INCREMENTAL ALGORITHMS FOR THE DESIGN OF TRIANGULAR-BASED SPLINE SURFACES

Dianna Xu

A Dissertation

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

2002

Professor Jean Gallier   Supervisor of Dissertation

Professor Benjamin Pierce   Graduate Group Chairperson

ACKNOWLEDGMENTS

I would like to thank my advisor Jean Gallier, who made this dissertation a most challenging and yet rewarding experience.

I would also like to thank my parents, whose unconditional support and understanding never failed to inspire me.

Special thanks to Marcus Haebler for his love and support, for being there when needed.

ABSTRACTS

INCREMENTAL ALGORITHMS FOR THE DESIGN OF TRIANGULAR-BASED SPLINE SURFACES

Dianna Xu

Professor Jean Gallier

Spline surfaces consisting of triangular patches have a number of advantages over their rectangular counterparts, such as the ability to handle surfaces of arbitrary topology.

Designing and interpolating triangular-based spline surfaces has been a research interest in the field of CAGD for some years. Algorithms for designing triangular splines with local flexibility was left by Ramshaw [Ram87] as an open problem in 1987. Although many approaches have been proposed in the years since, none could quite achieve the elegance and flexibility of the algorithms for designing rectangular splines surfaces. The difficulty with triangular spline surfaces is that unlike tensor-product surfaces, the familiar B-spline curve framework does not carry over.

We present a new de Boor-like algorithm to design triangular C^1-splines based on general triangulations of the parameter plane. Through careful analysis of the continuity constraints based on polar forms, we discovered a way of choosing strategic control points, so that the remaining control points are computed using a simple propagation scheme. Due to its local nature, the algorithm can be easily made incremental. The algorithm operates in linear time and handles holes and sharp corners easily. Preliminary results also suggest that the algorithm can be extended to C^2-splines.

Due to the amount of freedom our algorithm leaves around the vertex regions, it is readily extendable to handle interpolation. However, fairing methods are needed to improve the resulting surface quality.

We have also extended our algorithm to handle closed surfaces based on triangulated polyhedra. Parametric data fitting is achieved through G^1 triangular surfaces. We provide a new rigorous definition of a piecewise polynomial surface based on a triagulated polyhedron. We also define a new kind of geometric continuity associated with such a polynomial surface, the AG^k-continuity.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the early days of computer graphics, modeling was a matter of simple unstructured polygonal meshes. However, it quickly became clear that polygonal meshes take up large amounts of storage space, are difficult to manipulate, and most importantly, lack the fundamental smoothness required by such tasks as designing the body of an automobile or the wing of an aircraft.

Mathematically, polygonal lines and polygons are simply first-order linear approximations to curves and surfaces, respectively. The study of computer-based higher order free-form curves and surfaces then became the main focus of the field called Computer Aided Geometric Design or CAGD.

The general approach is to use the parametric method to represent these higher order functions. The resulting curves and surfaces are known as *parametric curves and surfaces*. Parametric curves define points on curve segments in 3D using three polynomials in one parameter (also called univariate). Parametric surfaces are natural extensions of parametric curves, and they define points on curved surface patches in

3D using three polynomials in two parameters (also called bivariate).

However, using a single parametric curve segment or surface patch poses many problems.

- First, it leads to high degrees either due to smoothness requirements or interpolation restraints. High degrees not only complicate computation, they also cause undesirable artifacts in curve or surface shape. For example, a high degree interpolating curve might look like this between two of the data points it is required to pass through, even if smoothness requirements are satisfied:

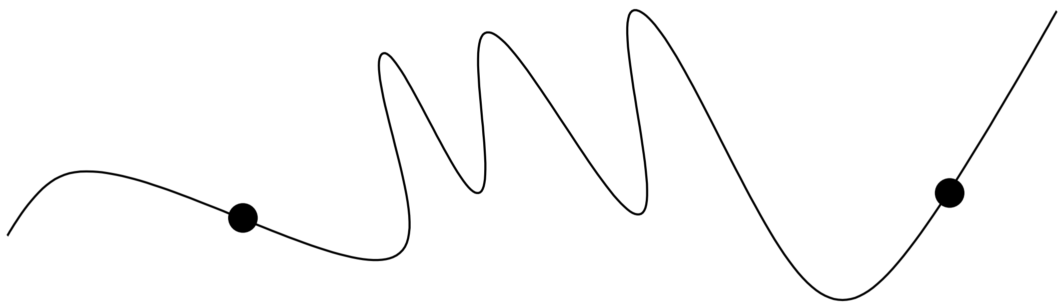


Figure 1.1: A wriggling interpolating curve

- More importantly, a single curve segment or surface patch lacks local control. That is, the modification of any small part of a curve or surface will affect the entire curve or surface. This forces recomputation of the entire curve or surface, as opposed to just the immediate neighborhood of such a curve or surface.

