corresponds to a vertex of the control mesh. If both blend ratios are one, then the quad midpoint ends up at the face midpoint of the mesh.

Then, for each vertex in the quad mesh with a degree larger than four, all the quad midpoints of the abutting quads are projected into a common plane. The projected quad midpoints and quad midpoints not touched by a projection are kept, and the rest of the construction uses these quad midpoints.

From the quad midpoints we define the *inner mesh* (see figure 4.2 right). The inner

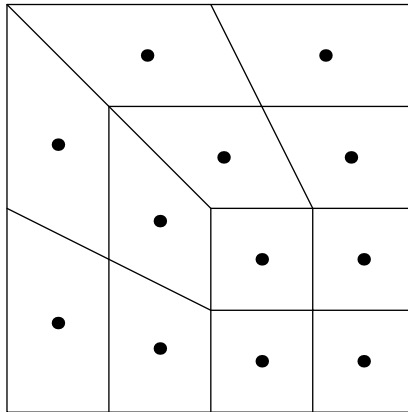


Figure 4.4: Edge cutting: The quad midpoints are bilinear interpolations of the vertices of the quad.

mesh is the dual of the quad mesh, and we use the quad midpoints as geometry for this inner mesh. The inner mesh is a planar cut polyhedra of the control mesh.

Quadratic meshing

The quadratic coefficients come in three flavours: Quad midpoints, quad mesh vertices and quad edge vertices (see figure 4.5).

First we calculate the quad mesh vertices. The position of each vertex in the quad mesh is set to the average of the midpoints of abutting quads.

Then we calculate the quad edge midpoint. A quad edge midpoint is the average of the midpoints of the two abutting quads. The quad edge midpoints of quad edges abutting a quad mesh vertex always lie in a common plane. If three quads abut, this is obviously true. If four quads abut, this is also true (proof: pose it as a matrix problem, and this matrix has rank 3). If more than four quads abut, the quad midpoints have been projected into a common plane and thus averages are in the same plane.

Each edge in the quad mesh has three coefficients associated (two quad vertex coefficients and one quad edge midpoint), and these coefficients can be interpreted as the coefficients for a quadratic Bézier curve segment. The curve segments connecting a quad mesh vertex have first order end derivatives in a common plane (the quad edge midpoints of edges abutting a vertex resides in the same plane and the curves have the same end point).

Quadratic patching

Each quad has nine coefficients associated: One quad midpoint, four quad edge midpoints, and four quad mesh vertices. One of these is from the split face, two is from split edges, and one is from a control mesh vertex.

We can patch each of these quads with either one quadratic Bézier patch (using the coefficients directly as Bézier coefficients) or alternatively as four triangular patches where the coefficients are weighted averages of the nine coefficients. The result is a surface that is C^1 in most areas.

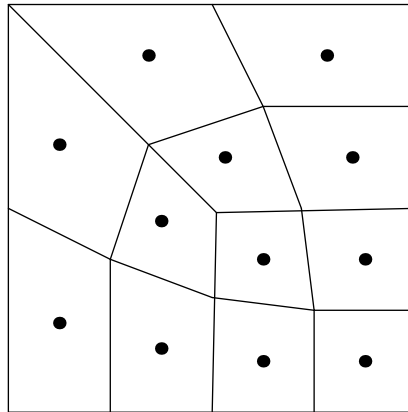


Figure 4.5: Quadratic meshing. The vertices of the quad mesh are relocated to the average of the quad midpoints of the abutting quads.

Degree raising

The term “irregular vertex” is dependent on context. In context of C^1 surface splines, the following classifies whether a vertex is regular or irregular.

Definition 4.1.1. *A regular vertex is a vertex where exactly four patches abut. A vertex which is not regular is irregular.*

The surface is not smooth in the neighbourhood of irregular vertices. The twist (mixed derivatives) does not necessarily match. We can fix this, but we need some additional degrees of freedom. To get these degrees of freedom we raise the degree of patches abutting irregular vertices from quadratic to bicubic. This action does not change the surface.

Twist adjustment

Then, we employ the new degrees of freedom to get the desired smoothness around irregular vertices as well. We only tweak the coefficients residing in the interior of the patch, and thus the boundary curve of each patch is kept. The procedure is dependent on our choice of four sided or three sided patches in the step “quadratic patching”.

4.2 Two other surface spline schemes

We give an overview of two other surface spline schemes.

4.2.1 Surfaces ... bicubics

The article (Peters 1994) is the predecessor of (Peters 1995a). To quote Peters: “For general modelling I recommend C^1 -surface splines. Surfaces ... bicubics is older, uses more patches, uses more complicated formulas, and does not prove the convex hull property. It is however the first of this type of construction.” The algorithm is rather similar to that of C^1 surface splines, however a bit more laborious:

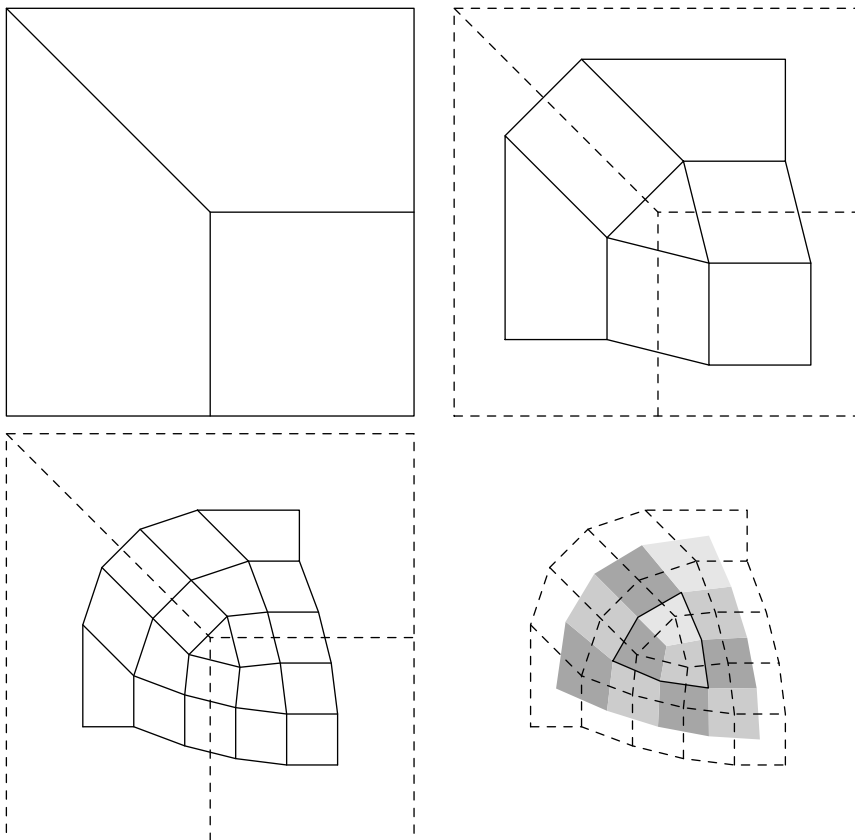


Figure 4.6: The scheme of surfaces ... bicubics. *Upper left:* The initial control mesh. *Upper right:* The refined mesh after one refinement. *Lower left:* The refined mesh refined yet another time. *Lower right:* The patch structure. Each vertex of the refined mesh is in the middle of one spline patch. The solid line renders the “hole” which is not fitted with quadrilateral patches, the three patches inside this hole is of cubic degree.

1. The input mesh is refined twice with the Doo-Sabin refining and averaging procedure in order to separate irregular vertices (which become non-quadrilateral mesh cells after the first refinement). *This removes the need for the local degree boundedness property.* However, it results in more patches.
2. Every vertex in the double refined mesh where all neighbours are regular (and thus is a 3×3 regular sub-mesh) is patched with a biquadratic B-spline.

The knots of the knot vectors are evenly spaced, and the knot vectors have no multiple knots. As a result of this, the patch only covers the area around the vertex and halfway to the neighbours (see figure 4.6). This knot vector makes it possible for the B-spline to use the neighbouring vertices as coefficients, and the patches will abut nicely and smooth.

Alternatively, the patches may be rewritten in Bézier form, but this increases the number of coefficients.

3. An irregular vertex in the control mesh will represent a triangle after the Doo-Sabin splits (see figure 4.6). The patches corresponding to the vertices of this triangle is not yet patched, and thus around each irregular vertex in the control mesh, we have a hole in the mesh in the shape of a convex n -gon built from n rectangular patches, where n is the degree of the irregular node.

Due to two steps of the Doo-Sabin refinement, none of these holes touches each other, and thus can be handled separately.

Each vertex corresponding to a patch in the hole (there is n of them) is patched with a bicubic patch. The patch boundaries of these meshes that abuts the regular patches are degree raised quadratics, and thus we get smooth transitions between the regular and irregular patches.

The increased degree of these patches gives the required degrees of freedom required to make a smooth transition between the irregular patches.

This scheme performs one more refinement step than C^1 surface splines. The gain from this is that the input mesh can have a more arbitrary topology, but at the expense of increased number of patches.

4.2.2 Localised-hierarchy surface splines

LeSS (Gonzalez & Peters 1999), an acronym for Localised-Hierarchy Surface Splines, is an extension of (Peters 1995a). In essence, the surface is the same, but methods are presented to remove a small part of the mesh and replace it with another mesh fragment. The influence of this operation is local. An interesting effect of this scheme is that we can not only make refinement, but actual topological changes to the mesh with local influence. We can make bridges between two parts of the meshes, or introduce topological holes.

However, it is important to note that *this new surface is not exactly equal to the old surface, only very similar.* Thus, we cannot get nested spaces of surface splines with this construction.

4.3 Summary

This chapter begins with the motivation of surface spline scheme. The main idea is to create a scheme behaving like the regular tensor product B-spline, but applicable for arbitrary topologies. In addition, by defining the surface as a set of standard Bézier patches, the scheme can easily be incorporated into existing geometry frameworks.

Three surface spline schemes are described, with extra emphasis on one particular scheme, the C^1 -surface spline. This scheme is the “full” surface spline scheme of which the simplified surface spline is derived from.

A regular vertex is a vertex surrounded by four faces. An irregular vertex is vertex where this is not the case. The idea is to refine the control mesh to separate irregular vertices. Then we associate a midpoint to each quad in the refined mesh. We can create a dual of the refined mesh by using the quad midpoints as vertices. This dual is a planar cut polyhedron after the local projection.

Around irregular vertices the following steps are necessary to produce a C^1 surface: The local planar projection assures coplanarity of Bézier control points of relevant coefficients from abutting patches. The raising of degree from quadratic to cubic on the patches increases the number of degrees of freedom which are used by the twist correction, which perturbs the inner control points to create matching cross-derivatives.

Chapter 5

The simplified surface spline

The simplified surface spline is a simplification of C^1 -surface splines (see section 4.1). The construction is almost the same, however with the following differences:

- No planar projection is performed.
- Only rectangular biquadratic Bézier patches are used, thus, only one kind of patches.
- No twist-correction is performed.

The simplified surface spline is not necessarily smooth in small neighbourhoods of irregular vertices (see definition 4.1.1). We trade full C^1 -continuity with “almost C^1 -continuity” for the benefit of a simpler scheme.

5.1 Overview

The scheme consists of five layers, and each layer has an array of points. A point in one layer is a convex combination of points in the preceding layer. The layers are:

1. Layer p is filled by the control mesh vertices, which are indexed by vertices. The polyhedral split combines the vertices of the control mesh to form the vertices of the quad mesh ($p \rightarrow ps$). The points in ps plays the same role as the refined points of the surface spline curve (section 3.1.1).
2. Layer ps is filled by the vertices of the quad mesh, which are indexed by vertices, edges and faces. The quad midpoint generation combines the vertices of the quad mesh to calculate the quad midpoints ($ps \rightarrow qm$). The quad midpoints are equivalent with the sub-interval midpoints of the surface spline curve (section 3.1.2).
3. Layer qm is filled by the quad midpoints, which are indexed by corners. Bézier coefficient calculation calculates the Bézier coefficients from the quad midpoints ($qm \rightarrow bc$). The Bézier coefficients corresponds to the Bézier coefficients of the surface spline curve (section 3.1.3). This step is changes geometry such that abutting patches often¹ have coplanar control points at joints.

¹For this always to happen, a planar projection must take place.

Topological term	Symbol	Indexing	Used in layers
Original vertices	•	(v_i)	p, pv, bv
Edge midpoint	★	(e_j)	pe, be
Face midpoint	◆	(f_k)	pf, bf
Quad midpoint	▼	(v_i, f_k)	qm
Edge quartile	▲	(v_i, e_j)	beq
Inner edge quartile	▲	(v_i, f_k)	biq

Table 5.1: Terms in the construction with the symbols used in the figure. Note that the edge quartiles and the inner edge quartiles have the same symbol. Indexing denotes how the points are indexed.

4. Layer bc is filled by the Bézier coefficients. The indexing is too complex to be stated in one sentence. The Bézier evaluation does a biquadratic Bézier evaluation of the relevant Bézier coefficients ($bc \rightarrow s$). The biquadratic Bézier evaluation is equivalent to the quadratic Bézier segment evaluation of the surface spline curve (section 3.1.4).
5. Layer s is the evaluated surface.

5.1.1 Topological terms of the simplified surface spline

All layers are indexed by topological terms, and these terms have symbols, used in the subsequent figures, associated with them. As an example, the vertex term refers to a coefficient attached to a vertex, and the inner-edge quartile term refers to a coefficient attached to the midpoint of an edge in the quad mesh. Section 2.2.2 defines the different positions in the mesh.

Since the control mesh can be more or less arbitrary, we cannot utilise regular sequential indices, which are used in the construction of B-splines. This can be confusing (if conventional splines over regular topologies have a mess of indices, surface splines over arbitrary topologies are even worse). For simplified surface splines, we use six terms to define the coefficients:

1. **Vertices** are where the vertices of the control mesh are. These are indexed by the vertices.
2. **Edge midpoints** are the middle of each edge of the control mesh. These are indexed by the edges.
3. **Face midpoints** are the middle of each face of the control mesh. These are indexed by the faces.
4. **Quad midpoints** are located in the middle of each quad of the refined control mesh (the quad mesh). There is a one-to-one correspondence between each quad in the quad mesh and each corner in the control mesh, and these are indexed by the corners of the control mesh.
5. **Edge quartiles** are in the middle of the edges in the quad mesh which is the result of a split edge. Every edge in the control mesh has two edge quartiles, between the edge midpoint and their respective end vertex of the edge (thus their

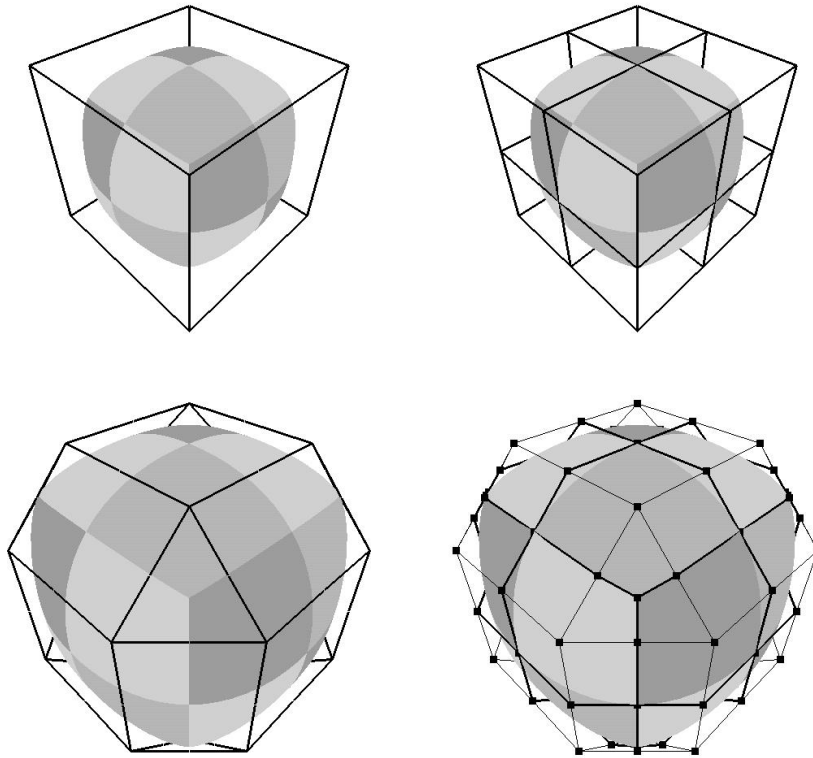


Figure 5.1: Surface spline concepts. The individual patches are coloured in different shades of grey. Upper left, the control mesh. Upper right, the quad mesh. Lower left, the inner mesh, and lower right, the Bézier coefficients.

name quartiles). The edge quartiles are indexed by an edge and a vertex (one of the two defining the edge).

6. **Inner edge quartiles** are in the middle of the edges in the quad mesh which is a result of face splitting (Each edge in the quad mesh are either the result of a split edge or a split face). Each edge in the control mesh about a face in the same mesh gives raise to an edge in the quad mesh which is the result of a face split. Thus, these are indexed by an edge and a face from the control mesh.

The reader is recommended to look at table 5.1 which is the key between terms and the symbols used in figures, as well as the figures 5.2, 5.3, 5.4 and 5.5 where these symbols are used.

5.1.2 The three meshes

There exist *three* meshes in the scheme (refer to figure 5.1. The first mesh is the *control mesh*, which is the input to the scheme. The second mesh is a refinement of the control mesh, which is known as the *quad mesh*. The vertices of the control mesh also exist in the refined mesh. Hence, the set of vertices in the control mesh is a subset of the

vertices in the refined mesh. The third mesh, not used explicitly, is the *inner mesh*. It is the aim of the C^1 -surface spline scheme to make this inner mesh into a planar cut polyhedron.

5.1.3 The intrinsic parameterisation

There is a one-to-one mapping between corners of the control mesh and the Bézier patches of the surface. Thus, the corners are useful for indexing the patches. Each patch has the unit square as its local domain.

Definition 5.1.1. *A parameter point p on the simplified surface spline surface is a triple*

$$p = (c, s, t), \quad (5.1)$$

where $(s, t) \in [0, 1] \times [0, 1]$, and c refers to a corner in the control mesh.

The patches do overlap at the patch boundaries. However, the abutting patches evaluate to the same value, and thus this is just a matter of choosing which patch to evaluate.

The parameterisation is much influenced by the *blend ratios*. Blend ratios are in the interval $[0, 1]$. If the blend ratios are outside this interval, the construction is no longer entirely constructed from convex combinations. The blend ratios play the same role as the knot spacing does for B-splines. From figure 4.1 we see that small blend ratios produce the same effect as small knot spacing, sharper geometry, and large blend ratios give smoother geometry. When blend ratios are reduced to zero, we lose a degree of continuity.

5.2 Layer I: The control mesh vertices

The input to the scheme is a control mesh with a blend ratio associated with each dart of this mesh and a control point associated with each vertex.

The initial layer is composed of the control points. Each vertex has a control point associated, and thus, we index the control points with the vertices. The control points must reside in a linear space such that convex combinations can take place. Otherwise, there are no restrictions, and thus, the points can be of arbitrary dimension. However, points from \mathbb{R}^3 are probably the most useful choice.

5.3 Layer II: The quad mesh points

The control mesh (figure 5.2) is split with the polyhedral split scheme (see section 2.4.2) to make the quad mesh (figure 5.3). The original vertices are kept, the edge midpoint is the average of the two vertices the edge connects, and the face midpoint is the average of the vertices of the face. We call the vertices obtained from this step the *ps*-vertices, referring to the splitting scheme, the polyhedral split.

We use the notation $|_{\text{restriction}}$ to restrict the expression to a specific case. Thus, $\mathbf{ps}|_V$ is the polyhedral split in the case of a vertex.

Definition 5.3.1. *We define the vertices of the layer \mathbf{ps} as a combination of vertices in the control mesh layer \mathbf{p} . There are three flavours of these; vertices, edge midpoints*

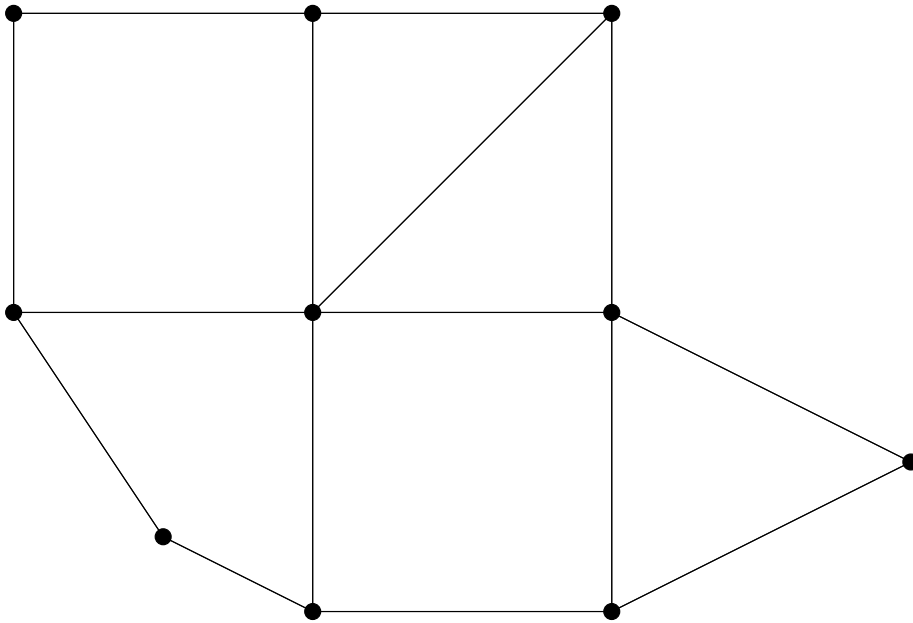


Figure 5.2: The initial control mesh. In this case: Three quadrilaterals and three triangles, fifteen edges and ten vertices.

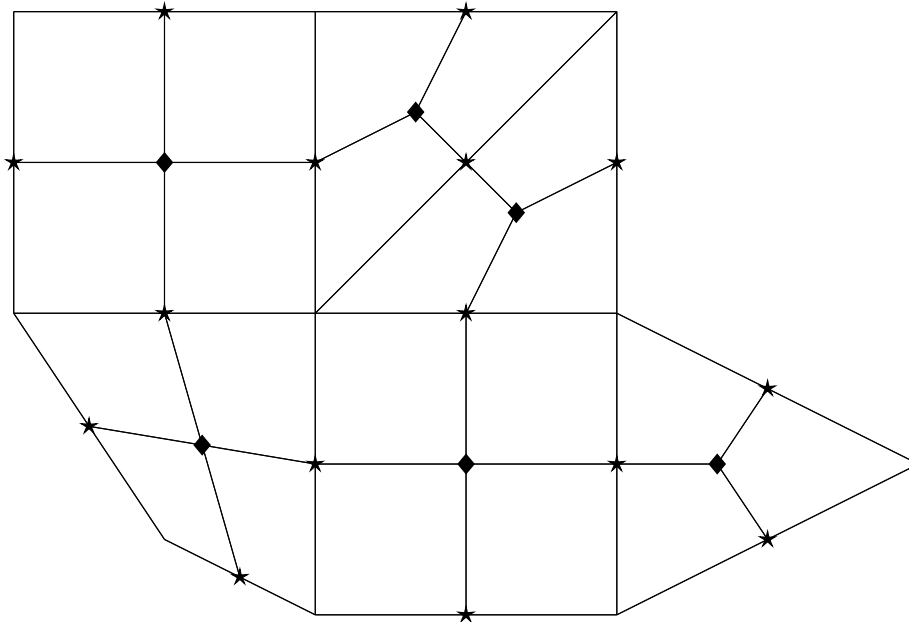


Figure 5.3: The quad mesh. This mesh defines the patch structure; each quadrilateral corresponds to one Bézier patch.

and face midpoints:

$$\begin{aligned}
 \mathbf{ps}|_V(v_i) &= \mathbf{p}(v_i) \\
 \mathbf{ps}|_{EM}(e_j) &= \frac{1}{2} \sum_{v_l \in e_j} \mathbf{p}(v_l) \\
 \mathbf{ps}|_{FM}(f_k) &= \frac{1}{\#\hat{f}_k} \sum_{v_l \in \hat{f}_k} \mathbf{p}(v_l).
 \end{aligned} \tag{5.2}$$

The three functions give the position of the vertex corresponding to the old vertex, position of the vertex corresponding to the edge midpoint and the position of the vertex corresponding to the face midpoint respectively.

The notation is defined in section 2.2. Quad meshes (the result of a polyhedral split) is treated in section 2.4. This operation is the equivalent of the refinement-step of the surface spline curve construction (section 3.1.1).

One important observation can be made; all of the new vertices are convex combinations of vertices in the control mesh.

Lemma 5.3.2. *All vertices of the quad mesh (layer II) are convex combinations of the vertices of the control mesh (layer I).*

Proof. The averages in definition 5.3.1 are obviously convex combinations. □

Corollary 5.3.3. *There are $\#\mathbb{V} + \#\mathbb{E} + \#\mathbb{F}$ points in the quad mesh layer.*

Proof. The points of the vertices are kept, and each edge get a midpoint as well as each face. □

5.4 Layer III: Quad midpoints (inner mesh vertices)

From the quad mesh vertices, we compute the quad midpoints. The quad midpoints define the inner mesh vertices. The inner mesh is not explicitly used, but the inner mesh is the planar cut polyhedron² from which we define the faces.

A quad midpoint is a bilinear interpolation of the four vertices of the quad. The parameters of this interpolation are the blend ratios, and the coefficients are the four vertices of the quad. A overview of the bilinear interpolation is given in section 1.4.2.

Each quad corresponds to a corner in the control mesh. Each corner in the control mesh “has” two non-exclusive edges associated, the edge in and the edge out. The combination of face and vertex (from the corner) in addition to an edge gives an exclusive key (a dart in this case), and for each such key we have a blend ratio associated. Thus, each corner has two exclusive blend ratios associated. This calculation is visualised in figures 5.3 and 1.1.

At the boundary, the edge quartiles of the curve play an equal role as the quad midpoints of the surface. To achieve this, we assign a value to the edge quartiles of every boundary edge in this step. This is to achieve a border in the form of a surface spline curve. No edge quartiles are defined for interior edges. Definition 5.4.1 uses mesh queries, defined in section 2.3.3.

²It is not necessarily a planar cut polyhedron since the planar projection is skipped

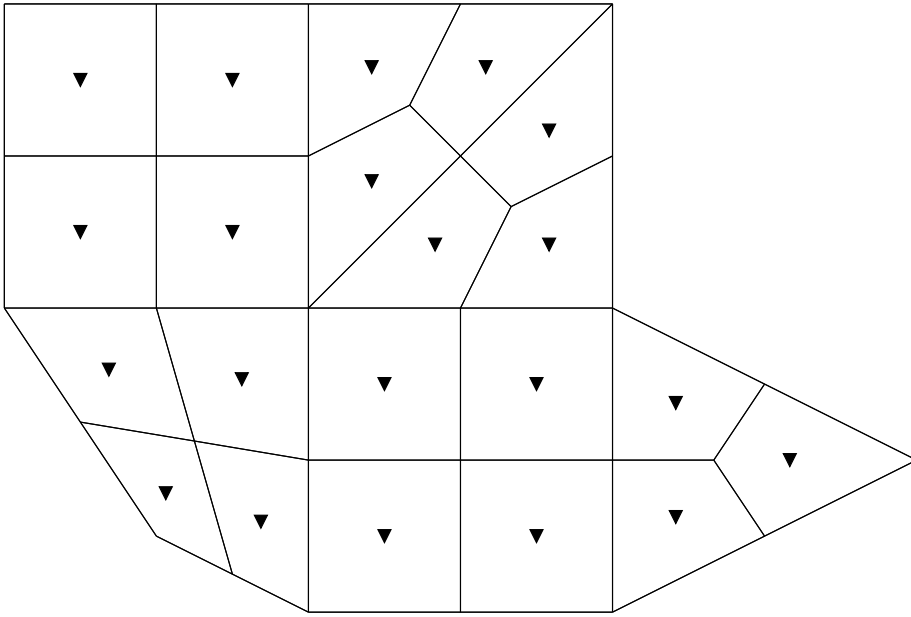


Figure 5.4: The quad midpoints. Connecting these points together forms the inner mesh.

Definition 5.4.1. Let the set \mathbb{BE} be the set of edges in the mesh \mathfrak{M} which are boundary edges,

$$\mathbb{BE} = \{e_j \in \mathfrak{M}[e; *, *, *] \mid \#\mathfrak{M}[f; *, e_j, *] = 1\} \quad (5.3)$$

which is the set of edges where the number of abutting faces are one. Similarly, we may define the set of boundary vertices

$$\mathbb{BV} = \{v_i \in \mathfrak{M}[v; *, *, *] \mid \#\mathfrak{M}[f; v_i, *, *] \neq \#\mathfrak{M}[e; v_i, *, *]\}, \quad (5.4)$$

which is the set of vertices where the number of connecting edges are not the same as the number of abutting faces.

One may read “if v_i is a boundary edge” as “if $v_i \in \mathbb{BV}$ ” to make it more formal, and less readable. The same goes for edges. The quad midpoints corresponds to the sub-interval midpoints of the surface spline curve (section 3.1.1).

Definition 5.4.2. We define the vertices of the layer qm , the quad midpoints, as combinations of the vertices of layer ps . The vertices are indexed by the corners of the control mesh. Given a corner (v_i, f_k) , we have two edges associated with it, e_{in} and

e_{out} as well as two blend ratios, b_{in} and b_{out} , and \mathbf{qm} is defined

$$\begin{aligned} \mathbf{qm}|_{QM}(v_i, f_k) &= (1 - b_{\text{in}})(1 - b_{\text{out}})\mathbf{ps}|_V(v_i) + \\ &\quad b_{\text{in}}(1 - b_{\text{out}})\mathbf{ps}|_{EM}(e_{\text{in}}) + \\ &\quad (1 - b_{\text{in}})b_{\text{out}}\mathbf{ps}|_{EM}(e_{\text{out}}) + \\ &\quad b_{\text{in}}b_{\text{out}}\mathbf{ps}|_{FM}(f_k), \\ \mathbf{qm}|_{EQ}(v_i, e_j) &= \begin{cases} (1 - b)\mathbf{ps}|_V(v_i) + b\mathbf{ps}|_{EM}(e_j) & \text{if } e_j \text{ is a boundary edge} \\ \text{undefined} & \text{otherwise,} \end{cases} \end{aligned} \tag{5.5}$$

where b is a blend ratio. The query $d = \mathfrak{M}[v, e, f; v_i, e_j, *]$ will give one dart, d , if e_j is a boundary edge, and b is the blend ratio associated with this dart.

The expressions in equation 5.5 are a bilinear interpolation and a linear interpolation respectively. If both blend-ratios are zero, the quad midpoint gets the position of the original vertex of the patch. If both blend-ratios are one, the patch midpoint gets the position of the face midpoint. Since all the Bézier coefficients depend on these patch-midpoints, we may utilise the blend-ratios to control the curvature of the resulting surface. While we're at it, patch midpoints are also convex combinations.

Lemma 5.4.3. *All vertices in $\mathbf{qm}(\dots)$ are convex combinations of the vertices of the control mesh.*

Proof. The bilinear interpolation is a convex function. See section 1.4.2 for details. Thus, in the quad midpoint case, the vertices are convex combinations of $\mathbf{ps}(\dots)$, which in turn is a convex combination of the vertices of the control mesh. From property 1.4.5 we know that this results in a convex combination. In the edge quartile case, the vertex is a linear interpolation of two points in $\mathbf{ps}(\dots)$ and the same argument applies here as well. \square

Corollary 5.4.4. *There are $\#\mathbb{C} + 2\#\mathbb{BE}$ points in the quad midpoint layer.*

Proof. There is a one-to-one correspondence between the quads in the quad mesh and the corners of the control mesh. Each quad has one quad midpoint, and there are $\#\mathbb{C}$ quads. In addition we need the edge quartiles at the boundary. Each edge has two edge quartiles, and thus this number is $2\#\mathbb{BE}$. \square

5.5 Layer IV: Bézier coefficients

The next layer is the layer of Bézier coefficients, which are combinations of the quad midpoints, and the Bézier coefficients are used by the biquadratic Bézier evaluation.

It is in this step the smoothing takes place. All Bézier coefficients are averages of surrounding quad midpoints (which reside in the interior of patches). In this way, the control points are coplanar for abutting patches along boundaries. The simplified surface spline construction is equal to the full C^1 surface spline scheme except at the neighbourhood of irregular vertices, and has thus the same smoothness properties as the full scheme at non-irregular neighbourhoods.

The Bézier coefficients of the surface scheme are analogous to the Bézier coefficient step of the surface spline curve construction. Bézier coefficients come in six flavours:

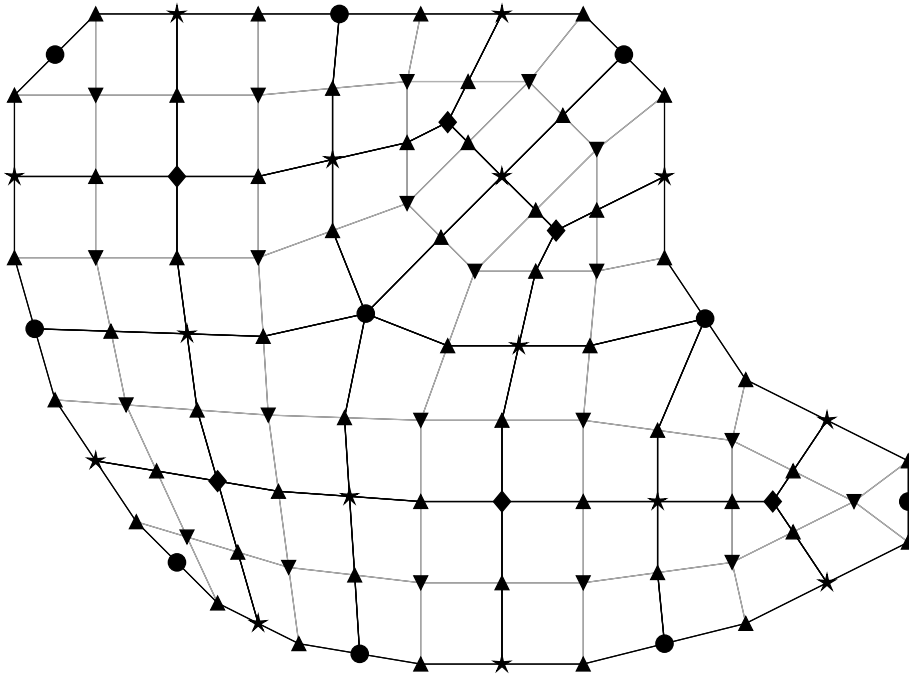


Figure 5.5: The Bézier coefficients.

1. **Vertex** coefficients are the average of the quad midpoints abutting that particular vertex in the control mesh. The number of abutting patches equals the number of faces abutting the vertex.
2. **Edge midpoint** coefficients are the average of the four abutting quad midpoints. Edge midpoint can also be the average of the two edge quartile coefficients of that edge, which gives the exact same result.
3. **Face midpoint** coefficients are the average of the quad midpoints of that particular face. If the face is an n -gon, the n quad midpoints of this face are used.
4. **Quad midpoints** coefficients are just a copy of the quad midpoint of the layer below.
5. **Edge quartile** coefficients are the average of the two quad midpoints abutting at that end of the edge.
6. **Inner edge quartile** coefficients are the average of two succeeding patch midpoints in a face in between the inner edge lies.

Figure 5.5 shows the layout of these coefficients and definition 5.5.1 defines the formulas.

The coefficients have two major cases at this step: boundary and non-boundary. In the case of a non-boundary coefficient, the coefficient is the average of the quad midpoints. However, in the boundary case, the coefficient is the average of some edge quartiles from the same layer. Consult figure 5.6 to get an idea of which coefficients are boundary and which are not.

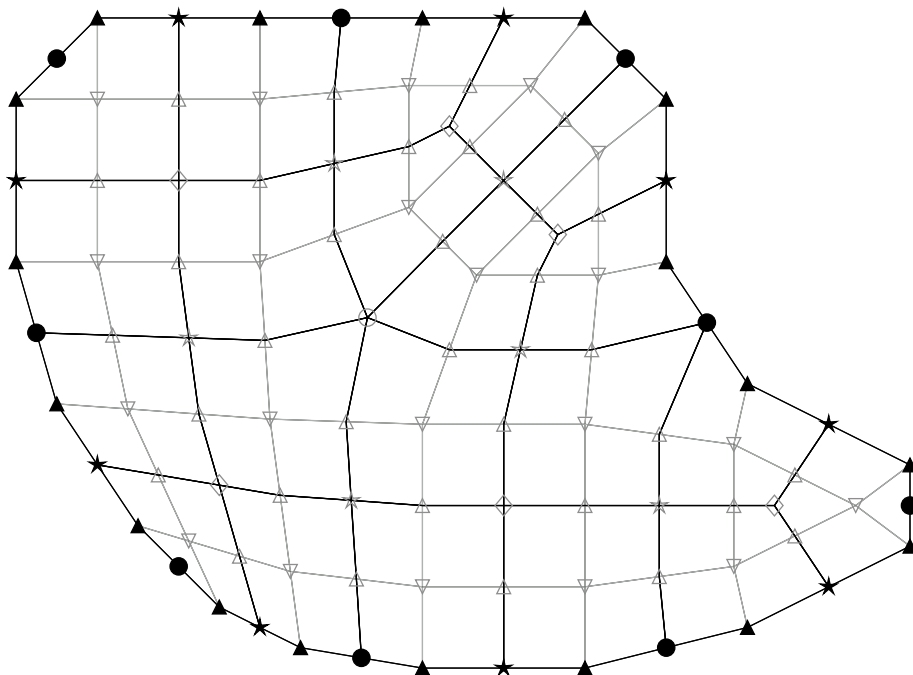


Figure 5.6: The boundary coefficients are in black while the non-boundary coefficients are in grey. Notice that no inner edge quartiles or face midpoints lie on the boundary.

Definition 5.5.1. All the Bézier coefficients are averages of a subset of the quad midpoints of the layer below. There are five types of Bézier coefficients: vertex (V), edge midpoint (EM), face midpoint (FM), edge quartile (EQ) and inner edge quartile (IQ). The general formula is

$$\mathbf{bc}(\dots) = \begin{cases} \frac{1}{\#\mathbb{C}} \sum_{(v,f) \in \mathbb{C}} \mathbf{qm}|_{QM}(v, f) & \text{if } \dots \text{ is non-boundary} \\ \frac{1}{\#\mathbb{EQ}} \sum_{(v,e) \in \mathbb{EQ}} \mathbf{qm}|_{EQ}(v, e) & \text{if } \dots \text{ is boundary,} \end{cases} \quad (5.6)$$

where \dots is the appropriate parameters. Obviously, only vertices, edge midpoints and edge quartiles can be boundary. In the non-boundary case, the Bézier coefficients are blends of the quad midpoints of the layer below. In the boundary case, the coefficients are blends of the edge quartiles defined in the layer below. In the non-boundary case, the set \mathbb{C} is a subset of the quad midpoints specific to the case and parameters,

$$\begin{aligned} \mathbb{C}|_V(v_i) &= \mathfrak{M}[v, f; v_i, *, *], \\ \mathbb{C}|_{EM}(e_j) &= \mathfrak{M}[v, f; *, e_j, *], \\ \mathbb{C}|_{FM}(f_k) &= \mathfrak{M}[v, f; *, *, f_k], \\ \mathbb{C}|_{QM}(v_i, f_k) &= \{(v_i, f_k)\}, \\ \mathbb{C}|_{EQ}(v_i, e_j) &= \mathfrak{M}[v, f; v_i, *, *] \text{ and} \\ \mathbb{C}|_{IQ}(e_j, f_k) &= \mathfrak{M}[v, f; *, e_j, f_k], \end{aligned} \quad (5.7)$$

In the boundary case, the set \mathbb{EQ} is a subset of the edge quartiles defined in the layer

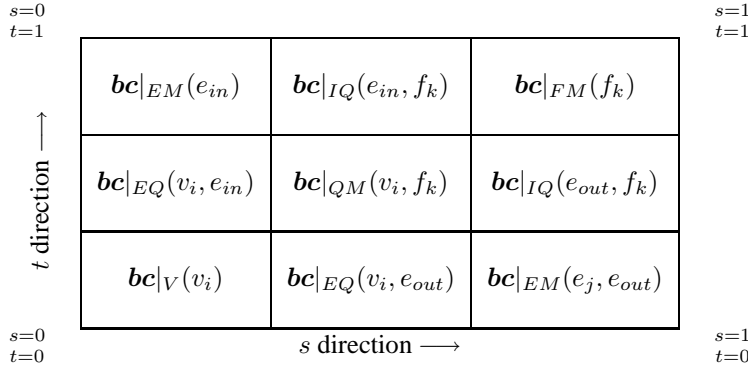


Figure 5.7: The layout of the coefficients used in the Bézier evaluation. From a corner (v_i, f_k) , we find the two edges, the incoming and outgoing. From this we know which vertices from the bc layer to use.

below,

$$\begin{aligned}
 \mathbb{EQ}|_V(v_i) &= \bigcup_{\forall e \in \mathbb{E}: v_i \in e} (v_i, e), \\
 \mathbb{EQ}|_{EM}(e_j) &= \mathfrak{M}[v, e; *, e_j, *] \\
 \mathbb{EQ}|_{EQ}(v_i, e_j) &= \{(v_i, e_j)\}
 \end{aligned} \tag{5.8}$$

Lemma 5.5.2. *The Bézier coefficients are convex combination of the control mesh vertices.*

Proof. All the functions are simple averages, which are convex combinations, of either (a) the quad midpoints of the layer below, or (b) the edge quartiles of the layer below. We utilise lemmas 5.3.2, 1.4.5, and 5.5.2, and this concludes the proof. \square

Corollary 5.5.3. *There are $\#\mathbb{V} + 3\#\mathbb{E} + \#\mathbb{F} + 2\#\mathbb{C}$ points in the Bézier coefficient layer.*

Proof. Each vertex has a point, each edge has three points (one midpoint and two quartiles), each face has the same number of inner-edge quartiles as corners and one midpoint. In addition has each patch a patch midpoint. \square

Corollary 5.5.4. *If the control mesh is closed, all Bézier coefficients can be calculated from the quad midpoints.*

Proof. If the control mesh is closed, it has no boundary edges, and edge quartiles are not used in this step. \square

5.6 Layer V: Bézier evaluation

Each patch is defined from nine coefficients. All of these coefficients, except the quad midpoints, are shared between neighbouring patches, which assures C^0 continuity. To get a consistent orientation of the surface, we let s and t run along the outgoing and the incoming edge respectively. If the control mesh is consistently oriented, the surface will be consistently oriented as well.

Definition 5.6.1. A patch is associated with each corner of the control mesh. The corner c is defined by a vertex-face pair. In addition, a corner has two edges associated with it, the incoming edge e_{in} and e_{out} .

$$\begin{aligned}
\mathbf{s}(c, s, t) = & (1-s)^2(1-t)^2 \mathbf{bc}|_V(v) + \\
& 2(1-s)^2(1-t)t \mathbf{bc}|_{EQ}(v, e_{out}) + \\
& (1-s)^2t^2 \mathbf{bc}|_{EM}(e_{out}) + \\
& 2(1-s)s(1-t)^2 \mathbf{bc}|_{EQ}(v, e_{in}) + \\
& 4(1-s)s(1-t)t \mathbf{bc}|_{QM}(v, f) + \\
& 2(1-s)st^2 \mathbf{bc}|_{IQ}(e_{out}, f) + \\
& s^2(1-t)^2 \mathbf{bc}|_{EM}(e_{in}) + \\
& 2s^2(1-t)t \mathbf{bc}|_{IQ}(e_{in}, f) + \\
& s^2t^2 \mathbf{bc}|_{FM}(f),
\end{aligned} \tag{5.9}$$

where (c, s, t) is a parameter point (definition 5.1.3), v, f, e_{in} and e_{out} are the vertex, face, incoming edge and outgoing edge of corner $c = (v_i, f_k)$ respectively.

The nine coefficients are blended with a biquadratic blend (see example 1.4.2). The biquadratic blend blends coefficients laid out in a three-by-three grid (see figure 5.7).

Property 5.6.2. The expressions given in definition 5.6.1 are convex combinations of the vertices of the control mesh.

Proof. The biquadratic Bézier blend is a convex combination of its coefficients (see section 1.4.2). In the usual spirit, we utilise properties 5.3.2, 1.4.5, and 5.5.2, and this concludes the proof. \square

With this, we have proved that any point on the simplified surface spline surface is computed as convex combinations of the control mesh vertices. This gives a useful property, the convex hull property.

Property 5.6.3. The complete simplified surface spline surface resides inside the convex hull of the control mesh vertices.

Proof. We know from property 5.6.2 that any point on the surface can be expressed as a convex combination of the control mesh vertices, and thus by property 1.4.4 we get property 5.6.3. \square

Chapter 6

The simplified surface spline basis

The steps of chapter 5 are intuitive and rather straight-forward. However, this way of representing the surface is not the best when dealing with qualitative aspects of approximation. We are interested in *how much* a specific control vertex is contributing to the surface at a given parameter point. Since all operations are convex combinations of the layer below we can view the scheme as a series of matrices applied to the control points. Each step defines a matrix. In addition, we introduce an additional matrix, the select matrix. The select matrix picks out the correct Bézier coefficients to be used in the Bézier evaluation. We define the following matrices:

1. The matrix \mathbf{S}_{ps} performs the polyhedral split. This matrix has $\#\mathbb{V}$ columns, and $\#\mathbb{V} + \#\mathbb{E} + \#\mathbb{F}$ rows.
2. The matrix \mathbf{S}_{qm} calculates the quad midpoint matrices and copies the boundary edge quartiles. This matrix has $\#\mathbb{V} + \#\mathbb{E} + \#\mathbb{F}$ columns and $\#\mathbb{C} + 2\#\mathbb{BE}$ rows.
3. The matrix \mathbf{S}_{bc} calculates the Bézier coefficients. This matrix has $\#\mathbb{C} + 2\#\mathbb{BE}$ columns and $\#\mathbb{V} + 3\#\mathbb{E} + \#\mathbb{F} + 2\#\mathbb{C}$ rows.
4. The matrix $\mathbf{S}_{sel}(c)$ is dependent on which patch to be evaluated. It extracts the correct set of Bézier coefficients to be used in the Bézier evaluation. This matrix has $\#\mathbb{V} + 3\#\mathbb{E} + \#\mathbb{F} + 2\#\mathbb{C}$ columns and 9 rows. This matrix contains nine 1's and the rest of the entries are zero. Which entries are non-zero is dependent on c .
5. The matrix $\mathbf{S}_{bz}(s, t)$ performs the Bézier evaluation. The entries in this matrix is the nine basis functions for the biquadratic Bézier patch. The matrix has 9 columns and 1 row.

Combining this, we get the matrix product

$$\begin{aligned}
 \mathbf{s}(c, s, t) &= \underbrace{\mathbf{S}_{bz}(s, t) \cdot \mathbf{S}_{sel}(c) \cdot \mathbf{S}_{bc} \cdot \mathbf{S}_{qm} \cdot \mathbf{S}_{ps}}_{\text{basis}} \cdot \mathbf{p} \\
 &= \sum_{b=1}^{N_v} B(v_b; c, s, t) \mathbf{p}(v_b),
 \end{aligned} \tag{6.1}$$

and the matrices combined give the basis functions. A basis function is a function that gives the amount of contribution a vertex has at a particular parameter value. It is in fact rather easy to deduce the entries for these matrices. However, these matrices become rather large, even for small models.

We will not use this matrix formulation directly when deducing the basis functions. Instead, we will write the matrix products explicitly as sums, and formulate the following hypothesis:

Hypothesis 6.0.1. *The simplified surface spline is a linear scheme (all steps are linear combinations of values of the previous step), thus we can write the surface as a sum,*

$$\mathbf{s}(c, s, t) = \sum_{b=1}^{N_v} B(v_b; c, s, t) \mathbf{p}(v_b), \quad (6.2)$$

where $B(v_b; c, s, t)$ is the basis function associated with v_i evaluated at the parameter point (c, u, v) .

Each vertex of the control mesh has its own basis function. Instead of writing “the vertex of which the basis function corresponds to”, we will consistently use v_b as this vertex. When we have several basis vertices (as in the sum of hypothesis 6.0.1), we use b as the summation variable.

6.1 Overview

Conceptually, we are sampling the scheme. A basis function is the response given by the scheme when emitting a unit pulse at v_b . The unit pulse is defined as

$$\delta_{ib} = \begin{cases} 1 & i = b \\ 0 & i \neq b \end{cases}. \quad (6.3)$$

We use the unit pulse to rewrite the control mesh vertices as a sampling of the control mesh vertices,

$$\mathbf{p}(v_i) = \sum_{b=1}^{N_v} \delta_{ib} \mathbf{p}(v_b), \quad (6.4)$$

This cumbersome identity will be useful. We first unwrap the scheme of section 5, then introduce our identity given in equation 6.4, and finally re-wrap the scheme with some algebraic juggling. The result is an expression on the form given in hypothesis 6.0.1.

From definitions 5.6.1, 5.5.1, 5.4.2 and 5.3.1 we see that the scheme can be composed of convex combinations of a “layer below”. The scheme is composed of layers of computation; first, the polyhedral split is performed, and is the layer above the control mesh vertices. Then the quad midpoints are calculated from the polyhedral split vertices, and form the layer above this again, and so on (we skip most of the parameters

for simplicity here).

$$\begin{aligned}
\mathbf{s}(c, s, t) &= \sum w_{bz} \mathbf{bc}(\dots) \\
\mathbf{s}(c, s, t) &= \sum w_{bz} \sum w_{bc} \mathbf{qm}(\dots) \\
\mathbf{s}(c, s, t) &= \sum w_{bz} \sum w_{bc} \sum w_{qm} \mathbf{ps}(\dots) \\
\mathbf{s}(c, s, t) &= \sum w_{bz} \sum w_{bc} \sum w_{qm} \sum_i w_{ps} \mathbf{p}(v_i) \\
\mathbf{s}(c, s, t) &= \sum w_{bz} \sum w_{bc} \sum w_{qm} \sum_i w_{ps} \underbrace{\sum_{b=1}^{N_v} \delta_{ij} \mathbf{p}(v_b)}_{\text{from eq. 6.4}}
\end{aligned} \tag{6.5}$$

We will prove that we may move the sum over b outside the nesting of the computation step by step,

$$\begin{aligned}
\mathbf{s}(c, s, t) &= \sum w_{bz}(u, v) \sum w_{bc}(c) \sum w_{qm} \overbrace{\sum_b \sum_i w_{ps} \delta_{ib} \mathbf{p}(v_b)}^{\mathbf{ps}(\dots)} \\
&\quad \underbrace{B_{ps}(v_b; \dots)} \\
\mathbf{s}(c, s, t) &= \sum w_{bz}(u, v) \sum w_{bc}(c) \overbrace{\sum_b \sum_i w_{qm} B_{ps}(v_b; \dots) \mathbf{p}(v_b)}^{\mathbf{qm}(\dots)} \\
&\quad \underbrace{B_{qm}(v_b; \dots)} \\
\mathbf{s}(c, s, t) &= \sum w_{bz}(u, v) \overbrace{\sum_b \sum_i w_{bc}(c) B_{qm}(v_b; \dots) \mathbf{p}(v_i)}^{\mathbf{bc}(\dots)} \\
&\quad \underbrace{B_{bc}(v_b; \dots)} \\
\mathbf{s}(c, s, t) &= \sum_b \underbrace{\sum_i w_{bz} B_{bc}(v_b; \dots) \mathbf{p}(v_b)}_{B(v_b; \dots)} \\
\mathbf{s}(c, s, t) &= \sum_b B(v_b; \dots) \mathbf{p}(v_i),
\end{aligned} \tag{6.6}$$

and we end up proving hypothesis 6.0.1. The un-nesting done in equation 6.5 are proved in the opposite direction in the definitions of the previous section. To overcome extremely tedious bookkeeping, we will perform the change of summation and re-nesting of equation 6.6 step by step, and not all steps at once.