- Finally, because a single parametric curve or surface is dependent on all of its control points, computation of a single point on the curve or surface is much more expensive.

The solutions to all the aforementioned problems of simple parametric curves and surfaces are spline curves and surfaces. Splines and spline surfaces are simply piecewise parametric curve segments or surface patches joined together for lower degree, local control and more flexibility in shape design. Because these curve segments or surface patches need to be "pieced" together, there are natural smoothness concerns along the borders where they join. This leads to the concept of parametric continuity. Parametric continuity requires that up to a certain order, parametric derivatives must match at the boundaries.

1.1 Motivation

1.1.1 Triangular splines versus rectangular splines

Spline curves and surfaces have been used extensively in architecture, manufacturing, graphics modeling and animation and a variety of other fields in the past decade. The industry standard these days is unquestionably the nonuniform rational B-splines, or NURBS. To a large extent, NURBS' dominance can be attributed to its fundamental B-splines properties, such as built-in continuity, incrementality and local control. This is because the B-spline basis functions have local support, which means the curve segments depend on just a few of the control points. There are two major advantages to local control: moving one control point will only affect a small portion of the curve in the immediate neighborhood and computation of a point on the curve is greatly simplified as it depends on the number of control points involved.

There are two natural ways to extend B-spline curves into B-spline surfaces. The first method generates tensor-product surfaces, as it regards the B-spline surface as the tensor product (basically, Cartesian product) of B-spline curves in two directions. This method partitions the parameter plane into a rectangular grid, therefore, the resulting spline surfaces are also called rectangular spline surfaces. By construction, all B-spline properties extend readily to tensor-product surfaces, including local control and continuity constraints. Also all the techniques and algorithms for display carry over directly. The ease with which tensor product surfaces can be understood and implemented led to their overwhelming popularity. NURBS surfaces are all tensor product surfaces.

However, the rectangular nature of tensor product surfaces severely limits their ability to model surfaces of arbitrary topology. The alternative is to partition the parameter plane into triangles and erect spline patches based on the domain triangulations. Such surfaces are called triangular spline surfaces. Triangular splines have been the focus of CAGD research for some years, as they are the natural choice in a number of application areas:

- *Closed surfaces with arbitrary topology.*

  Tensor-product surfaces are not well suited to represent closed surfaces of arbitrary topology. Because they are based on a rectangular domain, they can only represent sheets, cylinders and tori without introducing surface tangent discontinuities. Other topologies require meshes with nonquadrilateral faces.

- *Surface fitting.*

  The problem of finding a surface from a general mesh of points has attracted much interest in the recent years. This is related to advances in areas like laser scanning technology, mesh reconstruction and mesh simplification. The majority of meshes coming from these areas are often triangular in nature.

- *Scattered data interpolation.*

  This is closely related to the area of surface fitting, except that interpolation generally poses more constraints than approximation, as the surface is required to pass through certain set of data points. It turns out that this problem is often more easily solved with triangular splines [Far83, FN91].

In addition, triangular splines also have some other desirable properties:

- *Generality.*

  Triangles are the most fundamental planar shapes. Triangular patches can be easily assembled to form rectangular patches, or any other patches of arbitrary number of sides.

- *Locality.*

  It is often the case that triangular splines have more local control than their tensor product counterparts. Though the comparisons can only be loosely conducted, as it depends on whether a single triangular patch is compared to a rectangular one, or each triangle is viewed as half of a rectangle, thereby comparing two triangular patches to a rectangular one [DeR85]. However triangular splines seem to have a bit more local flexibility because the patches are only three-sided.

- *Lower degree.*

  This is not necessarily true for the single patch per control mesh face approach, but split-patch approaches often come up with lower degree surfaces than tensor product surfaces for the same order of continuity.

1.1.2 Subdivsion surfaces

Besides the parametric spline surfaces, there is another class of surfaces that is able to represent closed surfaces with arbitrary topology. These are known as the "subdivision sufaces." With these techniques, polyhedra are subdivided repeatedly to approximate spline surfaces. There are rectangular (represented by Doo-Sabin [Doo78, DS78] and Catmull-Clark [CJ78] methods) as well as triangular (see Loop [Loo87]) subdivision surface algorithms. The implementation of these surfaces are straightforward, which has led to recent popularity of these surfaces, such as usage in motion pictures [DRKT98], and over a smooth domain surface [LMH00].