We will also see that the simplified surface spline has good local support properties. We will prove a corollary for each step determining the size of the support of the expressions of the step, and combine this to find the support of the scheme in total. We start with the polyhedral split.

6.2 Layer II: Quad mesh points

Lemma 6.2.1. *The vertices of the polyhedral split layer can be written as*

$$\mathbf{ps}(\dots) = \sum_{b=1}^{N_v} B_{ps}(v_b; \dots) \mathbf{p}(v_b), \quad (6.7)$$

where \dots are either a vertex, an edge or a face. There are three cases in this layer: the vertex (V), the edge midpoint (EM) and the face midpoint (FM),

$$\begin{aligned} B_{ps}|_V(v_b; v_i) &= \delta_{ib} \\ B_{ps}|_{EM}(v_b; e_j) &= \frac{1}{2} \sum_{v_l \in e_j} \delta_{lb} \\ B_{ps}|_{FM}(v_b; f_k) &= \frac{1}{\#\hat{f}_k} \sum_{v_l \in e_j} \delta_{lb}, \end{aligned} \quad (6.8)$$

where \hat{f}_k is the vertices of face f_k as defined in definition 5.3.1.

Proof. From definition 5.3.1 we have

$$\mathbf{ps}|_V(v_i) = \mathbf{p}(v_i). \quad (6.9)$$

We introduce an extra summation,

$$\mathbf{ps}|_V(v_i) = \sum_{b=1}^{N_v} \delta_{ib} \mathbf{p}(v_b), \quad (6.10)$$

The equations 6.9 and 6.10 are equivalent. The expression for vertices in the layer of the polyhedral split, in the case of a vertex, are then

$$B_{ps}|_V(v_b; v_i) = \delta_{ib}. \quad (6.11)$$

Now, let's look at the edge midpoints of this layer, from definition 5.3.1

$$\begin{aligned} \mathbf{ps}|_{EM}(e_j) &= \frac{1}{2} \sum_{v_l \in e_j} \mathbf{p}(v_l) \\ &= \frac{1}{2} \sum_{v_l \in e_j} \sum_{b=1}^{N_v} \delta_{lb} \mathbf{p}(v_l) \\ &= \sum_{b=1}^{N_v} \left(\frac{1}{2} \sum_{v_l \in e_j} \delta_{lb} \right) \mathbf{p}(v_l) \\ &= \sum_{b=1}^{N_v} B_{ps}|_{EM}(v_b; e_j) \mathbf{p}(v_l). \end{aligned} \quad (6.12)$$

The change of summation in the two middle rows of equation 6.12 is appropriate, and

this enables us to find the expressions. Finally, the case of the face midpoints,

$$\begin{aligned}
 \mathbf{ps}|_{FM}(f_k) &= \frac{1}{\#\hat{f}_k} \sum_{v_k \in \hat{f}_k} \mathbf{p}(v_l) \\
 &= \sum_{b=1}^{N_v} \frac{1}{\#\hat{f}_k} \sum_{v_k \in \hat{f}_k} \delta_{lb} \mathbf{p}(v_b) \\
 &= \sum_{b=1}^{N_v} B_{ps}|_{FM}(v_b; f_k) \mathbf{p}(v_l),
 \end{aligned} \tag{6.13}$$

and this proves lemma 6.2.1. \square

Corollary 6.2.2. *The functions $B_{ps}(v_b; \dots)$ are*

$$\begin{aligned}
 B_{ps}|_V(v_b; v_i) &= 0 \quad \text{if } v_b \neq v_i, \\
 B_{ps}|_{EM}(v_b; e_j) &= 0 \quad \text{if } v_b \notin e_j \text{ and} \\
 B_{ps}|_{FM}(f_k) &= 0 \quad \text{if } v_b \notin \hat{f}_k.
 \end{aligned} \tag{6.14}$$

Proof. This follows easily from the definitions of B_{ps} given in lemma 6.2.1. \square

The expressions of the polyhedral split layer are non-zero at the vertex v_b , the midpoint of edges connected to v_b , and at the face midpoints of faces abutting v_b . Otherwise, the functions are zero.

6.3 Layer III: Quad midpoints

The vertices in the quad midpoint layer are convex combinations of the vertices in the polyhedral split layer. There are two cases of a quad midpoint:

1. it is in face the midpoint of a quad, or
2. it is the midpoint of an edge quartile.

Item (2) can be a bit confusing. The quad midpoint layer consists of a midpoint of each quad in the mesh *as well as the midpoint of each boundary quartile*. Remember from chapter 3 that the sub-interval midpoints played an equivalent role as the quad midpoints. The boundary of a simplified surface spline surface is defined by a surface spline curve, and the edge quartiles on the boundary are in fact the sub-edge midpoints of this boundary curve.

Lemma 6.3.1. *The points of the quad midpoint layer can be written as*

$$\mathbf{qm}(\dots) = \sum_{b=1}^{N_v} B_{qm}(v_b; \dots) \mathbf{p}(v_b), \tag{6.15}$$

where \dots are either a corner or an edge quartile. There are two cases in this layer, the regular quad midpoint, and the edge quartiles for use in boundary-cases. The

expressions are

$$\begin{aligned}
B_{qm}|_{QM}(v_b; v_i, f_k) &= (1 - b_{in})(1 - b_{out}) \begin{cases} 1 & \text{if } v_b = v_i \\ 0 & \text{otherwise} \end{cases} \\
&+ b_{in}(1 - b_{out}) \begin{cases} 1/2 & \text{if } v_b \in e_{in} \\ 0 & \text{otherwise} \end{cases} \\
&(1 - b_{in})b_{out} \begin{cases} 1/2 & \text{if } v_b \in e_{out} \\ 0 & \text{otherwise} \end{cases} \\
&b_{in}b_{out} \begin{cases} 1/\#f & \text{if } v_b \in \hat{f}_k \\ 0 & \text{otherwise,} \end{cases}
\end{aligned} \tag{6.16}$$

and

$$\begin{aligned}
B_{qm}|_{EQ}(v_b; v_i, e_j) &= (1 - b) \begin{cases} 1 & \text{if } v_b = v_i \\ 0 & \text{otherwise} \end{cases} \\
&+ b \begin{cases} 1/2 & \text{if } v_b \in e_j \\ 0 & \text{otherwise.} \end{cases}
\end{aligned} \tag{6.17}$$

The edges e_{in} and e_{out} are the incoming and outgoing edge of $c = (v_i, f_k)$, and b_{in} is the blend ratio associated with $\mathfrak{M}[v, e, f; v_i, e_{in}, f_k]$ and similarly for b_{out} . We assume that $B_{qm}|_{EQ}$ are only evaluated when e_j is a boundary edge, and thus, $\mathfrak{M}[v, e, f; v_i, e_j, *]$ returns a single dart, and b is the blend ratio associated with this dart.

Proof. We start with the case of a quad midpoint. From definition 5.4.2 and lemma 6.2.1,

$$\begin{aligned}
\mathbf{qm}|_{QM}(v_i, f_k) &= (1 - b_{in})(1 - b_{out}) \sum_{v_b=1}^{N_v} B_{ps}|_V(v_b; v_i) \mathbf{p}(v_b) + \\
&b_{in}(1 - b_{out}) \sum_{v_b=1}^{N_v} B_{ps}|_{EM}(v_b; e_{in}) \mathbf{p}(v_b) + \\
&(1 - b_{in})b_{out} \sum_{v_b=1}^{N_v} B_{ps}|_{EM}(v_b; e_{out}) \mathbf{p}(v_b) + \\
&b_{in}b_{out} \sum_{v_b=1}^{N_v} B_{ps}|_{FM}(v_b; f_k) \mathbf{p}(v_b).
\end{aligned} \tag{6.18}$$

We may move the summation of v_b outside the expression, and insert the expressions

for B_{ps} ,

$$\begin{aligned}
\mathbf{qm}|_{QM}(v_i, f_k) &= \sum_{v_b=1}^{N_v} \left((1 - b_{in})(1 - b_{out})\delta_{bi} + \right. \\
&\quad b_{in}(1 - b_{out}) \sum_{v_k \in e_{in}} \frac{\delta_{bk}}{2} + \\
&\quad (1 - b_{in})b_{out} \sum_{v_k \in e_{out}} \frac{\delta_{bk}}{2} + \\
&\quad \left. b_{in}b_{out} \sum_{v_k \in \hat{f}_k} \frac{\delta_{bk}}{1} \# f_k \right) \mathbf{p}(v_b).
\end{aligned} \tag{6.19}$$

Only one term of the sums may be nonzero for each v_b , and thus, we get equation 6.16. We do the same for the edge quartiles. We assume e_j to be a boundary edge, and from the same definition and lemma we get

$$\begin{aligned}
\mathbf{qm}|_{EQ}(v_i, e_j) &= (1 - b) \sum_{v_b=1}^{N_v} B_{ps}|_V(v_b; v_i) \mathbf{p}(v_b) + b \sum_{v_b=1}^{N_v} B_{ps}|_{EM}(v_b; e_j) \mathbf{p}(v_b) \\
&= \sum_{v_b=1}^{N_v} \left((1 - b)\delta_{ib} + b \sum_{v_k \in e_j} \frac{\delta_{ik}}{2} \right) \mathbf{p}(v_b).
\end{aligned} \tag{6.20}$$

Here as well, only one term of the sum is nonzero, and thus we get equation 6.17 of lemma 6.3.1. \square

From the nature of the control mesh, we see that the quad midpoints of this layer can be in four different situations in respect of v_b . Either

1. The vertex $v_b = v_i$, and thus $v_b \in e_{in}$, $v_b \in e_{out}$ and $v_b \in \hat{f}_k$. Thus all four of the terms in the bilinear blend are nonzero.
2. The vertex $v_i \neq v_b$ but either $v_b \in e_{in}$ or $v_b \in e_{out}$. Then $v_b \in \hat{f}_k$ and two of the terms are nonzero.
3. The vertex $v_b \notin e_{in}$ or $v_b \notin e_{out}$, and thus $v_b \neq v_i$ but $v_b \in \hat{f}_k$. Then only the term of the face midpoint is nonzero.
4. If $v_b \notin \hat{f}_k$, all of the terms are zero.

Situation (4) is especially interesting, and we will record this in a corollary. We call the set of nonzero patch midpoints for a particular vertex v_b the quad midpoint influence of v_b (see figure 6.1). This will be used when deducing the support of the expressions of other kinds of coefficients.

Corollary 6.3.2. *The functions $B_{qm}(v_b; \dots)$ are zero when*

$$\begin{aligned}
B_{qm}|_{QM}(v_b; v_i, f_k) &= 0 \quad \text{if } v_b \notin \hat{f}_k \\
B_{qm}|_{EQ}(v_b; v_i, e_j) &= 0 \quad \text{if } v_b \notin e_j.
\end{aligned} \tag{6.21}$$

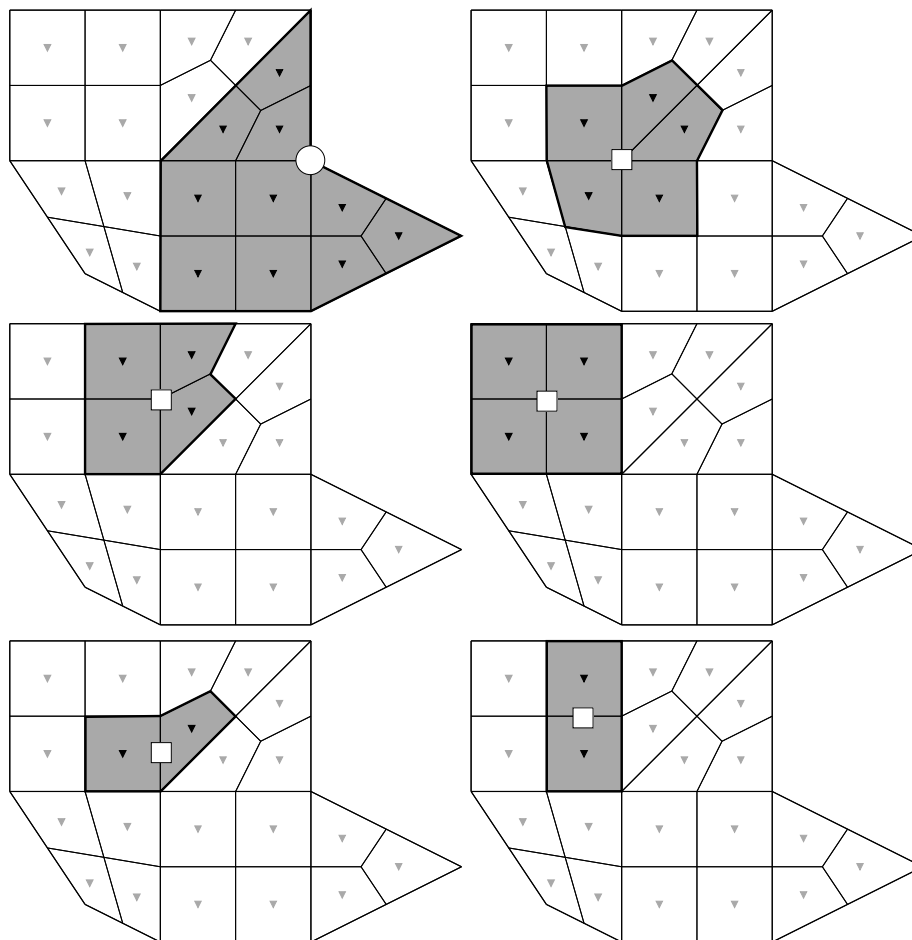


Figure 6.1: Deduction of support from the influence of v_b and sources of computation for the various coefficients. Upper left: The influence of v_b on the quad midpoints ▼. The gray quad midpoints are zero and the black quad midpoints are not necessarily nonzero. Upper right: The source of computation for original vertices (white square). The black quad midpoints are used in the calculation of this coefficient, while the gray ones are not used. Middle left: The source of computation for edge midpoints. Middle right: The source of computation for face midpoints. Lower left: The source of computation for edge quartiles. Lower right: The source of computation for inner edge quartiles.

Proof. From the fundamental concept of a mesh, the incoming and outgoing edge must be a part of the face the corner belongs to. Also, both of these two edges must connect to the vertex of the corner. Thus, if a vertex is not part of the face of the corner, the vertex cannot be the vertex of the corner or one of the end vertices of the incoming and outgoing edge. This if $v_b \notin \hat{f}_k$, then all four terms are zero, and therefore the result is zero. Similarly, if a vertex is not one of the two end vertices of an edge, then it obviously cannot be one of the two end vertices of the same edge. Thus, if $v_b \notin e_j$, then both terms are zero in the case of edge quartiles. \square

6.4 Layer IV: Bézier coefficients

The Bézier coefficients are convex combinations of either a set of quad midpoints of the layer below, or a set of edge quartiles.

Lemma 6.4.1. *The points of the Bézier coefficients layer can be written as*

$$\mathbf{bc}(\dots) = \sum_{b=1}^{N_v} B_{bc}(v_b; \dots) \mathbf{p}(v_b), \quad (6.22)$$

where

$$B_{bc}(v_b; \dots) = \begin{cases} \frac{1}{\#\mathbb{C}} \sum_{(v,f) \in \mathbb{C}} B_{qm}|_{QM}(v_b; v, f) & \text{if } \dots \text{ is non-boundary} \\ \frac{1}{\#\mathbb{EQ}} \sum_{(v,e) \in \mathbb{EQ}} B_{qm}|_{EQ}(v_b; v, e) & \text{if } \dots \text{ is boundary,} \end{cases} \quad (6.23)$$

that is, the expressions are averages of quad midpoints in the non-boundary case, and averages of the edge quartiles in the boundary case. The sets of which the averages are taken are defined in definition 5.5.1.

Proof. From definition 5.5.1 and lemma 6.3.1 we know that

$$\mathbf{bc}(\dots) = \begin{cases} \frac{1}{\#\mathbb{C}} \sum_{(v,f) \in \mathbb{C}} \sum_{v_b=1}^{N_v} B_{qm}|_{QM}(v_b; v, f) \mathbf{p}(v_b) & \text{if } \dots \text{ is non-boundary} \\ \frac{1}{\#\mathbb{EQ}} \sum_{(v,e) \in \mathbb{EQ}} \sum_{v_b=1}^{N_v} B_{qm}|_{EQ}(v_b; v, e) \mathbf{p}(v_b) & \text{if } \dots \text{ is boundary} \end{cases} \quad (6.24)$$

In both cases we can swap the two summations and,

$$\mathbf{bc}(\dots) = \begin{cases} \sum_{v_b=1}^{N_v} \left(\frac{1}{\#\mathbb{C}} \sum_{(v,f) \in \mathbb{C}} B_{qm}|_{QM}(v_b; v, f) \right) \mathbf{p}(v_b) & \text{non-boundary} \\ \sum_{v_b=1}^{N_v} \left(\frac{1}{\#\mathbb{EQ}} \sum_{(v,e) \in \mathbb{EQ}} B_{qm}|_{EQ}(v_b; v, e) \right) \mathbf{p}(v_b) & \text{boundary} \end{cases} \quad (6.25)$$

and by defining the expressions inside the big parentheses as the expressions, we get lemma 6.4.1. \square

We let \mathbb{Q} and \mathbb{EQ} be the sets of quad midpoints or edge quartiles used in the computation of a point, and let \mathbb{I}_{qm} and \mathbb{I}_{eq} be the influenced quad midpoints and edge quartiles respectively for a given vertex. In general, we have

$$B_{bc}(v_b; \dots) = 0 \quad \text{if} \quad \begin{cases} \mathbb{Q} \cap \mathbb{I}_{qm}(v_b) = \emptyset & \text{if non-boundary} \\ \mathbb{EQ} \cap \mathbb{I}_{eq}(v_b) = \emptyset & \text{if boundary.} \end{cases} \quad (6.26)$$

The specific cases are given in the following corollary.

Corollary 6.4.2. *The expressions used in the computation of a point in the Bézier coefficient layer are zero for a basis vertex v_b when*

1.

$$B_{bc}|_V(v_b; v_i) \equiv 0 \quad \text{if} \quad \begin{cases} v_i \notin \mathfrak{M}[v, *, *, f_k] & \forall f_k \in \mathfrak{M}[f; v_b, *, *] \\ \Downarrow \\ v_b \notin \mathfrak{M}[v, *, *, f_k] & \forall f_k \in \mathfrak{M}[f; v_i, *, *]. \end{cases} \quad (6.27)$$

These two tests are equivalent.

2. Both

$$\left. \begin{array}{l} B_{bc}|_{EM}(v_b; e_j) \equiv 0 \\ B_{bc}|_{EQ}(v_b; v_i, e_j) \equiv 0 \end{array} \right\} \quad \text{if} \quad e_j \notin \mathfrak{M}[e, *, *, f_k] \quad \forall f_k \in \mathfrak{M}[f; v_b, *, *]. \quad (6.28)$$

3. Both

$$\left. \begin{array}{l} B_{bc}|_{FM}(v_b; f_k) \equiv 0 \\ B_{bc}|_{IQ}(v_b; e_j, f_k) \equiv 0 \end{array} \right\} \quad \text{if} \quad \begin{cases} f_k \notin \mathfrak{M}[f; v_b, *, *] \\ \Downarrow \\ v_b \notin \hat{f}_k. \end{cases} \quad (6.29)$$

These two tests are equivalent.

The proof is rather argumentative, and figure 6.1 can be of good help when examining this proof.

Proof. We will begin with the non-boundary case. We make some observations of $\mathbb{I}_{qm}(v_b)$: The corners in this set form full faces, that is, either all or none of the corners of a face is in the set.

This implies that if a face is “in” the influence region, all of the inner edge quartiles and the face midpoint as well are possible nonzero. However, if the face is not inside, which means that none of the corners use this face, all of the before-mentioned coefficients are exactly equal to zero. Thus, the test for face midpoints and inner edge quartiles are the same.

This also implies that an edge can either (i) be inside the region, that is, both abutting faces are in the region, or (ii) the edge is on the border of the region, that is one of the abutting faces are in the region while the other is not or (iii) the edge is not in the region at all. Thus, if the edge is either (i) or (ii), the edge midpoint is influenced by quad midpoints in the influence region. If the edge is (iii), the edge cannot be influenced since both of the two faces used in all corners around the edge are not present in the influence region. Further, since both of the corners on one side of the edge are either both in or both out of the influence region, the test for edge quartiles are the same as for edge midpoints.

Test (1a) tests whether v_i is one of the vertices of the faces abutting v_b . If this is not the case, then obviously none of the surrounding corners of v_i is in the influence region.

To verify that (1a) and (1b) are equivalent, we remark that if v_i is one of the vertices of faces abutting v_b , then v_b must also be a vertex of this same face abutting v_i . Thus, this test is symmetric.

Test (2) tests whether e_j is one of the edges bounding the faces abutting v_b , thus this test checks if the edge e_j is of type (iii) or not.

Test (3a) tests whether f_k is one of the faces abutting v_b . If this is not the case, then as implied, both the face midpoint and the inner edge quartiles associated with f_k are equal to zero. Test (3b) is equivalent since if a face is one of the faces abutting a vertex, then this vertex must be in this face.

Let us turn to the boundary case. In fact, we can make some sharper tests at the boundary, but we will use the same test for simplicity.

For a boundary vertex v_i to be nonzero, the basis vertex v_b must satisfy either (i) $v_b = v_i$ or (ii) v_b must be in one of the two boundary edges connecting to v_i . Thus, if any of the tests in (1) are true, then neither of (i) or (ii) could be true.

For an edge midpoint (e_j) or an edge quartile (v_i, e_j) to be nonzero, the basis vertex v_b must be either of the two vertices in the edge. Thus, if e_j is not in any face abutting v_b , then this cannot be the case. \square

6.5 Layer V: The surface in terms of basis functions

Theorem 6.5.1. *The surface can be written as a linear combination of the basis functions, that is, a linear combination of the control points with the basis functions as weights,*

$$\mathbf{s}(c, s, t) = \sum_{b=1}^{N_b} B(v_b; c, s, t) \mathbf{p}(v_b), \quad (6.30)$$

where

$$\begin{aligned} B(v_b; c, s, t) = & (1-s)^2(1-t)^2 B_{bc|V}(v_b; v) + \\ & 2(1-s)^2(1-t)t B_{bc|EQ}(v_b; v, e_{out}) + \\ & (1-s)^2 t^2 B_{bc|EM}(v_b; e_{out}) + \\ & 2(1-s)s(1-t)^2 B_{bc|EQ}(v_b; v, e_{in}) + \\ & 4(1-s)s(1-t)t B_{bc|QM}(v_b; v, f) + \\ & 2(1-s)st^2 B_{bc|IQ}(v_b; e_{out}, f) + \\ & s^2(1-t)^2 B_{bc|EM}(v_b; e_{in}) + \\ & 2s^2(1-t)t B_{bc|IQ}(v_b; e_{in}, f) + \\ & s^2 t^2 B_{bc|FM}(v_b; f), \end{aligned} \quad (6.31)$$

and where v, f, e_{in} and e_{out} are the vertex, face, incoming edge and outgoing edge of corner $c = (v_i, f_k)$ respectively.

Proof. From definition 5.6.1 and lemma 6.4.1 we have

$$= B(v_b; c, s, t) \mathbf{p}(v_b), \quad (6.32)$$

where

$$\begin{aligned}
\mathbf{s}(c, s, t) = & (1-s)^2(1-t)^2 \sum_{b=1}^{N_b} B_{bc}|_V(v_b; v) \mathbf{p}(v_b) + \\
& 2(1-s)^2(1-t)t \sum_{b=1}^{N_b} B_{bc}|_{EQ}(v_b; v, e_{out}) \mathbf{p}(v_b) + \\
& (1-s)^2t^2 \sum_{b=1}^{N_b} B_{bc}|_{EM}(v_b; e_{out}) \mathbf{p}(v_b) + \\
& 2(1-s)s(1-t)^2 \sum_{b=1}^{N_b} B_{bc}|_{EQ}(v_b; v, e_{in}) \mathbf{p}(v_b) + \\
& 4(1-s)s(1-t)t \sum_{b=1}^{N_b} B_{bc}|_{QM}(v_b; v, f) \mathbf{p}(v_b) + \\
& 2(1-s)st^2 \sum_{b=1}^{N_b} B_{bc}|_{IQ}(v_b; e_{out}, f) \mathbf{p}(v_b) + \\
& s^2(1-t)^2 \sum_{b=1}^{N_b} B_{bc}|_{EM}(v_b; e_{in}) \mathbf{p}(v_b) + \\
& 2s^2(1-t)t \sum_{b=1}^{N_b} B_{bc}|_{IQ}(v_b; e_{in}, f) \mathbf{p}(v_b) + \\
& s^2t^2 \sum_{b=1}^{N_b} B_{bc}|_{FM}(v_b; f) \mathbf{p}(v_b), \tag{6.33}
\end{aligned}$$

The sum is common for all the terms, and thus can be moved outside the expressions. This applies to $\mathbf{p}(v_b)$ as well. We then get

$$\begin{aligned}
\mathbf{s}(c, s, t) = & \sum_{b=1}^{N_b} [(1-s)^2(1-t)^2 B_{bc}|_V(v_b; v) + \\
& 2(1-s)^2(1-t)t B_{bc}|_{EQ}(v_b; v, e_{out}) + \\
& (1-s)^2t^2 B_{bc}|_{EM}(v_b; e_{out}) + \\
& 2(1-s)s(1-t)^2 B_{bc}|_{EQ}(v_b; v, e_{in}) + \\
& 4(1-s)s(1-t)t B_{bc}|_{QM}(v_b; v, f) + \\
& 2(1-s)st^2 B_{bc}|_{IQ}(v_b; e_{out}, f) + \\
& s^2(1-t)^2 B_{bc}|_{EM}(v_b; e_{in}) + \\
& 2s^2(1-t)t B_{bc}|_{IQ}(v_b; e_{in}, f) + \\
& s^2t^2 B_{bc}|_{FM}(v_b; f)] \mathbf{p}(v_b). \tag{6.34}
\end{aligned}$$

The expression inside the brackets are indeed the basis functions defined in theorem 6.5.1, and thus this theorem is proved. \square

The parameter values (c, u, v) where a basis function for a particular vertex of the control mesh is nonzero is called *the support of v_b* . Actually, we will not be that specific, we will just denote the set of patches (indexed by the corner) which is nonzero.

Corollary 6.5.2. A patch $c = (v, f)$ is outside the support of v_b if

$$v_i \notin \left\{ \bigcup_{f_k \in \mathfrak{M}[f; v_b, *, *]} \mathfrak{M}[v; *, *, f_k] \right\} \quad (6.35)$$

that is, if v_b is not in any face abutting v_i , or equivalently

$$v_b \notin \left\{ \bigcup_{f_k \in \mathfrak{M}[f; v_i, *, *]} \mathfrak{M}[v; *, *, f_k] \right\} \quad (6.36)$$

Otherwise, the patch may have at least one non-zero coefficient.

Proof. If v_b is not in any face abutting v_i , then neither is any face abutting one of the two associated edges (since these faces abuts v_i as well as the edges), and v_b is not in f (since f by definition abuts v_i), and finally, $v_i \neq v_b$. With this information, we know from corollary 6.4.2 that none of the blended expressions are nonzero. The symmetry of the condition is proved in corollary 6.4.2. \square

6.6 The simplified surface spline space

In the previous section we found a set of functions which can be used as weights when performing the surface evaluation as one single linear combination. We called these functions basis functions, since they are suitable candidates for a simplified surface spline basis.

6.6.1 The space

We let B_i denote the basis function of vertex v_i . We put all the $n = N_v$ basis functions of the mesh \mathfrak{M} into a row vector,

$$\mathbf{B}(c, s, t) = [B(v_1; c, s, t) \quad B(v_2; c, s, t) \quad \dots \quad B(v_n; c, s, t)], \quad (6.37)$$

where $B(v_b; c, s, t)$ is defined in theorem 6.5.1, and the control mesh vertices in a column vector,

$$\mathbf{p} = \begin{bmatrix} \mathbf{p}(v_1) \\ \mathbf{p}(v_2) \\ \vdots \\ \mathbf{p}(v_n) \end{bmatrix}. \quad (6.38)$$

The simplified surface spline is then the vector product of the two vectors,

$$\mathbf{s}(c, s, t) = \mathbf{B}(c, s, t)\mathbf{p}. \quad (6.39)$$

Definition 6.6.1. A *simplified surface spline space* \mathcal{S} is defined by the connectivity and the blend ratios of a mesh, and is populated by all simplified surface splines with this mesh. An object in this space is specified by a vector of control points, one control point associated with each vertex in the mesh. Addition and scalar multiplication of object in this space is defined by addition of the control point vectors of the objects.

Theorem 6.5.1 states that any object in \mathcal{S} , can be written on the form of equation 6.39.

Lemma 6.6.2. The simplified surface spline space \mathcal{S} is a linear vector space.

6.6.2 Linear independence

We define what we mean by linearly independent basis functions, and how this relates to the points in the quad midpoint layer. If the surface is closed, the quad midpoint layer consists only of quad midpoints, otherwise, the surface has a boundary, and each boundary edge contributes two edge quartiles to this layer.

Definition 6.6.3. *A set of basis functions are **linearly independent** if and only if the zero vector gives the zero surface,*

$$\begin{aligned} B(c, s, t)\mathbf{p} = 0 \quad \forall (c \in \mathfrak{M}[v, f; *, *, *], s \in [0, 1], t \in [0, 1]) \\ \Downarrow \\ \mathbf{p} = [\mathbf{0} \quad \mathbf{0} \quad \dots \quad \mathbf{0}]^T. \end{aligned} \quad (6.40)$$

The zero surface is a surface that is zero for all possible parameter points. We can relate definition 6.6.3 to the points in the quad midpoint layer.

Lemma 6.6.4. *The patches of a simplified surface spline surface can only be exactly equal to zero if and only if all the points of the quad midpoint layer are equal to zero.*

Proof. The quad midpoint layer consists of the quad midpoint and the edge quartiles for the boundary edges. The weights used in the Bézier blending on each patch are parametric polynomials, and thus all coefficients must be zero for this to be the zero surface and vice versa.

Thus, if all the points in the quad midpoint layer are zero, all Bézier coefficients are zero, and thus the surface is the zero surface. Conversely; each patch has exactly one exclusive quad midpoint each. If at least one quad midpoint or edge quartile are non-zero, some of the Bézier coefficients are non-zero, and thus, the surface cannot be the zero surface. \square

We can reduce the problem to examine for what conditions all the quad midpoints are equal to zero. Lemma 6.6.4 is the connection between the quad midpoint layer and the zero surface. We will define the quad midpoint matrix, a matrix which transform control mesh vertices into quad midpoints, to examine the connection between the control points and the quad midpoint layer.

Definition 6.6.5. *The **quad midpoint matrix** B_{qm} is defined by*

$$\mathbf{B}_{qm} = \begin{bmatrix} B_{qm}|_{QM}(v_1; c_1) & \dots & B_{qm}|_{QM}(v_n; c_1) \\ \vdots & \ddots & \vdots \\ B_{qm}|_{QM}(v_1; c_m) & \dots & B_{qm}|_{QM}(v_n; c_m) \\ B_{qm}|_{EQ}(v_1; g_1) & \dots & B_{qm}|_{EQ}(v_n; g_1) \\ \vdots & \ddots & \vdots \\ B_{qm}|_{EQ}(v_1; g_l) & \dots & B_{qm}|_{EQ}(v_n; g_l) \end{bmatrix}, \quad (6.41)$$

where c_i , $1 \leq i \leq m$ are the corners, v_j , $1 \leq j \leq n$ are the vertices, and g_j , $1 \leq l \leq n$ are the boundary edge-ends of the control mesh. The expression $B_{qm}|_{QM}$ is defined in lemma 6.3.1.

The matrix B_{qm} is the product of the two matrices $S_{qm} \cdot S_{ps}$ in equation (6.1). Each edge has two edge-ends, and thus B_{qm} has $\#\mathbb{V}$ columns and $\#\mathbb{C} + 2\#\mathbb{BE}$ rows. A vertex has at least one corner, and thus, this matrix has never less rows than columns.

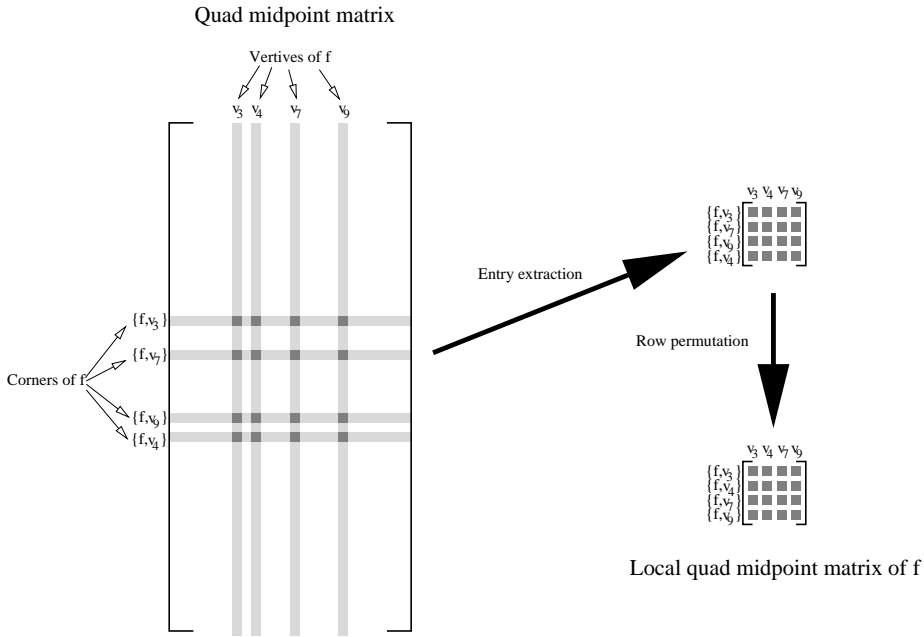


Figure 6.2: Construction of the local quad midpoint matrix.

Lemma 6.6.6. *The simplified surface spline basis functions are linearly independent if and only if the quad midpoint matrix \mathbf{B}_{qm} of definition 6.6.5 has full rank.*

Proof. This follows from lemma 6.6.4. If it is of full rank, only the zero vector give zero quad midpoints. If it isn't of full rank, it has a non-trivial null-space. \square

For the rest of this chapter we will use a one-way implication instead of the two way implication of lemma 6.6.6. In other words, we settle for finding one criterium implicating linear independent basis functions, a criterium not necessarily implied by linear independent basis functions.

Each vertex has at least one abutting corner, and thus, the number of corners is larger or equal to the number of vertices. Each corner corresponds to a row, and thus, we can prove full rank of the quad midpoint matrix by examining the $B_{qm}|_{QM}$ -rows only. For simplicity, we let $\mathbf{B}_{qm}|_{QM}$ be a sub-matrix of the quad midpoint matrix where all $B_{qm}|_{EQ}$ -rows are removed.

6.6.3 A local criterium

We will now show that diagonally dominant local quad midpoint matrices result in a non-singular quad midpoint matrix. Each face has a local quad midpoint matrix, and the entries of this matrix is a subset of the entries of the quad midpoint matrix.

Definition 6.6.7. *The local quad midpoint matrix for face f_k , denoted $\mathbf{B}_{qm}^L|_{QM}(f_k)$, is a subset of the entries of the quad midpoint matrix. First, remove all rows not corresponding to corners of f_k . Then, remove all columns not corresponding to vertices of f_k . Then permute the remaining rows such that the entry of vertex v_i in row $\{f, v_i\}$ lies on the diagonal.*

The construction is depicted in figure 6.2.

Lemma 6.6.8. *The local quad midpoint matrix for face f_k is a square matrix of size $\#f_k \times \#f_k$, and all the removed columns have zero as entries in the remaining rows.*

Proof. There is a one-to-one correspondence between quad midpoints and corners, and a corner can only belong to one face. There are $\#f_k$ vertices in face f_k , and thus the same number of corners. Hence, $\#f_k$ rows and $\#f_k$ columns are retained and the result is a square matrix of size $\#f_k$, see figure 6.2.

Each quad midpoint is a bilinear blend of only the vertices of the face, thus entries of vertices not belonging to f_k on rows belonging to f_k are zero. \square

Each row of the quad midpoint matrix “belongs” to one single face, and thus, the rows of the local quad midpoint matrices do not overlap. We will now prove that if all the local quad midpoint matrices are diagonally dominant, then the quad midpoint matrix $\mathbf{B}_{qm}|_{QM}$ is of full rank, and thus, \mathbf{B}_{qm} is of full rank as well.

The idea is to reverse the construction of definition 6.6.7. First, we pick N_v rows from the local matrices and pad them (the opposite of the column removal). In this way we can build a $N_v \times N_v$ diagonally dominant matrix. By adding the unused rows from the local matrices, and permuting rows this square matrix can be made identical to the quad midpoint matrix.

Lemma 6.6.9. *If all local quad midpoint matrices are diagonally dominant, the quad midpoint matrix is of full rank.*

Proof. For each face f_k , we define a padded local quad midpoint matrix $\hat{\mathbf{B}}_{qm}^L(f_k)$. This matrix is identical to $\mathbf{B}_{qm}^L(f_k)$ except columns of zeros are insert into appropriate places such that the matrix ordering along the columns match \mathbf{B}_{qm} . This matrix is of size $\#f_k \times N_v$.

For each vertex v_i pick one of the abutting faces, which we call f_k . Pick the row corresponding to corner (f_k, v_i) of $\hat{\mathbf{B}}_{qm}^L(f_k)$ and let this be row i of a matrix which we call \mathbf{D} .

The matrix \mathbf{D} is a $N_v \times N_v$ diagonally dominant matrix. By adding the remaining unused rows from the padded local quad midpoint matrices, we can obtain $\mathbf{B}_{qm}|_{QM}$ from \mathbf{D} . The matrix \mathbf{D} is of rank N_v , and is a sub-matrix of the quad midpoint matrix $\mathbf{B}_{qm}|_{QM}$. Hence $\mathbf{B}_{qm}|_{QM}$ is of rank N_v . \square

Note that the converse of lemma 6.6.9 is not necessarily true.

6.6.4 The local quad midpoint matrices

Then, if the local quad midpoint matrices are diagonal dominant, then the quad midpoint matrix is of full rank, and further, the basis functions are linearly independent. Initially, we try blend ratios of $1/2$ and see if we get diagonally dominant local quad midpoint matrices.

We deduce an expression for the local quad midpoint matrix. For simplicity, we rename the vertices of the face f_k as $v_1 \dots v_n$, where $n = \#f_k$. We let indices “wrap” such that $v_{n+1} = v_1$. Further, \mathbf{p}_i is the geometric position associated with v_i . We combine the steps of polyhedral split (definition 5.3.1) and quad midpoint generation (definition 5.4.2) and after some get:

$$\begin{aligned}
\mathbf{qm}_i &= \frac{1}{4}\mathbf{p}_i + \frac{1}{8}(\mathbf{p}_i + \mathbf{p}_{i+1}) + \frac{1}{8}(\mathbf{p}_{i-1} + \mathbf{p}_i) + \frac{1}{4n} \sum_{j=1}^n \mathbf{p}_j \\
&= \frac{1}{2}\mathbf{p}_i + \frac{1}{8}\mathbf{p}_{i+1} + \frac{1}{8}\mathbf{p}_{i-1} + \frac{1}{4n} \sum_{j=1}^n \mathbf{p}_j.
\end{aligned} \tag{6.42}$$

In the case of a triangle we get three expressions

$$\begin{aligned}
\mathbf{qm}_1 &= \frac{1}{24}(14\mathbf{p}_1 + 5(\mathbf{p}_2 + \mathbf{p}_3)) \\
\mathbf{qm}_2 &= \frac{1}{24}(14\mathbf{p}_2 + 5(\mathbf{p}_3 + \mathbf{p}_1)) \\
\mathbf{qm}_3 &= \frac{1}{24}(14\mathbf{p}_3 + 5(\mathbf{p}_1 + \mathbf{p}_2)).
\end{aligned} \tag{6.43}$$

which we can pose in matrix form,

$$\begin{aligned}
\begin{bmatrix} \mathbf{qm}_1 \\ \mathbf{qm}_2 \\ \mathbf{qm}_3 \end{bmatrix} &= \mathbf{B}_{\mathbf{qm}|_{QM}(f_k)} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \\
\begin{bmatrix} \mathbf{qm}_1 \\ \mathbf{qm}_2 \\ \mathbf{qm}_3 \end{bmatrix} &= \frac{1}{24} \begin{bmatrix} 14 & 5 & 5 \\ 5 & 14 & 5 \\ 5 & 5 & 14 \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}.
\end{aligned} \tag{6.44}$$

The local quad midpoint matrix is of size 3×3 , and 14 is larger than $5 + 5$, and thus this matrix is diagonally dominant. If we do the same for a quadrilateral, we get

$$\begin{bmatrix} \mathbf{qm}_1 \\ \mathbf{qm}_2 \\ \mathbf{qm}_3 \\ \mathbf{qm}_4 \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 9 & 3 & 1 & 3 \\ 3 & 9 & 3 & 1 \\ 1 & 3 & 9 & 3 \\ 3 & 1 & 3 & 9 \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \mathbf{p}_4 \end{bmatrix}, \tag{6.45}$$

which is diagonally dominant as well.

By investigating equation (6.42), we can obtain a general formulation for the elements of the local quad midpoint matrix. We let $a_{j,i}$ denote the element at position j, i and get

$$a_{j,i} = \begin{cases} \frac{1}{2} + \frac{1}{4n} & \text{if } i = j \\ \frac{1}{8} + \frac{1}{4n} & \text{if } i = \text{mod}(j-1, n) \text{ or } i = \text{mod}(j+1, n), \\ \frac{1}{4n} & \text{otherwise.} \end{cases} \tag{6.46}$$

The rows are convex combinations, and the diagonal element is $\frac{1}{2} + \frac{1}{4n}$, larger than $1/2$, and thus each row is diagonally dominant.

However, requiring all blend ratios to be equal to $1/2$ is an unnecessary rigid condition, so let us consider what happens in the general case.

Lemma 6.6.10. *The local quad midpoint matrix is diagonally dominant if the blend ratios for each corner of the face satisfy*

$$\frac{1}{2}(1 - b_{in} - b_{out}) + \frac{b_{in}b_{out}}{\#f} > 0, \quad (6.47)$$

where f is the face of the corner.

Proof. We set $n = \#f$. We combine definitions 5.3.1 and 5.4.2, and organise the coefficient in a cyclic order. The expression for the rows of the local quad midpoint matrix is

$$\begin{aligned} \mathbf{pm}_i &= (1 - b_{in}^i)(1 - b_{out}^i)\mathbf{p}_i + \frac{1}{2}(1 - b_{in}^i)b_{out}^i(\mathbf{p}_{i-1} + \mathbf{p}_i) + \\ &\frac{1}{2}b_{in}^i(1 - b_{out}^i)(\mathbf{p}_i + \mathbf{p}_{i+1}) + \frac{b_{in}^ib_{out}^i}{n} \sum_{j=1}^n \mathbf{p}_j. \end{aligned} \quad (6.48)$$

The weight of \mathbf{p}_i is the diagonal entry in the expression of \mathbf{pm}_i . If this is larger than $1/2$, then the row is diagonal dominant. With some algebraic juggling we get

$$\begin{aligned} \mathbf{pm}_i &= \left((1 - b_{in}^i)(1 - b_{out}^i) + \frac{1}{2}(1 - b_{in}^i)b_{out}^i + \frac{1}{2}b_{in}^i(1 - b_{out}^i) + \frac{b_{in}^ib_{out}^i}{n} \right) \mathbf{p}_i + \\ &\left(\frac{1}{2}(1 - b_{in}^i)b_{out}^i + \frac{b_{in}^ib_{out}^i}{n} \right) \mathbf{p}_{i-1} + \\ &\left(\frac{1}{2}b_{in}^i(1 - b_{out}^i) + \frac{b_{in}^ib_{out}^i}{n} \right) \mathbf{p}_{i+1} + \\ &\frac{b_{in}^ib_{out}^i}{n} \sum_{j=1, \dots, i-2, i+2, \dots, n} \mathbf{p}_j, \end{aligned} \quad (6.49)$$

and we pose the following inequality

$$\begin{aligned} (1 - b_{in}^i)(1 - b_{out}^i) + \frac{1}{2}(1 - b_{in}^i)b_{out}^i + \frac{1}{2}b_{in}^i(1 - b_{out}^i) + \frac{b_{in}^ib_{out}^i}{n} &> \frac{1}{2} \\ 1 - \frac{b_{in}^i}{2} - \frac{b_{out}^i}{2} + \frac{b_{in}^ib_{out}^i}{n} - \frac{1}{2} &> 0 \\ \frac{1}{2}(1 - b_{in}^i - b_{out}^i) + \frac{b_{in}^ib_{out}^i}{n} &> 0 \end{aligned} \quad (6.50)$$

Thus, if this requirement is fulfilled, the local quad midpoint matrix is diagonally dominant. \square

Figure 6.3 gives an indication of where the inequality of lemma 6.6.10 is satisfied.

6.6.5 The simplified surface spline space and its basis

We combine the results of this section to define when the basis functions form a basis for \mathcal{S} , the space of simplified surface splines for a given connectivity.

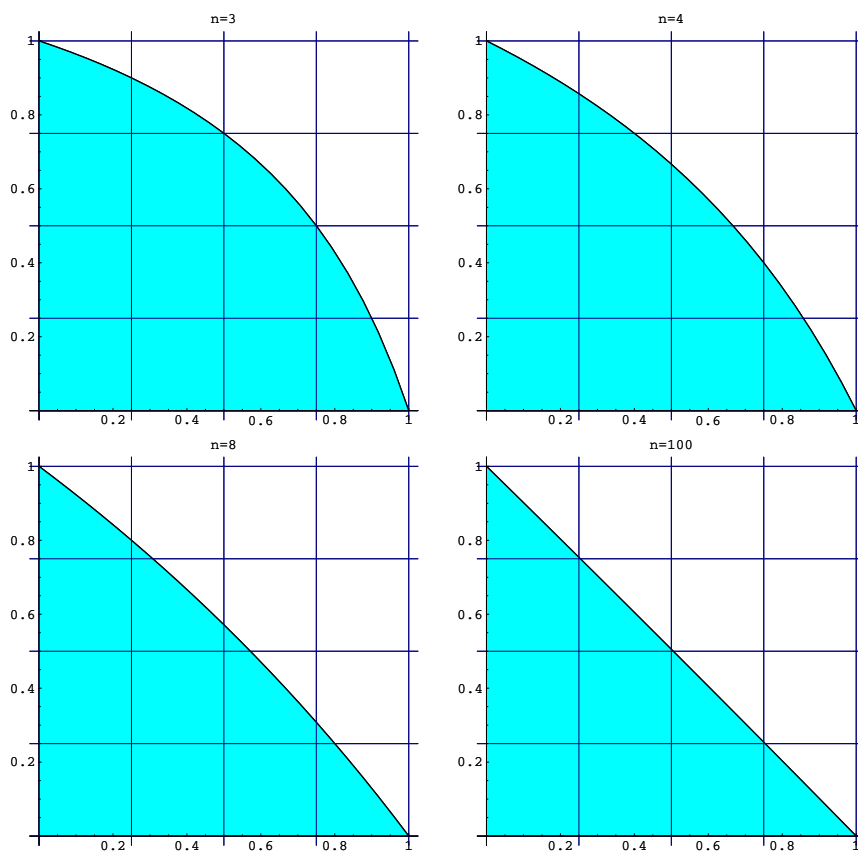


Figure 6.3: The domain where the inequality of lemma 6.6.10 is satisfied for a triangle (upper left), a quadrilateral (upper right), a octagon (lower left) and a 100-sided polygon (lower right). Notice that the domain converges towards $u + v < \frac{1}{2}$ when the vertex count increases.

Theorem 6.6.11. *The simplified surface spline basis functions form a **basis** for \mathcal{S} if (but not necessarily only if) all blend ratios satisfy*

$$\frac{1}{2}(1 - b_{in} - b_{out}) + \frac{b_{in}b_{out}}{\#f_k} > 0. \quad (6.51)$$

In this case, the dimension of \mathcal{S} is N_v , the number of control vertices.

Proof. This follows from combining lemmas 6.6.9 and 6.6.10 and definition 6.6.1. The basis functions are linearly independent and therefore form a basis for the space \mathcal{S} of simplified surface splines. \square

6.7 Summary

In this chapter we have found a set of basis functions for the simplified surface spline. These functions were found by inserting an extra summation over the vertices inside the scheme, and then moving this summation step by step to the outside of the scheme. The result is a sum over the basis functions. We also investigated the support of these basis functions.

In addition, we investigated when this set of basis functions formed a basis. We defined linear independence between basis functions, and linked this definition to the quad midpoints. Further, we found a local criterium for the blend ratios which gives linearly independent basis functions, which in turn forms a basis.

Chapter 7

Practical considerations

In this chapter we investigate some aspects of putting the simplified surface spline into practice. Many of the ideas presented in this chapter apply to C^1 -surface splines, and some ideas apply to any scheme based on convex combinations, for example B-splines and subdivision surfaces.

7.1 Evaluation of layer based schemes

The discussion presented in this section applies to most layer based schemes with a finite number of layers. A layer based scheme is a scheme where each layer represents a set of intermediate points, where each point in one layer is a convex combination of points in the layer below. The initial layer is populated with the control points, and the final layer are the evaluated point on the surface. Both surface splines and B-splines are such schemes.

There exist two major strategies when evaluating the simplified surface spline. The first strategy, called *direct evaluation*, begins with the control points, applies the layers of the scheme, and ends up the position of the surface at the given parameter point.

The other strategy, called *indirect evaluation*, applies the layers in the opposite order. We find the amount of influence each control point has for the given parameter value, and then multiply the this amount of influence with the position of the control point. The sum of this over all the control points is the position of the surface at the given parameter point.

These two strategies are analogous to the two major strategies used to evaluate B-splines. Which to use of these two strategies depends on the context. As a rule of thumb, if we want to evaluate the surface at the same parameter point for many sets of control vertices (for example when performing animation), indirect evaluation is a natural choice. On the other hand, if evaluating the surface at numerous parameter points, and the positions of the control vertices remains fixed, direct evaluation is probably appropriate. In addition, direct evaluation implies less book-keeping, and is thus easier to implement.

7.1.1 Evaluation in matrix guise

Each layer of the simplified surface spline scheme consists of convex combinations of the results of the layer below, and each layer only depends on the layer below. In

chapter 6 we stated that the simplified surface spline evaluation could be posed as a series of matrix multiplications. We have

$$\mathbf{s}(c, s, t) = \mathbf{S}_{bz}(s, t) \cdot \mathbf{S}_{sel}(c) \cdot \mathbf{S}_{bc} \cdot \mathbf{S}_{qm} \cdot \mathbf{S}_{ps} \cdot \mathbf{p} \quad (7.1)$$

Each matrix is a matrix of convex combinations.

7.1.2 Direct and indirect evaluation

There are in essence two ways to carry out these matrix multiplications. Direct and indirect evaluation corresponds to whether the matrices are multiplied together from the left or from the right.