However, although it can been shown that the limit surface of a closed subdivided surface is G^1 continuous (see [BD88], [Zor96]), the continuity analysis is often complicated. But most importantly, there is no easy way to parameterize a subdivision surface for purposes such as texture-mapping.

1.1.3 The challenge of triangular splines

Despite the many nice properties of triangular spline surfaces, they remained largely a research interest. Because the familiar B-spline curve framework does not carry over, they lack the simplity of tensor product surfaces.

Ramshaw [Ram87] pointed out in his 1987 technical report on blossoming that while derivative constraints resulting from continuity conditions on the boundary curves indicate that square patch spline surfaces should be constructed from bipolynomial surfaces of degree (n, n), instead of arbitrary surfaces of degree $2n$, it is not clear what triangular patch spline surfaces should be constructed from, if not from arbitrary surfaces of degree $3n + 1$. He went on to propose this open problem as the following challenge:

Challenge 1.2 *Find a natural way to blossom a triangular-patch surface $F \colon P \to \mathbb{A}^3$ of degree $3n+1$, that is, a way that builds in the appropriate derivative constraints.*

This was the original inspiration of our research. There are many different proposals to construct triangular Bézier spline surfaces or triangular B-spline surfaces, but while theoretically sound, they all failed to provide a simple and intuitive way of designing and manipulating the surface shape. In short, an algorithm in the spirit of the elegant de Boor algorithm [Far92, Gal00a] for tensor product surfaces does not exist for triangular surfaces.

This work introduces a new algorithm to design smooth triangular spline surfaces with quintic Bézier patches. We believe that our approach is simple, geometrically intuitive and computationally efficient. Our algorithm is designed to satisfy the following goals:

- Design of complex 3D shapes.

- Shapes may have arbitrarily many holes and sharp corners.
- Shape modification by an incremental algorithm.
- Local control so that modifications only affect local neighborhood.
- Reasonable smoothness (C^1).

Preliminary results show that the same scheme may be extended to C^2, with higher degree patches. Also our approach can be extended to handle interpolation almost immediately, due to the freedom around each vertex region. However there is a degrading effect in the surface quality without further fairing methods. *Surface fairing* is a term that describes the "smoothing" of a surface to satisfy aesthetic requirements. Additional degrees of freedom are often allowed to improve global distribution of curvature (or optimization of other parameters contributing to shape quality).

Most importantly, we will show that our approach can also be extended to handle closed surfaces based on triangulated polyhedra, instead of planar triangulations, while lowering the continuity to a G^1 surface. This is a non-trivial extension, and to our best knowledge, is also a new way of fitting G^1 splines over arbitrary topology. Our approach leads to a rigorous definition of triangular polynomial spline surfaces based on a triangulated polyhedron, and also associated geometric continuity for such surfaces.

The most relevant previous attempts we are Loop [Loo94], which cannot handle interpolation and Hahmann et al [HBT00, HBT01], which extended Loop's method with a split-domain approach to achieve interpolation.

Note that the first and foremost aim of this work is to come up with an algorithm for designing triangular spline surfaces. Because our framework allows enough freedom for a theoretically easy extension to interpolation, it is always in the back of our minds. In actual research, care has been taken to always take extension to interpolation into account. However, so far we have not conducted any substantial investigation to make claims for interpolation results. The algorithms we present here are best thought of as producing surfaces that approximate a given triangular mesh, especially in the G^1 case.

1.3 Overview

This work logically consists of four major parts.

I. The first part is the entire Chapter 2. It provides related work and useful mathematical background for this work. Section 2.1 surveys previous work related to our C^1 algorithm. Sections 2.2, 2.3, 2.4, 2.5, 2.6 and 2.7 provide background reading on mathematical concepts that we rely on heavily to obtain our results. It includes affine geometry, parametric curves and surfaces, Bézier and B-splines and polar forms for parametric curves and surfaces.

II. The second part is concerned with our algorithm to fit C^1 splines over planar triangulations. In Chapter 3 we move on to examine in detail the continuity conditions for two triangular patches to join with C^n continuity. All the studies are done through polar forms. In this chapter we lay down the theoretical foundations for our algorithm.

Chapter 4 discusses our algorithm in detail. Section 4.1 provides a detailed sketch of our algorithm based on equilateral triangulations. In Section 4.2 it is then extended to irregular triangulations. Section 4.3 discusses the local control aspect of the algorithm, and Section 4.4 explores the strengths and weaknesses of the algorithm and proposes directions for future work.

III. The third part presents the extension of the algorithm to construct closed surfaces based on triangulated polyhedra of arbitrary topology. In Chapter 5, Section 5.1 presents previous work on geometric continuity and Section 5.2 gives a formal definition of a triangulated polyhedron. Our definition of AG^k-continuous surfaces is presented in Section 5.3, Section 5.4 presents some preliminary results of fitting AG^1 surfaces.