The strategy of direct evaluation applies the matrix \mathbf{S}_{ps} to the control points \mathbf{p} . The result of this multiplication is then multiplied with \mathbf{S}_{qm} and so on. This corresponds to

$$\mathbf{s}(c, s, t) = \mathbf{S}_{bz}(s, t) \cdot \left(\mathbf{S}_{sel}(c) \cdot \left(\mathbf{S}_{bc} \cdot \left(\mathbf{S}_{qm} \cdot (\mathbf{S}_{ps} \cdot \mathbf{p}) \right) \right) \right). \quad (7.2)$$

The parentheses reveal the order of evaluation.

The strategy of indirect evaluation works in the opposite direction. The matrices $\mathbf{S}_{bz}(s, t)$ and $\mathbf{S}_{sel}(c)$ are first multiplied. The result of this is multiplied with \mathbf{S}_{bc} and so on. In the last step the intermediate result is multiplied with \mathbf{p} , the control points. Written as a matrix product this is

$$\mathbf{s}(c, s, t) = \left(\left(\left((\mathbf{S}_{bz}(s, t) \cdot \mathbf{S}_{sel}(c)) \cdot \mathbf{S}_{bc} \right) \cdot \mathbf{S}_{qm} \right) \cdot \mathbf{S}_{ps} \right) \cdot \mathbf{p}. \quad (7.3)$$

The last intermediate result is a row vector. This vector reveals the contribution of each control point to the surface point. In fact, the entries of this vector are the basis functions evaluated at a specific parameter point.

This matrix factorisation of the scheme is of more theoretical importance than of practical use. The matrices are seldom calculated explicitly. However, comparing with the two strategies given in chapter 4 we see that the strategy of section 5 (the scheme) is in fact direct evaluation, while the strategy of section 6 (basis functions) is indirect evaluation.

7.1.3 Retained Bézier evaluation

In the matrix product (equation 7.1) we see that only the two leftmost matrices of the expression are dependent on the parameter value. The rest of the matrices are constant for any parameter point. Thus it may be a good idea to split the evaluation in two; first do the constant part, and then for each parameter point carry out the two multiplications on the left. We get

$$\mathbf{s}(c, s, t) = \underbrace{\mathbf{S}_{bz}(s, t) \cdot \mathbf{S}_{sel}(c)}_{\text{parameter dependent}} \cdot \underbrace{\mathbf{S}_{bc} \cdot \mathbf{S}_{qm} \cdot \mathbf{S}_{ps}}_{\text{constant}} \cdot \mathbf{p}. \quad (7.4)$$

This expression can be split into a parameter dependent part and a constant part. The constant part of the evaluation is performed once for every set of control points. The parameter dependent part is performed once for every evaluation.

By retaining the Bézier evaluation, we can use the constant part to find the set of patches defining the surface. Having the surface represented as a set of patches is often useful for rendering, differentiation, integration, etc.

We can use both the direct and indirect approaches to find the constant part. Usually, the direct approach is the simplest. We can also find the basis functions expressed as a set of Bézier coefficients,

$$\mathbf{B}_{bc} = \mathbf{S}_{bc} \cdot \mathbf{S}_{qm} \cdot \mathbf{S}_{ps}. \quad (7.5)$$

This is simply the constant part where the multiplication with \mathbf{p} is omitted. The matrix \mathbf{B}_{bc} is of size $(N_v + 3N_e + N_f + 2N_c) \times N_v$, and is thus a pretty large matrix. However, from the local support property of the scheme, most of the entries are zero, and thus a full storage and matrix product is not necessary. Each time \mathbf{p} changes, we can find all the new Bézier coefficients by multiplying \mathbf{B}_{bc} and \mathbf{p} , and with a smart sparse representation of the basis matrix, this can be quite effective.

7.2 OpenGL rendering of surfaces with local support

This section deals with OpenGL at a rather detailed level, and many terms specific to rendering are used without definition. Books like (OpenGL ARB, Woo, Neider, Davis & Schreiner 1999) and (Foley et al. 1996) give thorough definitions and discussion of these terms. Even though this section is specific to OpenGL, the discussion applies to Direct3D (MSDN n.d.) as well.

7.2.1 Efficient rendering of Bézier patches

OpenGL has support for Bézier patches of arbitrary degree at core level. The application developer has the choice between sending the coefficients to OpenGL, or to tessellate the patch to triangles in the application. Using OpenGL for patch evaluation is straightforward. Usually, the implementation is quite good, and the speed acceptable. In some cases, one can write specialised code that run faster. On the other hand, due to the arrival of geometry engines on off-the-shelves consumer graphics card, the Bézier evaluation has the potential to be carried out in hardware — quite superior to software calculation in terms of speed. If OpenGL does the evaluation, this will be done in hardware when this is possible.

It is the application's responsibility to calculate the static part of equation 7.4 and perform the actions of the matrix $\mathbf{S}_{sel}(c)$, that is, shoving the correct coefficients for each patch to OpenGL. The best way to obtain the Bézier coefficients varies among different applications. We will give some thoughts on this question.

If the surface is *constant*, that is, the vertices are not changed except for a rigid transformation, direct evaluation is probably most appropriate. First, calculate all the Bézier coefficients. Then, for each patch, create a display list¹ for the evaluation of each patch. Rendering of the visible patches then become a simple call to the respective display list. However, the number of display lists is limited, and therefore, if rendering a high amount of patches, it can be a sound idea to cluster several patches into each display list.

¹Display lists are a mechanism in OpenGL to store a set of OpenGL commands on the server. For example, the rendering of an object could be encapsulated into a display list, and then this object would be rendered by simply calling this display list.

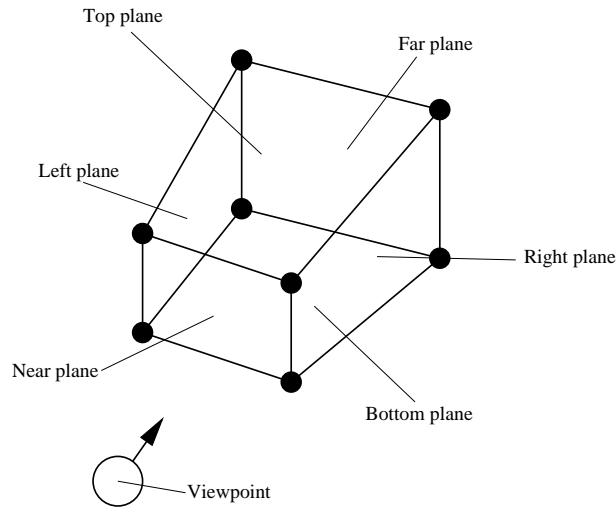


Figure 7.1: The view frustum.

If the model is *dynamic*, that is, the points in p are changed over time (e.g. by animation), display lists are not appropriate. Much effort must then be put into extracting the coefficients in the quickest way possible. Not all coefficients need to be calculated — a large amount of the model could be culled (culling is elaborated on in the next section), and coefficients used exclusively by culled patches are never used. Thus, we must find which coefficients are in need of re-computation, and then pass the visible patches to OpenGL.

7.2.2 Frustum culling

Culling of invisible parts of the scene are the bread and butter of fast rendering code. OpenGL has highly optimised clipping code, and employs hardware to perform this task if this is present. Thus, there is no need to cull at a very detailed level. Out-performing the clipping code of OpenGL is probably quite a challenge.

However, if it can be done without much computation, it is a good idea to cull at object level. This keeps some pressure off the rendering pipeline. We will present an algorithm to cull patches which is *definitely* outside the view frustum.

The view frustum

The view frustum is a convex set with the shape of a trapezoid. It is defined by either the intersection of half-spaces (each defined by a clipping plane) or as the convex hull of the eight points where three of the six before-mentioned clipping planes intersect, see figure 7.1.

The view frustum is defined in *clip coordinates*, and our model is defined in *world coordinates*. Thus, either the model must be transformed to clip coordinates, or the view frustum must be transformed to world coordinates. The latter is the natural choice. For this to make sense, we must take a superficial look at the vertex transformation pipeline. OpenGL does some transformations on the points the application

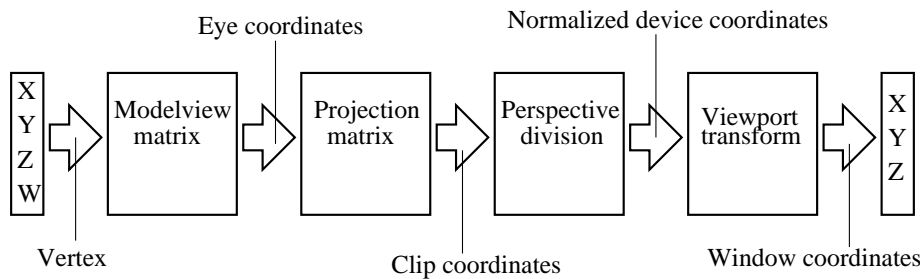


Figure 7.2: The vertex transformation pipeline of OpenGL.

passes into it, and these transformations are known collectively as the vertex transformation pipeline, see figure 7.2.

The vertex transformation pipeline is composed of three matrix multiplications and a division. First, points expressed in *world coordinates* are passed from the application. These points are in homogenous coordinates. The points are converted to *eye coordinates* by the *model view matrix*. The model view matrix defines the orientation of the camera in the world. The eye coordinates are then transformed to *clip coordinates* by the *projection matrix*. Clipping takes place in clip coordinates, and after this, the *perspective division* is done. This division transforms the point from homogenous coordinates to non-homogenous coordinates. Finally, these points are transformed by the *view port transform*, which scales and translates coordinates to match the pixel resolution of the frame buffer.

The model-view matrix and the projection matrix combined define the transformation from world space to clip space, and thus the inverse transforms the view frustum to world coordinates. However, this inversion is not necessary; we can find the view frustum directly from the product of the model-view matrix and the projection matrix. The web site (Gribb 1999) gives an overview on how to extract the half spaces defining the frustum directly.

The view frustum defines in world space the part of the world is visible through the current view. Thus, everything outside the view frustum is invisible.

Culling

The key element to our culling strategy is the local support property of the simplified surface spline, as well as the fact that the scheme is composed of convex combinations. Thus, a patch resides inside the convex hull spanned by the control points which have a non-zero contribution to that particular patch. Therefore, if this convex hull is completely outside the view frustum, the patch is not visible.

All patches of one face (in the control mesh) have the same set of control points with non-zero contribution. Thus, we can deal with all the patches of a face combined.

We test if a vertex is inside or outside the view frustum by checking its position relative to the six half-spaces defining the view frustum. Each half-space is defined by a plane,

$$Ax + By + Cy + D < 0, \quad (7.6)$$

where A , B , C and D are the coefficients from the plane equation. If the inequality is satisfied, the vertex is inside the half space. This check requires three multiplications and three additions and a comparison per half-space.

The algorithm consists of two stages. We let \oplus denote a bitwise OR, and $\%_0$ be a prefix of numbers written in binary form. Further, we define the set $\mathbb{A}(f_k)$ to be the set of control vertices contributing to the patches of f_k (This set can be found with the aid of corollary 6.5.2).

1. **For each vertex** v_i : Check the position of the vertex against the six half-spaces, and store the result as a bit-mask of six bits, denoted $bm(v_i)$. If the vertex is inside a half-space, the corresponding bit is set, otherwise it is not set.
2. **For each face** f_k : We define the bitmask for the face $bm(f_k)$,

$$bm(f_k) = \bigoplus_{v_j \in \mathbb{A}(f_k)} bm(v_j). \quad (7.7)$$

If $bm(f_k) = \%111111$ then render the patches of the face, otherwise cull the patches.

Thus, $bm(f_k)$ is the result from combining all the bitmasks of contributing vertices with a bitwise OR. If at least one of the bits are not set, all of the patches are completely outside a half-space.

We need the bit-mask to store the results from the six tests. Even though all the supporting vertices are completely outside the view frustum, the patch does not necessarily reside completely outside the view frustum. This is why reason for all half space checks are done for each patch.

7.2.3 Occlusion tests

Surfaces inside the view frustum can be occluded by other surfaces. This is usually dealt with either by using the painter’s algorithm or a depth buffer. Evaluating and triangulating patches invisible in the image is a waste of precious processor time. Two vendor specific extensions exists to deal with these problems: the `HP_occlusion_test` and `NV_occlusion_query` extensions. Information on extensions to the OpenGL standard can be found in (*OpenGL Extension Registry* n.d.).

These two extensions provide the means to do a “dry run” of the shape (or a bounding shape), to check if any pixels of this shape will be visible in the image. The dry run do not alter the image, but the full rasterisation process is done, and therefore, the tests have a rasterisation cost.

Rasterising a bunch of pixels, to find out whether to rasterise even more pixels, isn’t always a wise choice — especially if the rasterisation is a bottleneck in the rendering pipeline. Anyway, let’s assume that rasterisation is *not* the bottleneck. Then, to reduce the number of potentially expensive Bézier evaluation and tessellations, we need a simple bounding object to use in the dry run.

Examples of simple occlusion detection shapes are the following: The first and obvious candidate is the convex hull of the control points contributing to the patches of the face. However, the 3D convex hull is not particularly easy and fast to find.

An alternative is to use a coordinate system aligned bounding box. This is not an optimal shape, but we just need the three pairs of minimum and maximum values, and the dry run is the rasterisation of six quadrilaterals.

Instead of using polygons, we may use lines, lowering the pressure at the rasterisation stage. However, testing with lines can fail in pathological cases. With n vertices,

we need $\binom{n}{2}$ lines to connect them all. The number of lines grows quite rapidly when n increases, but usually n is rather low.

We conclude with an important, but not necessarily obvious note. When using occlusion tests, it is important to render the scene from front to back (the opposite of painter's algorithm). This results in a maximum number of objects being culled by the occlusion tests.

7.3 Benefits of using composite polynomial patches

The surface of the surface spline schemes is defined by a set of polynomial patches. We will now discuss two useful aspects of this.

7.3.1 Ray tracing

Raytracing consists of intersecting an object with a parameterised line. The use of raytracing is not limited to the image generation technique called raytracing: Picking and simple collision detection, as an example, may make use of ray tracing. In an interactive application, picking is the task of finding *which* object is pointed to by the mouse. A strategy for this is to transform the pixel coordinates of the mouse pointer into the world space and shoot a ray from this pixel and outwardly away from the eye point and see which object this ray intersects.

An example of simple collision detection is the following: Assume for a moment we are creating a car simulation, in where the world is represented as surface splines. We need to know where the tires touch the ground, what kind of surface this is (asphalt, gravel, grass) to determine the friction, and we must know the normal plane of the surface at each wheel. A strategy to solve these problems is to employ ray tracing. We begin by positioning the object in an approximate position. Then, for each point of contact (for example, one point per wheel), we shoot a ray from this point and in the direction of the gravity pull (usually straight downward). From this, we know whether the points of contact are above or have pierced through the ground, and we then reposition the car.

Thus, raytracing is a common task. Several strategies to raytrace polynomial patches are given in the book (Arvo, Cook, Glassner, Hanrahan, Heckbert & Kirk 1989), in which Hanrahan gives an extensive survey of ray-surface intersection algorithms:

1. We can tessellate the surface into planar triangles. Ray-triangle intersections are easily dealt with.
2. We can convert our patches to implicit functions, and in our case we then get an implicit function of degree eight.
3. We can rewrite the ray as an intersection between two planes and insert this into the expression for the patch. We then get a pair of implicit equations, and by elimination, we can write this as a single equation, and this polynomial will be of degree four.

Algorithm (1) is the most common. However, if the number of rays are few, and we do not need the tessellated surface for anything else, one of the other methods can be suitable.

Algorithms (2) and (3) can be solved by some numerical solver like Newton iteration. Algorithm (3) solves the equation for the parameter coordinates, and this can

be useful if we need additional information about the surface at that point besides the point of intersection (e.g. derivatives and the normal plane).

7.3.2 Mass, centre of gravity and moment of inertia tensor

Often it is useful to find different physical properties of an object. When dealing with rigid body dynamics, as an example, we need the mass, centre of gravity and the moment of inertia tensor to pose a suitable differential equation.

The mass, centre of gravity and moment of inertia tensor can often be calculated quite easily for bodies defined by a simplified surface spline. All of these integrals are volume integrals, and thus we need some way of specifying the limits of the integral from the extent of the object — which is rarely a trivial exercise.

Depending on the density function, these integrals can be simplified from volume integrals to surface integrals by the Gauss divergence theorem. The article (Gonzalez-Ochoa et al. 1998) gives a thorough exposition of this. The key element is that the surface spline surface is defined as a collection of standard Bézier patches, and these are rather simple to perform surface integrals on.

Chapter 8

Approximation experiment

In this chapter we will put the simplified surface spline to practice; we will state some approximation methods using this basis. To our knowledge, surface splines have not been used extensively for approximation, and thus this experiment should be interesting.

8.1 Our choice of parameterisation

We have employed a very simple parameterisation in this experiment. The *original*, that is the object which we try to approximate, is parameterised in the same manner as the simplified surface spline. All of the originals have the same connectivity as the surface spline used to approximate that particular original. Thus, in essence, the originals and the approximation have the same mesh, but have a different scheme and different control points.

The originals are polyhedral meshes, and they one of two surface schemes. The first surface scheme refines the mesh once with polyhedral split, such that all faces in the refined mesh are quads. Each of these quads are then patched with a bilinear Bézier interpolation. The result is a method to define a surface for polyhedral meshes, which gives flat faces if the polyhedron is planar, and can give a well-defined surface if the polyhedron is not planar. Further, since the original is refined once with the polyhedral split, they can be parameterised with the same parameterisation used for simplified surface splines (see section 5.1.3).

The other surface scheme is almost identical to the first surface scheme, except that evaluated points are normalised. The result of this is that all points on the surface is on the unit sphere (a point on the surface is defined as the point where a line from the point returned from the first scheme and the origin pierces the unit sphere). However, to “fill” the complete sphere with surface, the polyhedral mesh used to define the surface must enclose the origin entirely.

There are three types of parameter points we will use extensively, the parameter point of a vertex, the parameter point of an edge midpoint and the parameter point of a face midpoint. When evaluating the surface spline at such parameter points, we can choose one among several abutting patches to evaluate. Since the surface is C^0 -continuous, we get the same result regardless of which of these patches we choose.

Further, both the biquadratic and bilinear Bézier blends used to define the surfaces interpolate their corners, and the corners are located at these three types of parameter

Model	N_v	N_e	N_c	N_f
cube0	8	12	24	6
cube1	26	48	96	24
cube2	98	192	384	96
cube3	386	768	1536	384
cube4	1538	3072	6144	1536
sphere0	8	12	24	6
sphere1	26	48	96	24
sphere2	98	192	384	96
sphere3	386	768	1536	384
sphere4	1538	3072	6144	1536
jittercube	26	48	96	24
bunny	473	1194	2388	723

Table 8.1: Miscellaneous data on the original models. The symbol N_v denote number of vertices, N_e denote number of edges, N_c denote number of corners, and N_f denote number of faces.

points. Thus, we can use the Bézier coefficients directly, and skip the Bézier evaluation altogether.

8.2 The original models

Four different models with different characteristics were employed to test out the four approximation methods. Two of the models, the cube model and the sphere model, were refined into four levels. Thus, twelve models were employed in total.

The blend ratios of all models were set to $1/2$. Probably a lot of approximation power could be gained by using more sensible choices for the blend ratios. However, we chose to keep things simple, and these blend ratios makes the system at least solvable.

The first five of the models have their origin in the standard cube, “cube0”. This cube was refined four times with the polyhedral split method (see section 2.4.2) producing “cube1”, “cube2”, “cube3” and “cube4”. A cube has sharp corners, and these sharp corners were a challenge for the smooth simplified surface spline.

We also wanted to test smooth surfaces without sharp corners. We created five spheres of different level of resolution by normalising the surface (see class `ControlSphere` in section 8.5) of the five cube models. Thus, the sphere and cube models have equal connectivity at each subdivision level.

The “jittercube” is a perturbed version of “cube2”. This model was included to test how the schemes would handle noisy and extreme data.

To test the method on “real-world” data, we used the “bunny”. The “bunny” is a heavily decimated version of the bunny model from Stanford University Computer Graphics Laboratory. This decimation resulted in quite a coarse model, and was therefore a challenge for approximation with a smooth surface. In addition to decimation, many of the triangle pairs forming quadrilaterals have been exchanged with quadrilaterals.

8.3 Error, approximation order and stability

The basis of our error estimate is the geometric distance between the surface of the original and the surface of the approximation for a given parameter point. This distance is calculated at several parameter points, and from these samples the mean and the maximum value are calculated. The mean and the maximum value are used as two estimates of the approximation error.

We will use these two error approximates to compare the different approximation methods, and we will investigate if we can find some information on the approximation order. We assume the error is on the form

$$E_h(f) \sim h^n \| f^{(n)} \|, \quad (8.1)$$

where f is the original, h is a measurement of the size of parameter domain resolution and n is the approximation order. We let h be the maximum length of edges in the polyhedron. The polyhedral split refines each quadrilateral into four quadrilaterals, and thus halves the length of each edge. We therefore assume that h is halved for each refinement.

Thus, if the surface is smooth, we assume that

$$\frac{E_h(f)}{E_{h/2}(f)} \sim \frac{h^n \| f^{(n)} \|}{(h/2)^n \| f^{(n)} \|} \approx 2^n. \quad (8.2)$$

The simplified surface scheme has the convex hull property, and thus, should have an approximation order of at least 2. However, the term $\| f^{(n)} \|$ grows rather large for the cube models. The cube has sharp edges, and thus the first derivative is discontinuous, and the second derivative is therefore infinite. Thus, we anticipate a behaviour of order 1 for the cube approximation experiment and a behaviour of order 2 for the sphere experiment.

To give a notion of the stability of the simplified surface spline basis, we have computed the condition number and the number of iterations of the linear systems to be solved in the approximation experiments.

8.4 Four approximation methods

We have employed four different approximation methods in our experiment. The first, “lazy approximation” is extremely simple. The next “vertex interpolation” is interpolation at each control vertex. The “least squares” increases the number of interpolation constraints, while “faired least squares” adds a number of fairness constraints. The two last methods produce over-determined systems, and we use a least squares construction to minimise the residuals and thus find the best solution.

8.4.1 Lazy approximation

The name of the lazy approximation scheme was inspired by the lazy wavelets. The reason for this name is that nothing at all has to be done. For each vertex, the original is evaluated at the parameter point of this vertex, and the result is simply used as a control point.

It is worth to note that this is indeed the same scheme as the variation diminishing approximation scheme for B-splines, however no variation diminishing properties

have been proved, except for the convex hull property. But the non-existence of these properties hasn't been proved either, and thus, they may exist.

8.4.2 Vertex interpolation

Vertex interpolation is a special case of interpolation. As mentioned in section 8.1, we can use the Bézier coefficients at the vertices directly, and therefore skip the quadratic Bézier evaluation. The linear system to be solved is

$$B_V \cdot \mathbf{p} = \mathbf{F}_V, \quad (8.3)$$

where B_V is the matrix of basis functions for the Bézier coefficients at vertices, \mathbf{p} is the control points we're trying to find, and finally, \mathbf{F}_V is the original evaluated at the vertices. To simplify the notation, we let $n = N_v$, which is the number of vertices in the control mesh. If we write out the matrices, we get

$$\begin{bmatrix} B_{bc|V}(v_1; v_1) & \dots & B_{bc|V}(v_n; v_1) \\ \vdots & \ddots & \dots \\ B_{bc|V}(v_1; v_n) & \dots & B_{bc|V}(v_n; v_n) \end{bmatrix} \begin{bmatrix} \mathbf{p}(v_1) \\ \vdots \\ \mathbf{p}(v_n) \end{bmatrix} = \begin{bmatrix} \mathbf{f}(v_1) \\ \vdots \\ \mathbf{f}(v_n) \end{bmatrix}. \quad (8.4)$$

The matrix B_V is a $N_v \times N_V$ matrix and both the column vectors \mathbf{p} and \mathbf{F}_V have N_v entries. It is essential to know *if* and *when* this system has a solution.

Lemma 8.4.1. *The linear system of equation 8.4 has a solution if B is a basis (theorem 6.6.11).*

Proof. If theorem 6.6.11 is satisfied, the expressions for quad midpoints of corners abutting the vertex of which the basis functions belongs are all larger than $1/2$ (see the proof of lemma 6.6.10). The Bézier coefficient at the vertex are the average of all the abutting quad midpoints.

From this we see that all the diagonal entries are averages of values larger than $1/2$, and therefore this average is larger than $1/2$. Proposition 5.5.2 states that the Bézier coefficients are convex combinations of the control mesh vertices, and therefore the matrix B_V is diagonally dominant, and according to property 1.4.8 this matrix is non-singular. \square

The system is not symmetric and positive definite, and our choice of solver fell on the Gauss-Seidel algorithm.

8.4.3 Least squares

The vertex interpolation method of section 8.4.2 uses only the vertex positions of the original. Each row of the vertex interpolation matrix is an interpolation constraint for a vertex. To capture more of the surface shape, we can use a least squares construction to add more interpolation conditions.

We add rows of further interpolation constraints, but adding more rows to the system results in an over-determined system. Such a system has more constraints than free variables. Hoping for a solution of such a system is rather futile. However, we can find the "best" solution of this problem by using a norm.

With this motivation, we introduce our next method, the least squares method. The least squares method is actually just a method to produce a specific minimum of an

over-determined system, not how to build this system. We will choose a rather simple strategy; we will *try* to interpolate the original at the vertices, the edge midpoints and the face midpoints. This results in $N_v + N_e + N_f$ equations and N_v degrees of freedom. To minimise the error of the system, we will use the 2-norm (see (Trefethen & Bau 1997, p. 18)).

Property 8.4.2. *Given an over-determined set of linear equations $\mathbf{Ax} = \mathbf{b}$, the solution of $\mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{b}$ minimises the 2-norm residual $r = \|\mathbf{b} - \mathbf{Ax}\|_2$. This problem has a solution if and only if \mathbf{A} has full rank.*

The proof is given in any text on linear algebra, e.g. (Trefethen & Bau 1997).

To build the system, we define two matrices and two vectors in addition to \mathbf{B}_V and \mathbf{F}_v , defined in section 8.4.2. We let $n = N_v, m = N_e, l = N_f$ and define:

$$\mathbf{B}_{EM} = \begin{bmatrix} B_{bc|EM}(v_1; e_1) & \dots & B_{bc|EM}(v_n; e_1) \\ \vdots & \ddots & \vdots \\ B_{bc|EM}(v_1; e_m) & \dots & B_{bc|EM}(v_n; e_m) \end{bmatrix} \quad \mathbf{F}_{EM} = \begin{bmatrix} \mathbf{f}_{EM}(e_1) \\ \vdots \\ \mathbf{f}_{EM}(e_m) \end{bmatrix}$$

$$\mathbf{B}_{FM} = \begin{bmatrix} B_{bc|FM}(v_1; f_1) & \dots & B_{bc|EM}(v_n; f_1) \\ \vdots & \ddots & \dots \\ B_{bc|FM}(v_1; f_l) & \dots & B_{bc|EM}(v_n; f_l) \end{bmatrix} \quad \mathbf{F}_{FM} = \begin{bmatrix} \mathbf{f}_{FM}(f_1) \\ \vdots \\ \mathbf{f}_{FM}(f_l) \end{bmatrix}.$$

The entries of the two matrices are the expressions for the edge midpoints and face midpoints of lemma 6.4.1.

To build the complete system, we stack these matrices and vectors on top of each other, and we get the following over-determined linear system of equations:

$$\begin{bmatrix} \mathbf{B}_V \\ \mathbf{B}_{EM} \\ \mathbf{B}_{FM} \end{bmatrix} \begin{bmatrix} \mathbf{p}(v_1) \\ \vdots \\ \mathbf{p}(v_n) \end{bmatrix} = \begin{bmatrix} \mathbf{F}_V \\ \mathbf{F}_{EM} \\ \mathbf{F}_{FM} \end{bmatrix}. \quad (8.5)$$

In this system, there are $n + m + l$ equations and n unknowns. If the blend ratios satisfy the requirements of theorem 6.6.11, we know from lemma 8.4.1 that the matrix \mathbf{B}_V is non-singular, and thus of full rank. Adding rows does not lower the rank of the matrix, and thus, the complete stack of matrices has full rank. By property 8.4.2 we know that we may minimise the residual of equation 8.5. When applying the transposed matrices, the system becomes symmetric and positive definite, and thus, we may solve this system with the conjugate gradient method.

When building these systems, the stacked matrix can become quite large. It is rather easy to build the products $\mathbf{A}^T \mathbf{A}$ and $\mathbf{A}^T \mathbf{b}$ directly. Each element in the product of two matrices is a regular inner product. Thus the entry \hat{a}_{ij} of $\mathbf{A}^T \mathbf{A}$ is the inner product of column i and column j of \mathbf{A} . Similarly, the entries of $\mathbf{A}^T \mathbf{b}$ are inner products of a column of \mathbf{A}^T and \mathbf{b} . The matrices of basis functions are relatively sparse; all entries corresponding to vertices outside the support of the basis vertex are zero. Using this fact can speed up the inner product calculations.

8.4.4 Faired least squares

In the previous section we stacked a lot of interpolation conditions into a composite system matrix to create an over-determined system, which we minimised with the least

squares method. Instead of adding additional interpolation conditions to the vertex interpolation, we will in this method add N_v fairing constraints.

A faired surface is a surface where smoothness has been minimised in some way, resulting in a nice and smooth surface. We will use a rather naive fairing constraint in this method. We want each vertex to lie where the averages of the neighbours of that vertex is. We define

$$g(v_i; v_j) = \begin{cases} -1 & \text{if } v_i = v_j \\ 1/\#\mathfrak{M}[e; v_i, *, *] & \text{if } \{v_j, v_i\} \in \mathfrak{M}[e; v_i, *, *] \\ 0 & \text{otherwise} \end{cases} \quad (8.6)$$

to formulate this mathematically. The expression $g(v_i; v_j)$ is equal to -1 if the vertices are equal, and is one over the number of edges connecting v_i if v_i and v_j are connected. Otherwise, this expression is zero. The sum of all $g(v_i; v_j)$ for a vertex v_i sums to zero.

With this function we define the matrix of fairing conditions,

$$\mathbf{G} = \alpha \begin{bmatrix} g(v_1; v_1) & \dots & g(v_n; v_1) \\ \vdots & \ddots & \vdots \\ g(v_1; v_n) & \dots & g(v_n; v_n) \end{bmatrix}. \quad (8.7)$$

The parameter α defines the amount of contribution the fairing conditions give to the system. Setting $\alpha = 0$ results in an interpolation problem. Each of the rows in \mathbf{G} sum to zero, and the matrix is square and has N_v columns and rows.

By stacking the vertex interpolation conditions and the fairing conditions, and letting $\mathbf{0}$ denote a column vector of N_v zeros (the fairing conditions equals zero), we get the system

$$\begin{bmatrix} \mathbf{B}_V \\ \mathbf{G} \end{bmatrix} \begin{bmatrix} \mathbf{p}(v_1) \\ \vdots \\ \mathbf{p}(v_n) \end{bmatrix} = \begin{bmatrix} \mathbf{F}_V \\ \mathbf{0} \end{bmatrix} \quad (8.8)$$

In this system we have $2N_v$ equations and N_v degrees of freedom. To solve this system, we apply the least squares method of the previous section. As always with these kinds of least squares systems, the linear problem to solve is positive definite and symmetric, and thus the conjugate gradient method is a natural choice for solving the system.

8.5 Application

In this section we will briefly describe the application developed to perform the approximation experiments.

8.5.1 Overview

The aim of this program is to test out the four approximation methods. A class diagram is given in figure 8.1.

All surfaces are subclasses of the common surface interface `ControlMeshEmbedding`. A `ControlMeshEmbedding` has the ability to be evaluated in terms of control mesh parameter points and has the ability to render itself. A control mesh parameter point is a corner index plus a point in the unit square (the same parametrisation used by the simplified surface spline surface, see chapter 5.1.3). In addition, just for plain

convenience, a `ControlMeshEmbedding` can also be evaluated at a vertex, at an edge midpoint and at a face midpoint.

When rendering, the render function of `ControlMeshEmbedding` is called with a valid OpenGL context. This function is virtual, and it is thus the responsibility of the subclass to implement this. However, the `ControlMeshEmbedding` class provides some help: It allocates textures and fills them with distance information (as well as calculating statistics on this information). Thus, the subclass only has to bind the correct texture when rendering.

The `SimplifiedSurfaceSpline` class is implemented as a subclass of `ControlMeshEmbedding`. In addition, the class has an instance of a `SimplifiedSurfaceSplineBasis`, the basis in which the surface is expressed. In addition to fulfilling the `ControlMeshEmbedding` interface, the simplified surface spline has an approximation constructor. Instead of specifying the surface with control points, one can give the constructor another `ControlMeshEmbedding` and an approximation scheme. Four different approximation schemes are implemented: Lazy, interpolation, least squares and faired least squares.

The application is very simple. First, it reads a control mesh (by using the `SSS-Reader` class), and define an original from this (by using either `ControlPolyhedron` or `ControlSphere`) data. Then, a `SimplifiedSurfaceSpline` is created for each of the four scheme by passing the original and the appropriate scheme to the constructor. Finally, the rest of the program is a simple GLUT-program opening a window and setting up an OpenGL context.

8.5.2 Nitty-gritty details

We will dive into the heart of the matter; the details of the classes. We divide the classes in two: core classes and helper classes. The core classes represent the conceptual flow of the program, while the helper classes, are, as the name implies, helper classes.

Core classes

- **ControlMesh.** This class is used to hold the connectivity and blend ratios (blend ratios are only used by the simplified surface spline surface) for surfaces, and answer mesh queries. However, the mesh query paradigm is not strictly followed (the reason for this is that it was programmed prior to the development of this notation, and the awkwardness of this class interface was one of the reasons for developing this notation). It is also rather bloated, much information which would be easily deducible with a more clever data structure is explicitly stored, and therefore it has a large memory footprint. To summarise: It is not an ideal implementation, but even so, it works.
- **ControlMeshEmbedding.** This is the superclass for the surfaces parameterised by a control mesh. This class serves several purposes. Mainly, it is an abstract interface for surfaces parametrised by control meshes. The classes sub-classing this class must implement a few methods for evaluating itself and a method for OpenGL rendering. In addition, as mentioned earlier, the class implements some functionality to calculate distances between two surfaces and generating texture maps of this information.
- **SimplifiedSurfaceSplineBasis.** This class has functionality to evaluate the basis functions defined by its control mesh. The implementation follows the math-

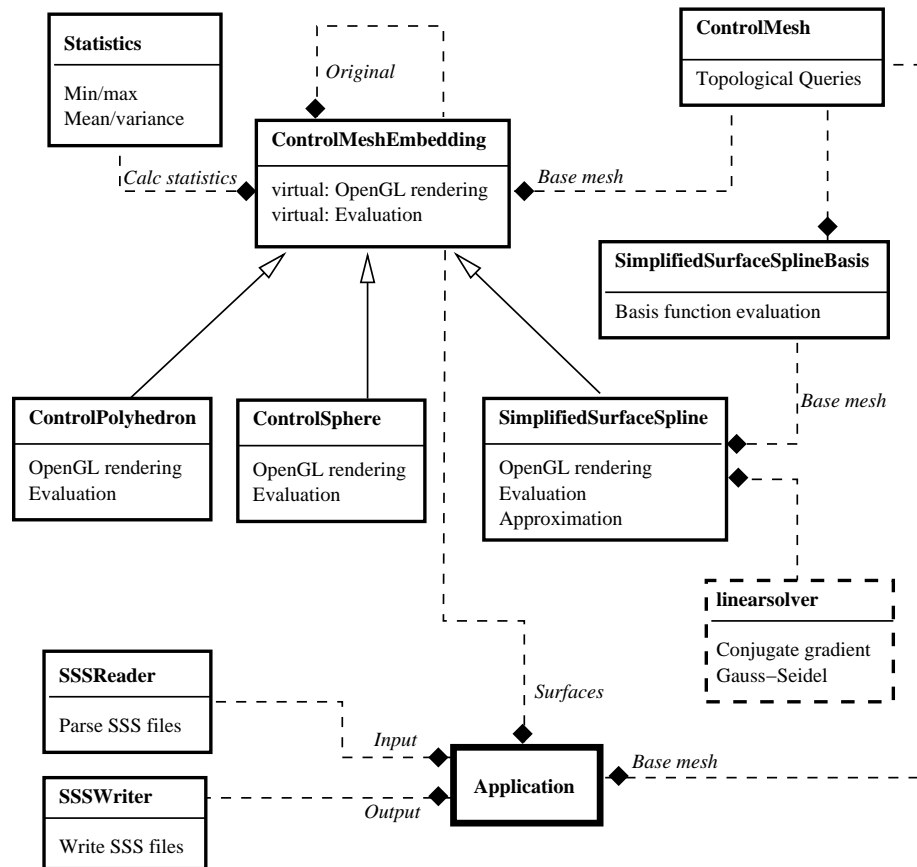


Figure 8.1:]

Class diagram of application. Arrows denote class inheritance, and dotted lines with diamond denotes a 'has-a' relationship, where the class with diamond has one or more objects of the class without the diamond.

emational definition rather closely. Thus, each evaluation results in a few calls down the layers. A speed-up could be obtained by joining all or some of the layers in the evaluation process, but this would give increased code-size and complexity.

The class has functionality to determine whether or not a parameter point is inside the support of the basis function of a vertex. This is used extensively when building the approximation matrices, and gives a significant speed-up.

- **SimplifiedSurfaceSpline.** This is the implementation of the simplified surface spline surface. It has a **ControlMesh** and a **SimplifiedSurfaceSplineBasis** defined by this **ControlMesh**. In addition, it has a set of points, one for each of the vertices in the **ControlMesh**. The set of points defines the surface in the **SimplifiedSurfaceSplineBasis**.

The approximation methods reside in the constructor. Instead of explicitly defining the surface with a set of points (which is done with the **ControlPolyhedron** and **ControlSphere**), the approximation constructor is given a **ControlMesh-Embedding** and a choice of approximation. We elaborate on the approximation methods in the section 8.4.

To evaluate the surface at a parameter point, all the basis functions with this parameter point in its support are evaluated and multiplied with the point of their associated vertex, and then added together. The result of this is the point on the surface.

- **ControlPolyhedron.** This is in essence just a plain polyhedral mesh. However, since control meshes allows n -gons, and they are not necessarily planar, some tricks must be employed. The trick here is to refine the mesh once with the face splitting scheme (similar to the surface spline), and use a bilinear interpolation over each quad. The resulting surface is flat where the control mesh facets are flat, and in addition is well-defined where the facets are not flat. A desired side-effect of this is that it is possible to parameterise the surface with the control mesh parameter points.
- **ControlSphere.** This class was implemented to test the methods on smooth convex objects. This class is almost the same as a **ControlPolyhedron**, but all evaluated points are normalised, and thus, the surface is “shrink-wrapped” to the unit sphere (see section 8.1).

Helper classes

- **Linearsolver.** This is in fact not a class, but just some C-functions to build a sparse matrix representation and solve these linear systems. Two solvers are available, the Gauss-Seidel method, useful for asymmetric matrices, and the conjugate gradient method, useful for symmetric and positive definite matrices. These are straight-forward implementations of the pseudo-code given in (Cheney & Kincaid 1996).
- **Statistics.** This is a simple class to calculate the mean, variance, minimum and maximum of a series of numbers.
- **SSSReader.** Class reading the SSS-file format, a simple ASCII-based file format for simplified surface splines.

- SSSWriter. The class for writing SSS-files.

The approximation schemes

All of the four approximation methods are built as a linear problem represented as a matrix equation. In the interpolation scheme, the A matrix is built, while in the least squares-based schemes $A^T A$ is built directly. The systems are solved separately for X , Y and Z . Most entries of the matrices are zeros, and advantage of this is taken by examining the support of each basis function.

Most of the computation time is spent building the matrices. The evaluation of the simplified surface spline basis is rather expensive since all layers are implemented explicitly, and not in a composite fashion. This results in the overhead of several function calls per evaluation.

There are two major strategies for evaluation of the basis functions (see section 7.1). The easiest is to run the full scheme for each vertex, setting the value of the current vertex to one, and the rest to zero. In this way, the resulting surface is the basis function for that particular vertex. The other approach is to find expressions for the basis functions analytically, and then use these functions. This implementation uses the latter approach.

8.6 Results

As a practical test for the simplified surface spline, we applied the four approximation methods on several models. Images of the results can be found in figures 8.2 to 8.13. In the upper left corner, there is an image of the approximation subject. In the upper right corner the errors of the different methods are depicted. The black column represents the mean value of the errors and the grey block is the maximum error for the method (see section 8.3). In the middle and bottom rows the output of the four approximation methods are depicted. The surface is coloured according to their local error. White is zero error, and black is maximum error.

Data on the approximation experiments can be found in tables 8.2 to 8.5. In this tables E_i^μ is the mean error of model i , E_i^∞ is the maximum error, $\frac{E_{i-1}}{E_i}$ is the ratio between the error of the refined and unrefined version of a model. The number of iterations used is in column “#it”, and the condition number of the matrix used in the linear systems based on the 2norm and infinity norm is in column C^2 and C^∞ respectively.

From the experiments, we see that the *lazy approximation* consistently gives an approximation smaller than the original. Because of the convex hull property, this comes as no surprise, and is particularly visible in the sphere2 approximation. The lazy approximation has the largest maximum error for all models except the bunny model, where the faired approximation has a slightly larger error. The mean error of the lazy approximations are small for cube models, but large for sphere models. The local convex hull property can be used to interpret this: When a local region is flat, the local convex hull of that region is thin and the approximation resides inside this volume. In curved areas, the local convex hull grows fatter, and the deviation increases. However, the lazy approximation gives smooth and well-shaped control meshes.

Except for the sphere2 case, the *vertex interpolation* has larger average error than the other methods. Another drawback of the vertex interpolation is the oscillations, and thus, non-smooth and badly shaped control meshes. This is particularly obvious

in the bunny model. In addition, at sharp corners, the approximation has some major artefacts that are not present in the original: Instead of smoothing out corners like the lazy approximation, the vertex interpolation creates an artificial wobble.

The *least squares approximation* remedies the artefacts of vertex interpolation to some extent. Oscillations are still present, but on a smaller scale. However, the control mesh is not particularly well formed. The average error, as well as the maximum error, is in general smaller compared with the other methods.

When measuring average and maximum error, the *faired approximation* has the worst approximation performance on average. However, it does not make the models too small (which lazy approximation did on the sphere2 model), and the oscillations are moderate. In fact, the control mesh of this scheme is smoother than the control meshes from the lazy approximation.

All of the approximations were of rather good condition. The systems solved by the conjugate gradient method behaved very well, and a good solution was found after few iterations.

As a conclusion, the least squares method had the best approximation properties, but at the cost of an oscillating surface. The fair approximation had moderate approximation properties for all cases, but suffered little of the problems of the other schemes (oscillations and making the model too small). Thus, a least squares method with some fairing criteria is probably a suitable approximation scheme for the simplified surface spline.

Introducing blend ratio estimation and better parameterisation could on the other hand give completely different results. Moreover, as a critique to the experiment is the fact that the models were specially crafted to expose weaknesses in the approximation methods. Thus, real-life situations could give other results.

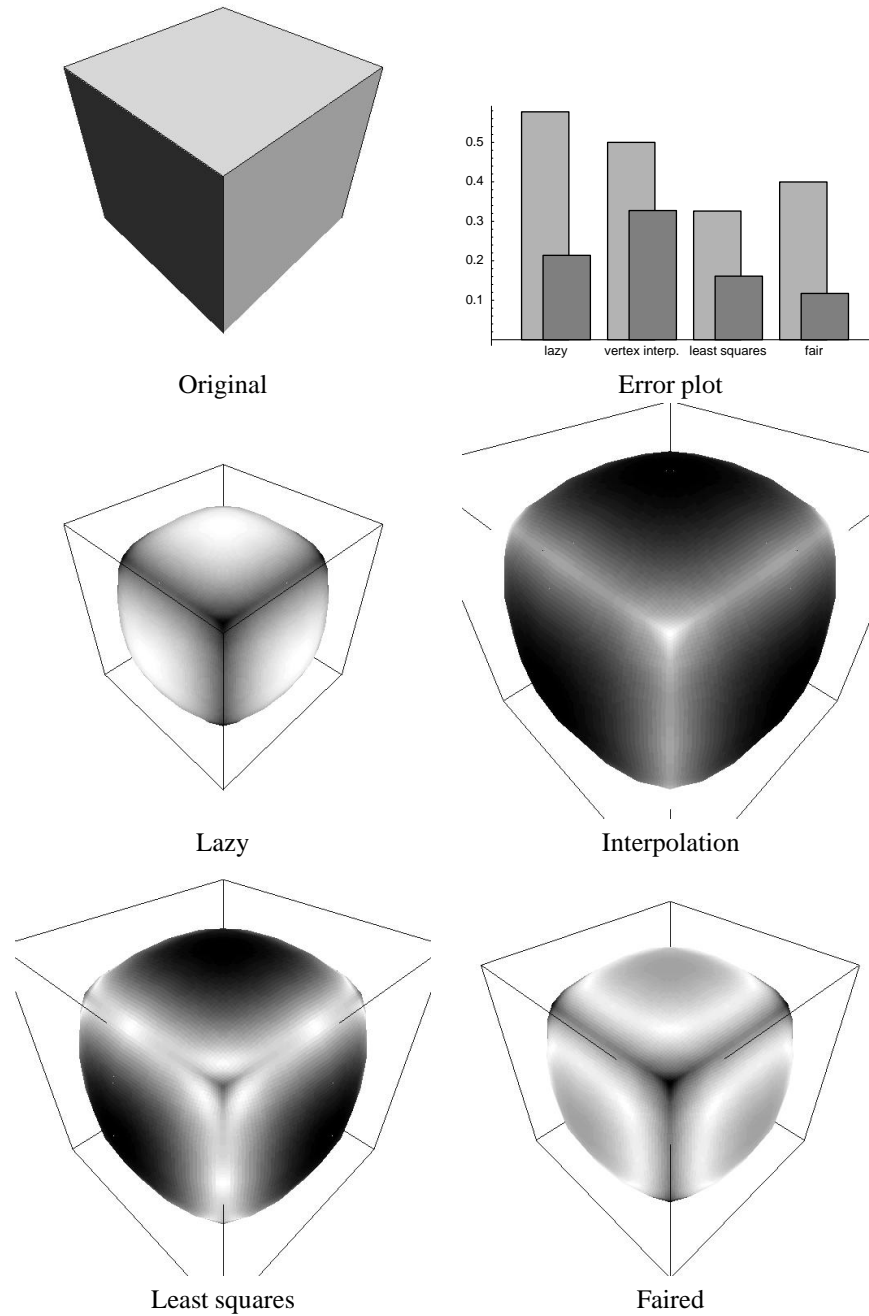


Figure 8.2: Original: "cube0", $N_v = 8$, $N_e = 12$, $N_c = 24$ and $N_f = 6$

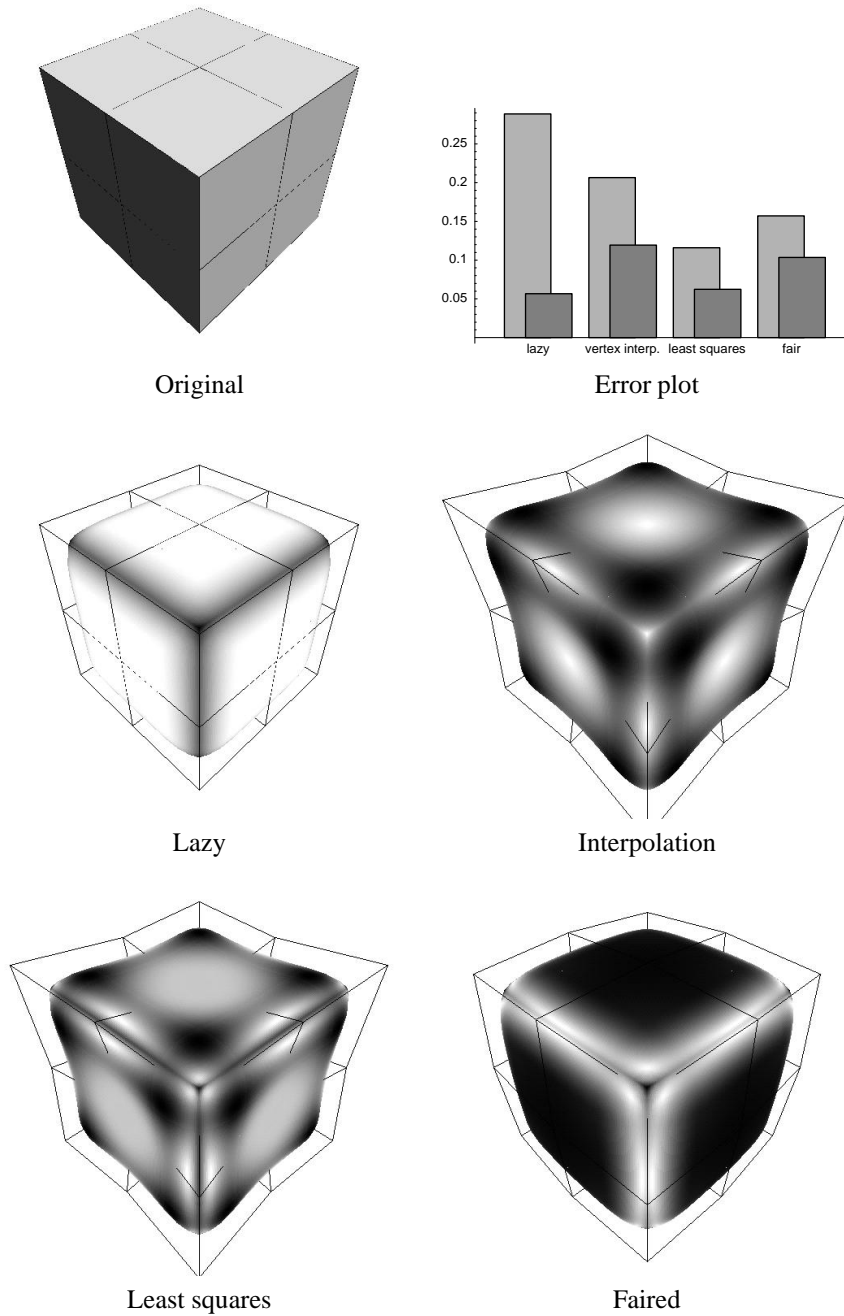
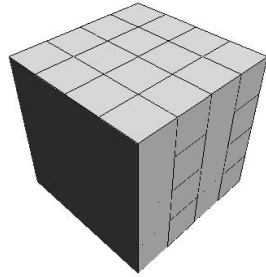
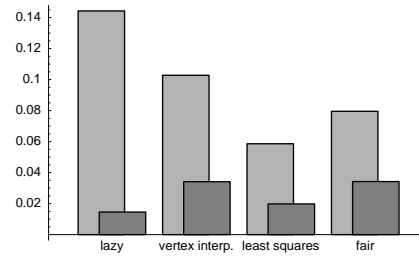


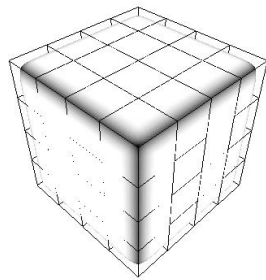
Figure 8.3: Original: “cube1”, $N_v = 26$, $N_e = 48$, $N_c = 96$ and $N_f = 24$



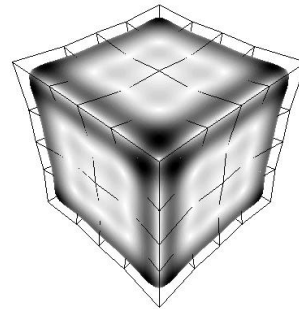
Original



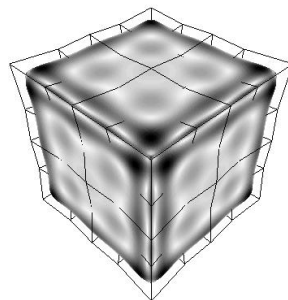
Error plot



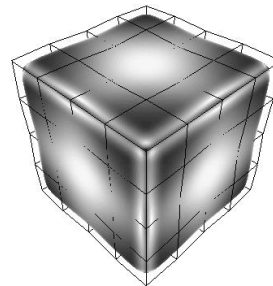
Lazy



Interpolation

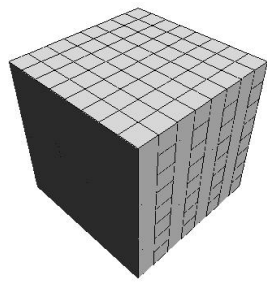


Least squares

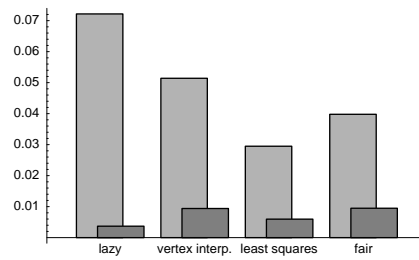


Faired

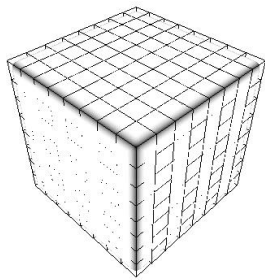
Figure 8.4: Original: "cube2", $N_v = 98$, $N_e = 192$, $N_c = 384$ and $N_f = 96$



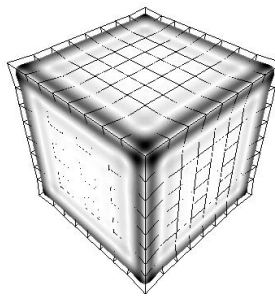
Original



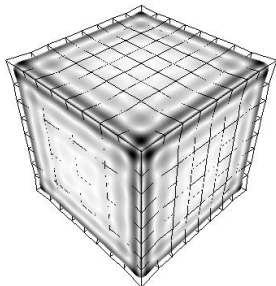
Error plot



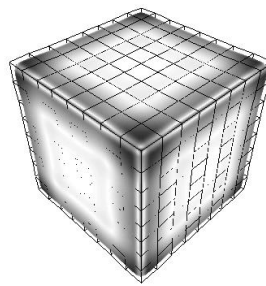
Lazy



Interpolation

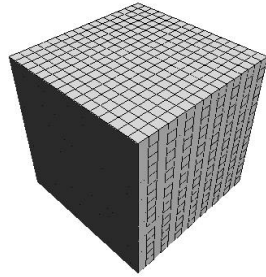


Least squares

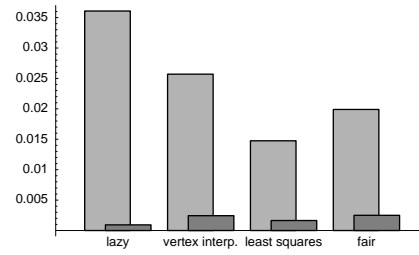


Faired

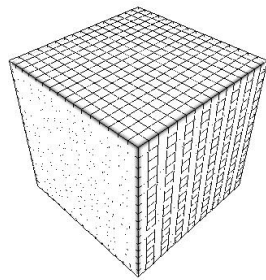
Figure 8.5: Original: "cube3", $N_v = 386$, $N_e = 768$, $N_c = 1536$ and $N_f = 384$



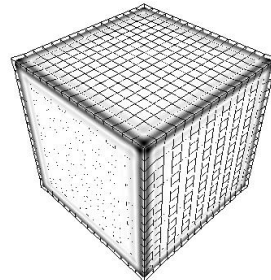
Original



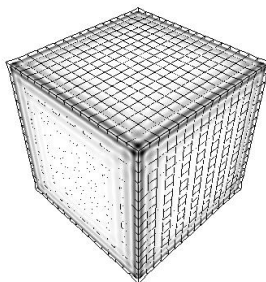
Error plot



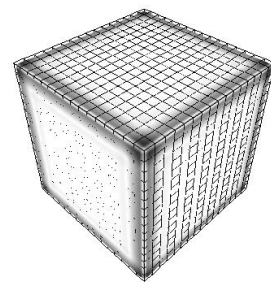
Lazy



Interpolation



Least squares



Faired

Figure 8.6: Original: "cube4", $N_v = 1538$, $N_e = 3072$, $N_c = 6144$ and $N_f = 1536$

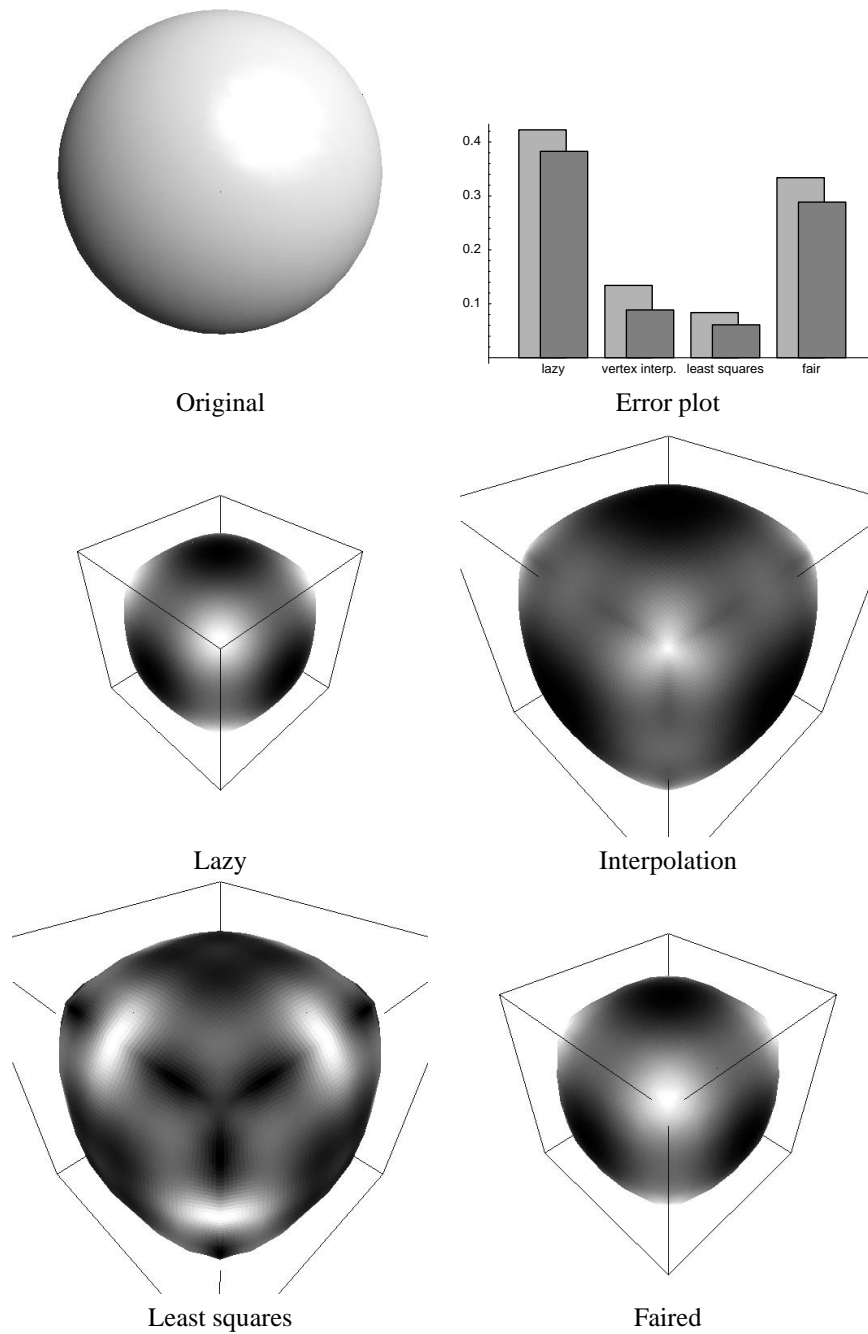


Figure 8.7: Original: “sphere0”, $N_v = 8$, $N_e = 12$, $N_c = 24$ and $N_f = 6$

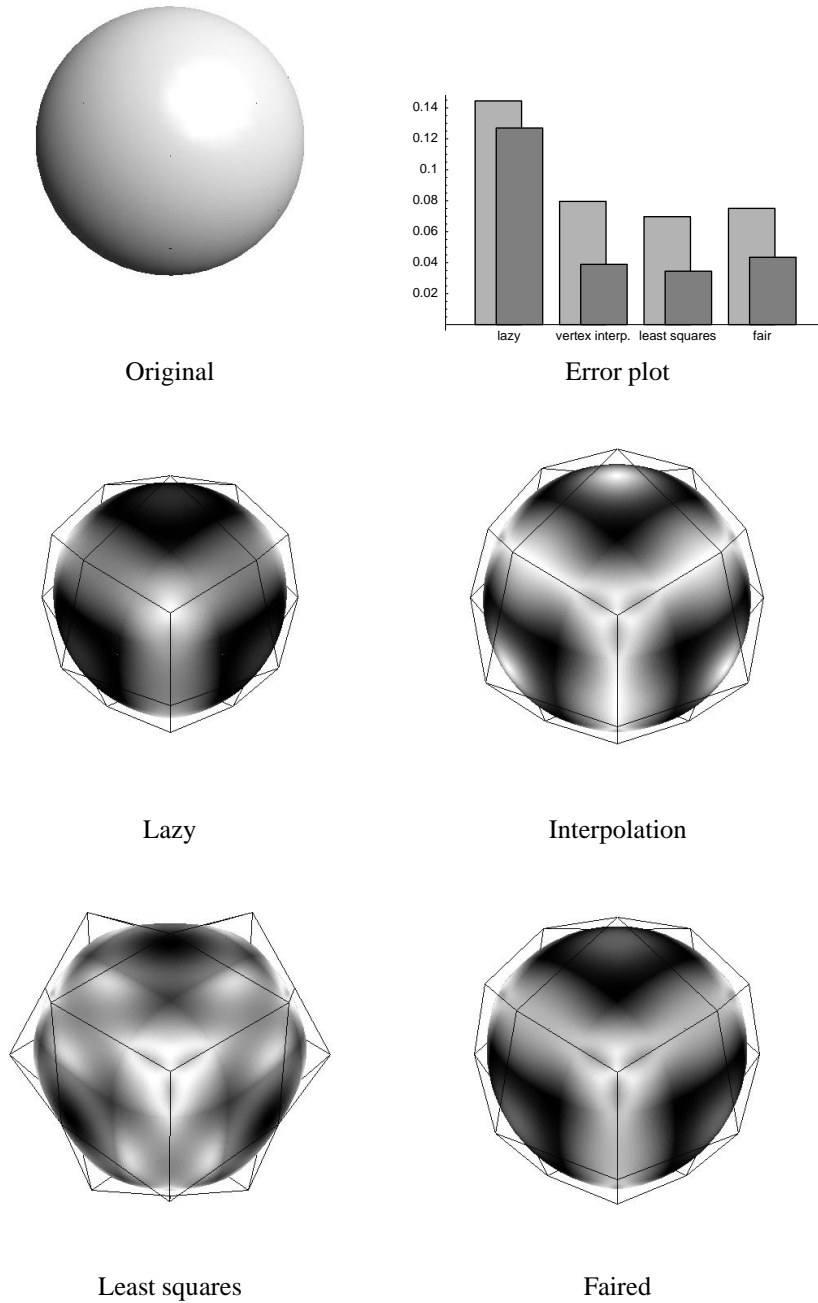
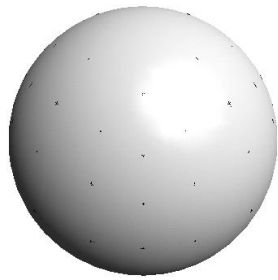
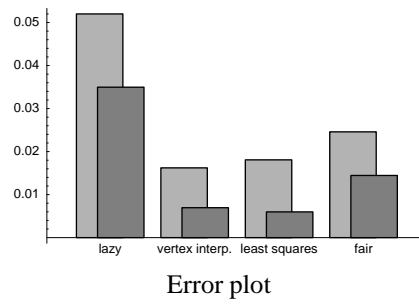


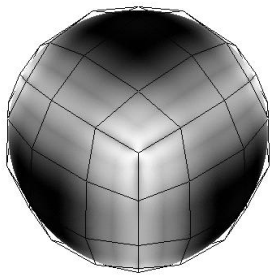
Figure 8.8: Original: “sphere1”, $N_v = 26$, $N_e = 48$, $N_c = 96$ and $N_f = 24$



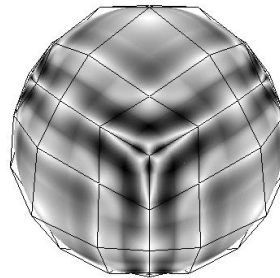
Original



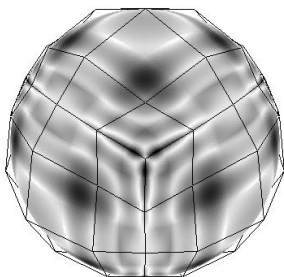
Error plot



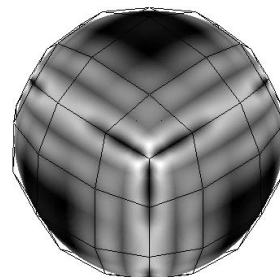
Lazy



Interpolation

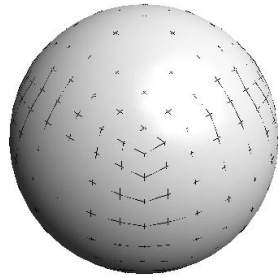


Least squares

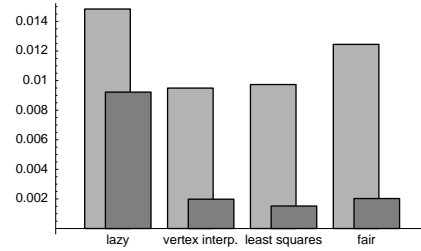


Faired

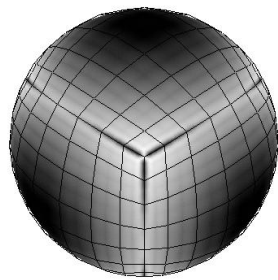
Figure 8.9: Original: "sphere2", $N_v = 98$, $N_e = 192$, $N_c = 384$ and $N_f = 96$



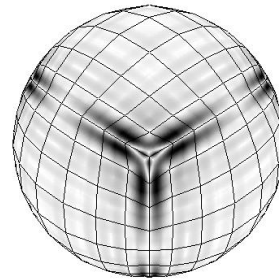
Original



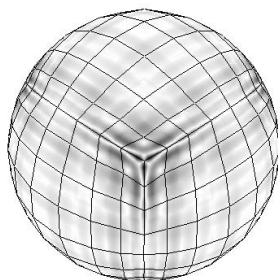
Error plot



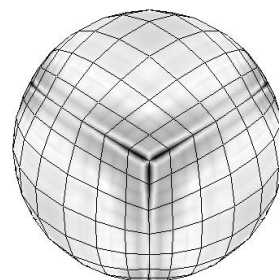
Lazy



Interpolation

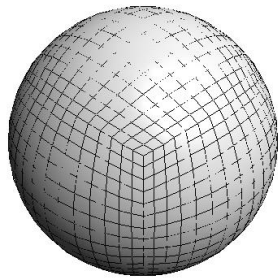


Least squares

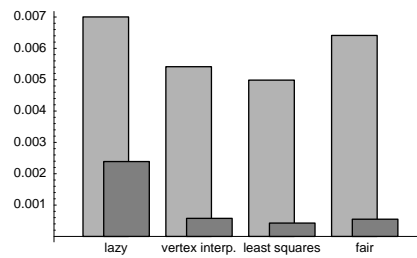


Faired

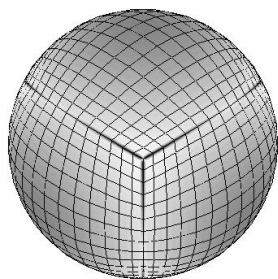
Figure 8.10: Original: "sphere3", $N_v = 386$, $N_e = 768$, $N_c = 1536$ and $N_f = 384$



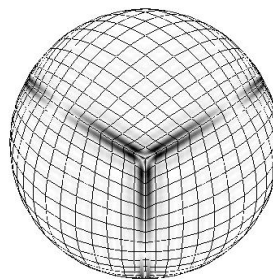
Original



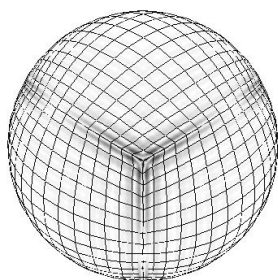
Error plot



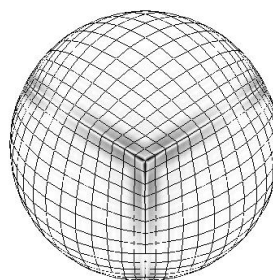
Lazy



Interpolation



Least squares



Faired

Figure 8.11: Original: "sphere4", $N_v = 386$, $N_e = 768$, $N_c = 1536$ and $N_f = 384$

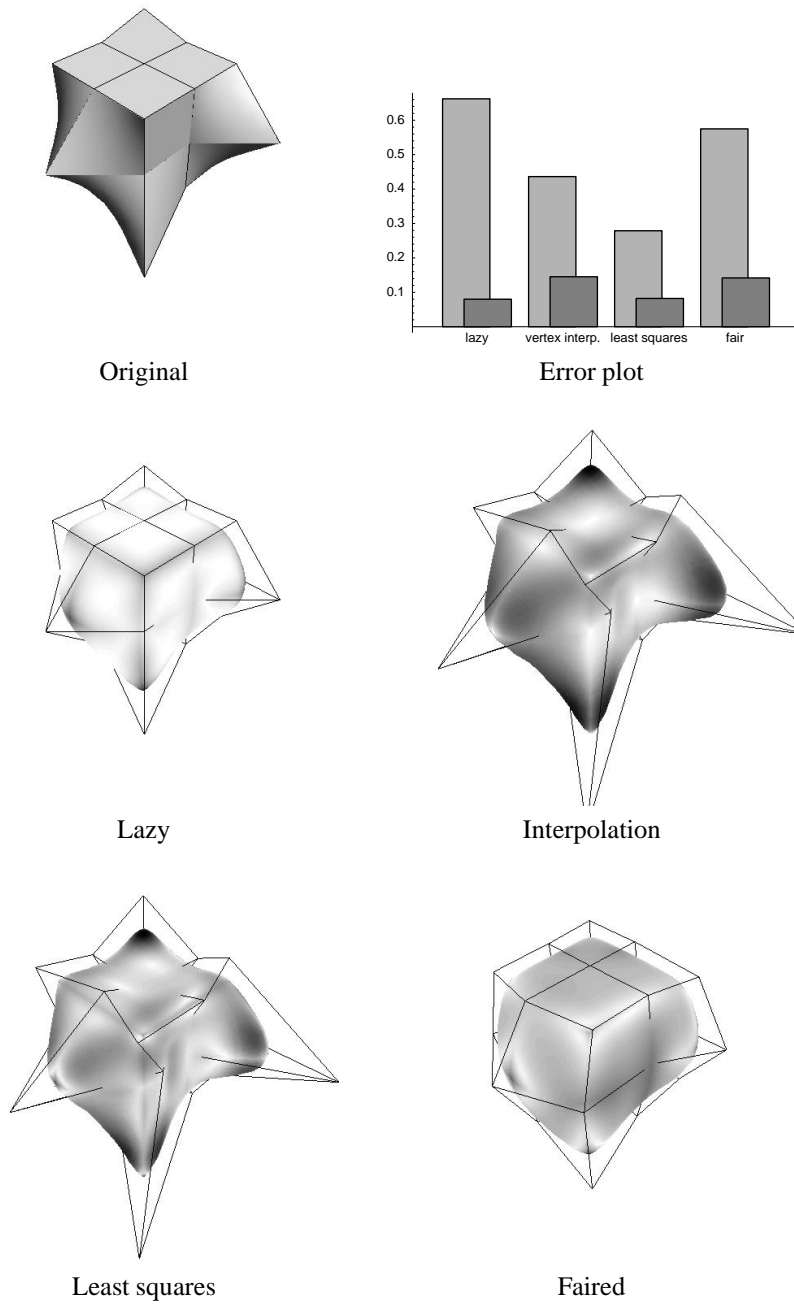


Figure 8.12: Original: "jittercube", $N_v = 26$, $N_e = 48$, $N_c = 96$ and $N_f = 24$

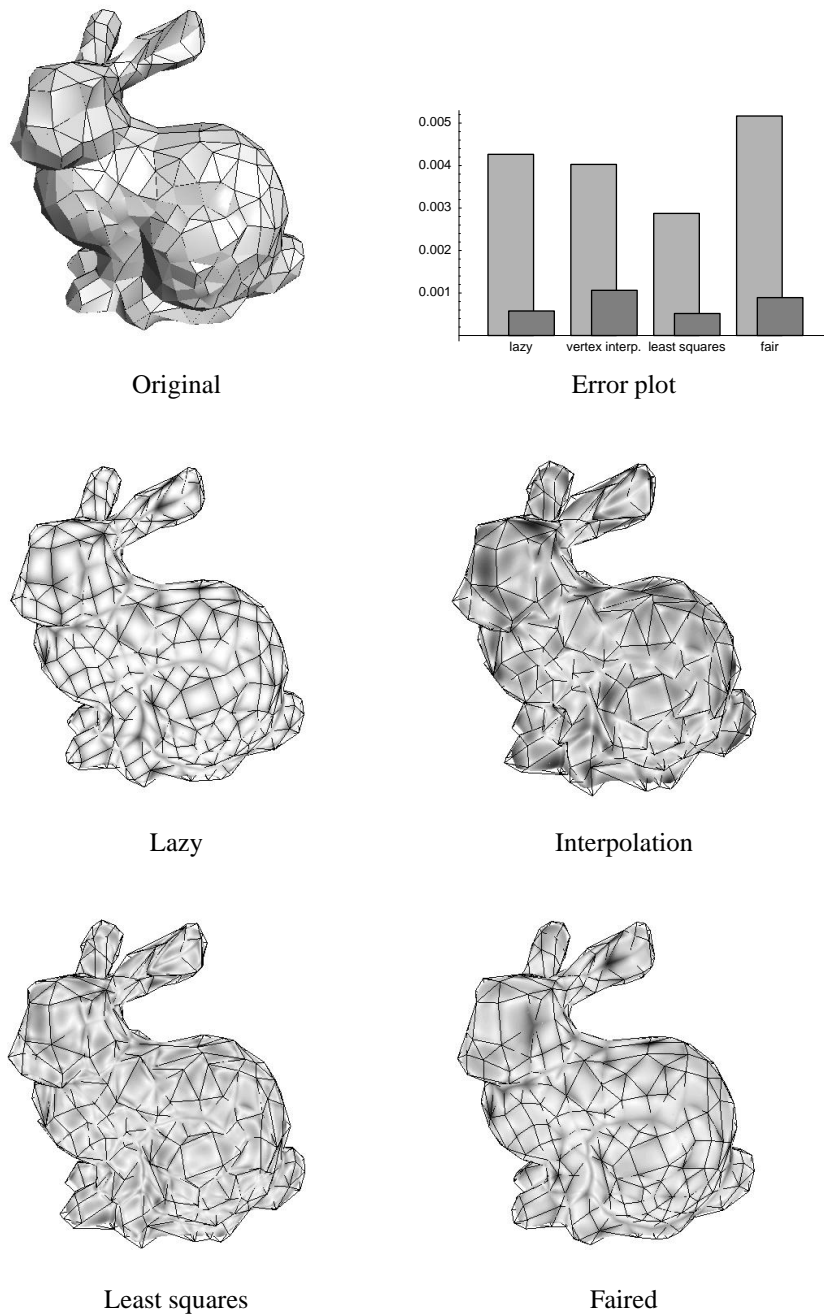


Figure 8.13: Original: "bunny", $N_v = 473$, $N_e = 1194$, $N_c = 2388$ and $N_f = 723$

model	E_i^μ	$\frac{E_{i-1}^\mu}{E_i^\mu}$	E_i^∞	$\frac{E_{i-1}^\infty}{E_i^\infty}$	#it	C^2	C^∞
cube0	$2.138 \cdot 10^{-1}$	—	$5.774 \cdot 10^{-1}$	—	—	—	—
cube1	$5.666 \cdot 10^{-2}$	3.773	$2.887 \cdot 10^{-1}$	2.000	—	—	—
cube2	$1.457 \cdot 10^{-2}$	3.890	$1.443 \cdot 10^{-1}$	2.000	—	—	—
cube3	$3.693 \cdot 10^{-3}$	3.944	$7.217 \cdot 10^{-2}$	2.000	—	—	—
cube4	$9.301 \cdot 10^{-4}$	3.971	$3.608 \cdot 10^{-2}$	2.000	—	—	—
sphere0	$3.827 \cdot 10^{-1}$	—	$4.226 \cdot 10^{-1}$	—	—	—	—
sphere1	$1.270 \cdot 10^{-1}$	3.013	$1.445 \cdot 10^{-1}$	2.925	—	—	—
sphere2	$3.497 \cdot 10^{-2}$	3.632	$5.200 \cdot 10^{-2}$	2.779	—	—	—
sphere3	$9.223 \cdot 10^{-3}$	3.791	$1.484 \cdot 10^{-2}$	3.503	—	—	—
sphere4	$2.389 \cdot 10^{-3}$	3.861	$7.003 \cdot 10^{-3}$	2.120	—	—	—
jittercube	$8.001 \cdot 10^{-2}$	—	$6.622 \cdot 10^{-1}$	—	—	—	—
bunny	$5.775 \cdot 10^{-4}$	—	$4.264 \cdot 10^{-3}$	—	—	—	—

Table 8.2: Results from lazy approximation

model	E_i^μ	$\frac{E_{i-1}^\mu}{E_i^\mu}$	E_i^∞	$\frac{E_{i-1}^\infty}{E_i^\infty}$	#it	C^2	C^∞
cube0	$3.273 \cdot 10^{-1}$	—	$5.000 \cdot 10^{-1}$	—	100	4.000	4.000
cube1	$1.194 \cdot 10^{-1}$	2.742	$2.065 \cdot 10^{-1}$	2.421	100	4.026	4.000
cube2	$3.414 \cdot 10^{-2}$	3.497	$1.028 \cdot 10^{-1}$	2.009	100	4.007	4.000
cube3	$9.398 \cdot 10^{-3}$	3.633	$5.141 \cdot 10^{-2}$	2.000	100	4.002	4.000
cube4	$2.439 \cdot 10^{-3}$	3.853	$2.570 \cdot 10^{-2}$	2.000	100	4.000	4.000
sphere0	$8.861 \cdot 10^{-2}$	—	$1.341 \cdot 10^{-1}$	—	100	4.000	4.000
sphere1	$3.894 \cdot 10^{-2}$	2.276	$7.959 \cdot 10^{-2}$	1.685	100	4.026	4.000
sphere2	$6.956 \cdot 10^{-3}$	5.598	$1.622 \cdot 10^{-2}$	4.907	100	4.007	4.000
sphere3	$1.983 \cdot 10^{-3}$	3.508	$9.494 \cdot 10^{-3}$	1.708	100	4.002	4.000
sphere4	$5.765 \cdot 10^{-4}$	3.439	$5.415 \cdot 10^{-3}$	1.753	100	4.000	4.000
jittercube	$1.452 \cdot 10^{-1}$	—	$4.362 \cdot 10^{-1}$	—	100	4.026	4.000
bunny	$1.064 \cdot 10^{-3}$	—	$4.027 \cdot 10^{-3}$	—	100	3.087	3.840

Table 8.3: Results from vertex interpolation

model	E_i^μ	$\frac{E_{i-1}^\mu}{E_i^\mu}$	E_i^∞	$\frac{E_{i-1}^\infty}{E_i^\infty}$	#it	C^2	C^∞
cube0	$1.611 \cdot 10^{-1}$	—	$3.260 \cdot 10^{-1}$	—	3	$5.200 \cdot 10$	$5.200 \cdot 10$
cube1	$6.235 \cdot 10^{-2}$	2.584	$1.160 \cdot 10^{-1}$	2.809	4	$6.128 \cdot 10$	$6.625 \cdot 10$
cube2	$1.981 \cdot 10^{-2}$	3.148	$5.863 \cdot 10^{-2}$	1.979	12	$6.336 \cdot 10$	$6.481 \cdot 10$
cube3	$5.964 \cdot 10^{-3}$	3.321	$2.949 \cdot 10^{-2}$	1.988	37	$6.385 \cdot 10$	$6.469 \cdot 10$
cube4	$1.643 \cdot 10^{-3}$	3.631	$1.475 \cdot 10^{-2}$	1.999	60	$6.397 \cdot 10$	$6.469 \cdot 10$
sphere0	$6.103 \cdot 10^{-2}$	—	$8.369 \cdot 10^{-2}$	—	4	$5.200 \cdot 10$	$5.200 \cdot 10$
sphere1	$3.451 \cdot 10^{-2}$	1.768	$6.973 \cdot 10^{-2}$	1.200	9	$6.128 \cdot 10$	$6.625 \cdot 10$
sphere2	$5.983 \cdot 10^{-3}$	5.769	$1.808 \cdot 10^{-2}$	3.856	15	$6.336 \cdot 10$	$6.481 \cdot 10$
sphere3	$1.521 \cdot 10^{-3}$	3.933	$9.734 \cdot 10^{-3}$	1.858	39	$6.385 \cdot 10$	$6.469 \cdot 10$
sphere4	$4.263 \cdot 10^{-4}$	3.569	$4.987 \cdot 10^{-3}$	1.952	59	$6.397 \cdot 10$	$6.469 \cdot 10$
jittercube	$8.221 \cdot 10^{-2}$	—	$2.789 \cdot 10^{-1}$	—	14	$6.128 \cdot 10$	$6.625 \cdot 10$
bunny	$5.183 \cdot 10^{-4}$	—	$2.874 \cdot 10^{-3}$	—	42	$2.661 \cdot 10$	$7.224 \cdot 10$

Table 8.4: Results from least squares approximation

model	E_i^μ	$\frac{E_{i-1}^\mu}{E_i^\mu}$	E_i^∞	$\frac{E_{i-1}^\infty}{E_i^\infty}$	#it	C^2	C^∞
cube0	$1.173 \cdot 10^{-1}$	—	$3.997 \cdot 10^{-1}$	—	2	2.185	2.846
cube1	$1.035 \cdot 10^{-1}$	1.133	$1.571 \cdot 10^{-1}$	2.544	2	2.341	3.438
cube2	$3.423 \cdot 10^{-2}$	3.023	$7.956 \cdot 10^{-2}$	1.975	7	2.312	3.477
cube3	$9.484 \cdot 10^{-3}$	3.610	$3.979 \cdot 10^{-2}$	1.999	11	2.312	2.312
cube4	$2.499 \cdot 10^{-3}$	3.796	$1.990 \cdot 10^{-2}$	2.000	12	2.314	3.503
sphere0	$2.886 \cdot 10^{-1}$	—	$3.338 \cdot 10^{-1}$	—	2	2.185	2.846
sphere1	$4.350 \cdot 10^{-2}$	6.634	$7.509 \cdot 10^{-2}$	4.446	2	2.341	3.438
sphere2	$1.445 \cdot 10^{-2}$	3.010	$2.459 \cdot 10^{-2}$	3.053	7	2.312	3.477
sphere3	$2.027 \cdot 10^{-3}$	7.130	$1.245 \cdot 10^{-2}$	1.975	10	2.312	2.312
sphere4	$5.497 \cdot 10^{-4}$	3.688	$6.412 \cdot 10^{-3}$	1.942	11	2.314	3.503
jittercube	$1.418 \cdot 10^{-1}$	—	$5.749 \cdot 10^{-1}$	—	9	2.341	3.438
bunny	$8.919 \cdot 10^{-4}$	—	$5.166 \cdot 10^{-3}$	—	9	1.779	3.590

Table 8.5: Results from faired approximation

Chapter 9

Conclusions and further work

The aim of this thesis is to explore the approximation properties of the surface spline schemes. We will in this chapter try to state some conclusions as well as suggesting topics for further research.

To explore the approximation properties of the surface spline schemes, we have developed the simplified version of the C^1 -surface spline, the simplified surface spline. This scheme has explicit formulas for both surface evaluation, as well as basis functions, in addition to a known dimension of the space. Thus it should be quite a good basis for approximation purposes.

However, the results were not particularly impressive. There can be several reasons for this. First of all, the conditioning of the basis is not known. The upper bound in the condition inequality is one (since the scheme is composed of convex combinations), but the lower bound is unknown. Obtaining this lower bound can reveal whether or not this basis is stable. However, this can be tested numerically, and the condition numbers from the approximation experiments indicate stability.

Further, an approximation of a model where large and small faces intermixed would be interesting, and could reveal more of the stability properties of the basis. In addition, it would be interesting to run the tests with better parameterisations as well as a more sensible utilisation of the blend ratios. It is probably easy to create some heuristics to determine suitable blend ratios. Another interesting approximation project would be to check if the condition number of the approximation problem is somewhat linked to the degree of vertices.

Extending the scheme to the full C^1 -surface spline scheme will probably not improve the condition, since an increased number of operations are performed as well as the planarisation-projections (which could have an effect on the dimension of the basis as well) that transmogrify the mesh into a planar cut polyhedron. But, on the other hand, it is possible that the full scheme will deal with irregular corners in a better way.

The importance of parameterisation cannot be emphasised too much. It is common knowledge that oscillations often occur where excess parameter space is present. In addition, the speed of the parameterisation is of high importance; sudden changes in parameter speed can give strange results. Hence, an disproportion between geometric size of the surface and the size of the corresponding parameter space is therefore indeed a peril. The parameterisation used in the approximation experiments were particularly simple, and thus a better parameterisation could improve the results.

In addition, the power of the blend ratios were not utilised either. The blend ratios

have the ability to shape the parameter space, and thus make sharp corners and creases more “natural” for the surface spline.

With this said, we have found a useful method of dealing with polygonal meshes (and indeed surface splines), the mesh notation. In our opinion, the formulas are nice, and hopefully equally understandable. It is our hope that the framework as well as the results of this thesis can serve as an inspiration for further exploration of the approximation properties of the surface spline schemes.

Finding some bound on the distance between a C^1 -surface spline surface and a simplified surface spline surface of the same coefficients would be interesting. If this is small, the simplified surface spline basis would be a suitable approximation basis for obtaining coefficients to be used with the full scheme.

The surface splines share a fundamental problem with subdivision surfaces, and this is how to find the control mesh (base mesh). Current decimation algorithms are well suited for coarse polygonal representations. Polygons are inherently flat, but surface splines are curved, and thus, most standard decimation strategies are not necessarily optimal for finding a control mesh. Decimation with a more suitable criteria based on some intrinsic properties of the surface spline like “flatness” or curvature could be interesting.

Bibliography

- Arvo, J., Cook, R. L., Glassner, A., Hanrahan, P., Heckbert, P. S. & Kirk, D. (1989), *An Introduction to Ray Tracing*, Morgan Kaufmann Publishers, Inc.
- Catmull, E. & Clark, J. (1978), 'Recursively generated B-spline surfaces on arbitrary topological meshes', *Computer-aided Design* **10**(6), 350–355.
- Cheney, W. & Kincaid, D. (1996), *Numerical Analysis*, 2. edn, Brooks/Cole Publishing Company.
- Doo, D. & Sabin, M. (1978), 'Behaviour of recursive subdivision surfaces near extraordinary points', *Computer-aided Design* **10**(6), 356–360.
- Eck, M. & Hoppe, H. (1996), Automatic reconstruction of b-spline surfaces of arbitrary topological type, in 'SIGGRAPH 96 Conference Proceedings', Annual Conference Series, SIGGRAPH, ACM Press, pp. 325–334.
- Elmasri, R. A. & Navathe, S. B. (1994), *Fundamentals of database systems*, 2nd edn, The Benjamin/Cummings Publishing Company, Inc.
- Farin, G. (1996), *Curves and Surfaces for CAGD, a practical guide*, 4. edn, Academic press.
- Foley, J. D., van Dam, A., Feiner, S. K. & Hughes, J. F. (1996), *Computer Graphics: Principle and practice*, The systems programming series, 2. edn, Addison-Wesley Publishing Company. in C.
- Gallier, J. (2000), *Curves and Surfaces in Geometric Modeling, theory and algorithms*, Morgan Kaufmann Publishers, Inc.
- Gonzalez, C. & Peters, J. (1999), Localized hierarchy surface splines, in S. S. J. Rossignac, ed., 'ACM Symposium on Interactive 3D Graphics', pp. 7–15.
- Gonzalez-Ochoa, C., McCammon, S. & Peters, J. (1998), 'Computing moments of objects enclosed by piecewise polynomial surfaces', *ACM Transactions on Graphics* **17**(3), 143–157.
- Gribb, G. (1999), 'Re: Getting the modelview matrix — how bad?', OpenGL Gamedeveloper Mailinglist. http://apollo.iwt.uni-bielefeld.de/~ml_robot/OpenGL-12-1999/0347.html.
- Halbwachs, Y., Courriouz, G., Renaud, X. & Repousseau, P. (1996), 'Topological and geometric characterization of fault networks using 3-dimensional generalized maps', *Mathematical Geology* **28**(5), 625–656.

- Halbwachs, Y. & Hjelle, Ø. (1999), Generalized maps in geological modeling: Object oriented design of topological kernel. To appear in the book *Advances in Software Tools for Scientific Computing* edited by H.P. Langtangen and E. Quak.
- Mørken, K. & Lyche, T. (2002), Spline methods. IN-329 Lecture notes.
- MSDN (n.d.), 'Direct3D architecture', Microsoft Developer Network website. <http://msdn.microsoft.com/library/>.
- OpenGL ARB, Woo, M., Neider, J., Davis, T. & Schreiner, D. (1999), *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*, 3rd edn, Addison-Wesley Publishing Company.
- OpenGL Extension Registry* (n.d.), SGI web site. <http://oss.sgi.com/projects/ogl-sample/registry/>.
- Peters, J. (1994), Surfaces of arbitrary topology constructed from biquadratics and bicubics, in N. S. Sapidis, ed., 'Designing fair curves and surfaces: shape quality in geometric modelling and computer aided design', Society for Industrial and Applied Mathematics, pp. 277–293.
- Peters, J. (1995a), ' C^1 -surface splines', *SIAM Journal on Numerical Analysis* **32**(2), 645–666.
- Peters, J. (1995b), 'Smoothing polyhedra made easy', *ACM Transactions on Graphics* **14**(2), 161–169.
- Peters, J. (1996), 'Curvature continuous spline surfaces over irregular meshes', *Computer-Aided Geometric Design* **13**(2), 101–131.
- Peters, J. (1998), 'Pcp2nurb — smooth free-form surfacing with linearly trimmed bicubic b-splines', *ACM Transactions on Mathematical Software* **24**(3), 261–267.
- Peters, J. (2001), *Handbook of Computer Aided Design*, Springer-Verlag, chapter Geometric Continuity. In press.
- Stollnitz, E. J., DeRose, T. D. & Salesin, D. H. (1996), *Wavelets for Computer Graphics, theory and applications*, Morgan Kaufmann Publishers.
- Trefethen, L. N. & Bau, D. I. (1997), *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics.

Index

- B_{bc} , 98, 99
- C^1 surface spline, 87
- C^1 -surface spline, 55
- $R_1 \times R_2$, 18
- \mathfrak{M} , 12
- \hat{f} , 14
- $\pi(R)$, 18
- $\sigma(R)$, 18
- \mathbb{C} , 16
- \mathbb{D} , 16
- $\mathbb{E}\mathbb{E}$, 16
- $\mathbb{H}\mathbb{E}$, 16
- B_V , 98, 100
- B_{EM} , 99
- B_{FM} , 99
- F_V , 98, 100
- F_{EM} , 99
- F_{FM} , 99
- G , 99, 100
- e_j , 12
- e_{in} , 14
- e_{out} , 14
- f_k , 12
- g , 99
- v_i , 12
- 2-norm, 98, 99

- Animation, 87
- Approximation, 2, 95, 103, 104
 - faired least squares, 97
 - lazy, 97
 - least squares, 97
 - variation diminishing, 97
 - vertex interpolation, 97, 98
- Arbitrary topology, 2
- Attribute, *see* relational algebra

- B-spline, 1, 45, 56, 58, 87
 - knot vector, 1
- B-splines, 97
- Bézier
 - coefficient, 98
- Bézier curve
 - control points, 6
 - definition of, 6
- Bézier curve segment, 49
- Bézier patch, 45
- Bézier surfaces
 - tensor product construction, 7
- Basis
 - edge midpoint function, 99
 - face midpoint function, 99
 - function, 98
 - support, 99
 - vertex basis function, 98
- Basis function, 2, 79–81, 86, 101
 - evaluation strategy, 104
 - Linearly independent, 81
 - linearly independent, 80, 86
 - support, 103, 104
- Bernstein polynomial
 - definition of, 6
- Bilinear interpolation, 103
- Blend ratio, 1, 31–33, 45, 47, 58, 60, 96
 - criterion, 86
- Boundary curve, 31
 - of patch, 50
- Boundary edges, *see* mesh

- Cartesian product, *see* relational algebra
- Circuit, *see* set
- Clough-Tocher, *see* macro splines
- Code
 - complexity, 101
 - size, 101
- Coefficient, 97
- Computation time, 104
- Conjugate gradient solver, 99, 100
- Connectivity, 1, 12
 - kernel, 2

- subset, 2
- Constraints
 - fair, 99, 100
- Constructor, 103
- Continuity, 55
 - at a joint, 8
- Control Mesh, 100, 101
- Control mesh, 47, 48, 55–58, *see* mesh,
 - control
 - embedding, 95
- Control point, 31–33
- ControlMesh (class), 101, 103
- ControlMeshEmbedding (class), 100, 101, 103
- ControlPolyhedron (class), 101, 103
- ControlSphere (class), 96, 101, 103
- Convex combination, 1, 4, 33, 83, 98
 - definition of, 4
 - matrix form, 5
 - of a convex combination, 4
- Convex hull, 4, 92
- Convex set, 4
- Core class, 101
- Corner, *see* mesh
- Dart, *see* mesh
- Data structure, 101
- Decimation, 96
- Degree raising, 47
- Direct3D, 89
- Domain, *see* relational algebra
- Doo-Sabin split, 52
- Doo-Sabin subdivision, *see* Mesh, subdivision scheme
- Dual mesh, *see* Mesh, dual, *see* mesh
- Edge, *see* mesh
 - cutting, 47
 - midpoint, 48, 56
 - quartile, 56
 - end, *see* mesh
- Element, *see* set
- Embedded mesh, *see* mesh
- Evaluation
 - direct, 87, 88
 - in hardware, 89
 - indirect, 87, 88
 - retained, 88
- Face, *see* mesh
 - refinement, 57
- Face midpoint, 48, 56
- Face split, *see* Mesh, subdivision scheme
- Fair
 - constraints, 99, 100
- Faired least squares, 101
- Foreign key, *see* relational algebra
- Front to back rendering, 92
- Function call
 - overhead, 104
- G-map, 11, 12, 16, 19
- Gauss-Seidel solver, 98
- Geometrical midpoint, 32
- Geometry engine, 89
- Geometry kernel
 - interface, 19
- GIS, 2
- Half-edge, *see* mesh
- Half-space, 91
- Helper class, 101
- Incoming edge, *see* mesh, corner
- Inner edge quartile, 57
- Inner mesh, 47, 57, *see* Mesh, inner
- Inner product, 99
- Interpolation, 101
 - bilinear, 60
 - constraint, 98
 - edge midpoint, 98
 - face midpoint, 98
 - vertex, 98
- Interval
 - midpoint, 33
- Interval midpoint, 32, 33
- Irregular vertex, 48, 50
- Knot spacing, 47, 58
- Knot vector, 45
- Layer based schemes, 87
- Lazy approximation, 101
- Least squares, 101
 - faired, 101
- LeSS, 52
- Linear system, 98
 - building, 103
 - composite, 99
 - conjugate gradient solver, 99, 100, 103

- fair, 99
- Gauss-Seidel solver, 98, 103
- least squares, 98, 99
- minimisation, 98
- norm, 98
- over determined, 98, 99
- positive definite systems, 103
- rank, 99
- residual, 99
- Linearsolver (class), 103
- Local construction, 45
- Local degree bounded, *see* Mesh
- Local degree boundedness, 45, 48
- Local planarisation, 45
- Local support property, 89
- Local topology change, 52
- Logical midpoint, 32
- Möbius strip, 12
- Macro splines
 - Clough-Tocher, 1
 - Powell-Sabin, 1
- Matrix
 - composite, 99
 - diagonal element, 83
 - diagonally dominant, 5, 81, 83, 98
 - factorisation, 88
 - multiplication, 99
 - non-singular, 81, 98
 - of convex combinations, *see* convex combination
 - positive definite, 98–100
 - rank, 82, 99
 - sparse, 89
 - square, 82
 - stacked, 5, 99, 100
 - symmetric, 98–100
 - transpose, 104
- Memory footprint, 101
- Mesh, 12, 19, 23, 79
 - boundary, 24
 - edges, *see* mesh, edge
 - simple, 24
 - closed, 24
 - connectivity, 20
 - control, 1
 - corner, 14, 16, 23, 25, 58, 83
 - consecutive ordering, 24
 - definition of, 14
 - edge in, 60
 - edge out, 60
 - example of, 16
 - incoming edge, 14
 - outgoing edge, 14
 - dart, 20
 - definition of, 16
 - projection, 22
 - projection definition, 22
 - query, 21, 22
 - query definition, 21
 - definition of, 12
 - dual, 24, 28
 - definition of, 25
 - edge, 12, 14, 19, 23, 56, 57
 - boundary, 16
 - definition of, 12
 - incoming, *see* mesh, corner
 - non-boundary, 16
 - outgoing, *see* mesh, corner
 - edge-end, 16
 - definition of, 16
 - edges, 12
 - embedded, 12, 25
 - face, 12, 14, 19, 23–25, 56, 57, 83
 - definition of, 12
 - half-edge, 16, 23
 - definition of, 16
 - inner, 25
 - local degree bounded, 28, 29
 - position, 14
 - quad, 14, 27
 - quadrilateral, *see* mesh, quad
 - query, 19, 21
 - example, 23
 - refinement, *see* Mesh, subdivision
 - scheme
 - scheme, 12
 - simple boundary, 12
 - splitting, *see* Mesh, subdivision scheme
 - subdivision scheme
 - Doo-Sabin, 28
 - polyhedral, 24–26, 28
 - vertex, 24, 27
 - triangle, 83
 - triangular mesh, 14
 - vertex, 12, 14, 19, 23–25, 27, 56
 - definition of, 12
 - degree, 27
 - vertices
 - of face, 14

- Mesh notation, 1, 2
- Mesh query
 - definition of, 23
- Mesh refinement, 45, 47
- Mid-edge subdivision, *see* Mesh, subdivision scheme
- Midpoint
 - of interval, 33
 - of sub-interval, 33
- Minimal degree, 45
- Mixed derivatives, 50
- Model
 - bunny, 96
- Occlusion, 92
- Offset invariant ordered set, *see* set
- OpenGL, 89, 90, 100
 - HP_occlusion_test, 92
 - NV_occlusion_query, 92
 - context, 100, 101
 - coordinate system
 - clip, 90, 91
 - normalised device, 91
 - window, 91
 - world, 90, 91
 - display list, 92
 - GLUT, 101
 - occlusion test, 92
 - rasterisation, 92
 - texture, 101
 - textures, 100
 - transformation
 - modelview matrix, 90, 91
 - perspective division, 90, 91
 - projection matrix, 90, 91
 - viewport transform, 90, 91
 - vertex transformation, 90
 - view frustum, 90
- Original, 95
- Outgoing edge, *see* mesh, corner
- Pair, 11, *see* set
- Parameter point, 95, 103
 - edge midpoint, 95
 - face midpoint, 95
 - vertex, 95
- Parameterisation, 95
- Patch boundary, 58
- PCP, *see* Polyhedron, planar cut
- Pixel count, 92
- Planar cut polyhedron, 45, 47, 57
- Plane equation, 91
- Polyhedral split, 47, 96, *see* Mesh, subdivision scheme
- Polyhedron, 25, 28, 29, 45
 - planar, 28
 - planar cut, 28
- Powell-Sabin, *see* macro splines
- Primary key, *see* relational algebra
- Project operator, *see* relational algebra
- Projective convexity, 47
- Quad edge, 47, 49
- Quad mesh, 47, 48, 55, 57, *see* mesh
- Quad midpoint, 56, 98
 - matrix, 80–83
 - local, 81–83
- Quad midpoint matrix, 81
- Quad midpoints, 55, 80
- Quadrilateral, 47, 96
- Rectangular patch, 45
- Refinement, 96, *see* Mesh, subdivision scheme
- Regular parameterisation, 9
- Relation, *see* relational algebra
- Relational algebra, 2
 - attribute, 18
 - Cartesian product, 18, 20
 - domain, 18, 19
 - foreign key, 18, 19
 - join operator, 20
 - primary key, 18
 - project operator, 18, 20, 22
 - relation, 18, 19
 - virtual, 20
 - select operator, 18, 21
 - tuple, 19
- Relational database, 2, 19
- Rendering, 100, 101
 - constant model, 89
 - dynamic model, 89
- Scheme, *see* mesh
- Select operator, *see* relational algebra
- Set
 - circuit, 12
 - definition of, 11
 - length of, 12
 - element, 11

- offset invariant ordered, 11
 - definition of, 11
- pair, 11, 12
 - definition of, 11
 - size of, 11
- Simple boundary, *see* mesh
- Simplified surface spline, 1, 2
 - parameter point, 58
- SimplifiedSurfaceSpline (class), 101, 103
- SimplifiedSurfaceSplineBasis (class), 101, 103
- Smoothness, 99
- Smoothness constraint, 45
- Space
 - simplified surface spline, 79
 - dimension, 86
- Splitting, *see* Mesh, subdivision scheme
- SQL, 20
 - AND, 22
 - AS, 21
 - CREATE
 - TABLE, 18–20
 - VIEW, 21
 - FOREIGN KEY, 18, 20
 - FROM, 18, 21, 22
 - PRIMARY KEY, 18–20
 - REFERENCES, 18, 20
 - SELECT, 18, 21, 22
 - WHERE, 18, 21, 22
- SSSReader (class), 101, 103
- SSSWriter (class), 103
- Stanford University, 96
- Statistics
 - average, 103
 - mean, 103
- Statistics (class), 103
- Sub-interval, 32, 33
 - midpoint, 33
- Subdivision
 - property, 2
- Subdivision surface, 87
- Surface
 - fair, 99
 - smoothness, 99
- Surface spline, 1, 2
 - C^1 , 45
 - simplified, *see* Simplified surface spline
- Surface spline curve, 31
 - definition of, 31
- Surface splines
 - C^1 , 1
- Tensor product
 - Bézier surface example, 7
- topological 2-manifold, 12
- Topological holes, 52
- Triangular mesh, *see* mesh
- Triangular patch, 45, 49
- Twist adjustment, 47, 50
- Vector space
 - linear, 79
- Vertex, *see* mesh
- Vertex classification, 91, 92
- View frustum, 92
- Wavelets
 - lazy, 97
- Zero surface, 80