Chapter 6 proceeds to explore two approaches to construct an algorithm that fits AG^1 surfaces over triangulated polyhedra. The global approach is discussed in Section 6.1 and the local approach and the subsequent lowering of continuity conditions are discussed in Section 6.2.

IV. Lastly, a description of the current implementation is given in Chapter 7 and conclusions and future work are given in Chapter 8.

The decision was made to separate the related work into two sections, one containing those related to C^1 triangular splines and the other those related to G^1 splines. This method was chosen because although in our method, moving to G^1 closed surfaces was the natural next step, these two problems are actually quite different in nature, with the G^1 fitting being a lot harder and consequently having a larger body of related literature.

1.4 Notation

The conventions used throughout this presentation are as follows:

- Points are denoted by lower case letters such as a.
- Vectors are denoted by one or two lower case letters topped by an arrow such as $\overrightarrow{v}$ or $\overrightarrow{ab}$. In the latter case, $\overrightarrow{ab}$ is a vector that goes from point a to point b.
- Vector spaces are denoted by blackboard bold upper case letters such as $\mathbb{A}$.
- Coefficients and scalars are denoted by greek letters such as λ.
- Polynomial maps are denoted by upper case letters such as F, and their associated polar forms are denoted by the lower case of the same letters, such as f.
- t is the standard symbol used for curve parameters, while u and v are used for parameters representing a surface.
- Given a function $f(t)$, the ith derivative of f is denoted by $f^{(i)}(t)$.

Chapter 2

Related Work and Background Review

2.1 Related works

Note that in this section, we only survey work related to fitting C^1 triangular spline surfaces. Previous work on geometric continuity and fitting G^1 curves and surfaces is given separately in Section 5.1. Sometimes it is hard to make a sharp separation because in many approaches C^1 is a special case of G^1. In general, if a study seems to focus more on G^1, then it is included in the G^1 section.

C^1 surface fitting methods can first be divided into global and local categories. Global methods always involve solving systems of equations which generally become large and unmanageable as the data set gets bigger. Thus, majority of the methods are local, although many of them do resort to global measures for shape fairing.

An extensive survey of local methods for scattered data interpolation has been conducted by Franke [Fra82]. It is an excellent starting point for literature on this topic. According to Franke, the methods can be divided into inverse distance weighted methods due to Shepard [She68], rectangle based methods, triangle based methods and finite element based methods. We will only look at triangle based methods and finite element based methods because they are all based on a domain triangulation, and are, therefore, most relevant.

Early triangle based methods were all blending schemes with weight functions, such as Gold et al [GCR77] and Franke and Nielson [FN83]. There has been no recent literature in this category to our best knowledge, most likely due to the appearance and popularity of geometric continuity and G^1 surfaces.

Akima's method [Aki78] belongs to the finite element category, but it is in fact the closest in spirit to our approach. It fits a C^1 surface using quintic finite elements, and requires estimation of derivatives. In general estimating derivatives seems a common theme in most finite element approaches, which is very similar to our tangent plane placement problem (see Section 4.4.5). There are also finite element approaches using split-domain schemes, such as Lawson [Law77].

2.2 Affine geometry and affine maps

Our main results make heavy use of affine geometry and its properties. It is therefore appropriate to first introduce some of the definitions and notations here. This is only intended as a brief review of affine geometry. For more information on affine geometry, please refer to Gallier [Gal00a, Gal00b] and Berger [Ber90].

Affine space arises naturally from the need of coordinate-independent systems.

The ability to focus on the object studied alone and not on its relations to some arbitrary coordinate system is highly desirable.

The key to the problem of frame invariance is to make a clear distinction between points and vectors. Thus the basic idea of an affine space is to duplicate $\mathbb{R}^3$ into two copies, the first copy corresponding to points in $\mathbb{R}^3$, with no structure, and the second copy corresponding to free vectors with vector space structure, acting on the points of the first copy [Gal00a].

Definition 2.2.1 An *affine space* is either the degenerate space reduced to the empty set, or a triple $\langle E, \overrightarrow{E}, +\rangle$ consisting of a nonempty set E (of *points*), a vector space $\overrightarrow{E}$ (of *translations*, or *free vectors*), and an action $+ \colon E \times \overrightarrow{E} \to E$, satisfying the following conditions.

(A1) $a + \overrightarrow{0} = a$, for every $a \in E$.

(A2) $(a + \overrightarrow{u}) + \overrightarrow{v} = a + (\overrightarrow{u} + \overrightarrow{v})$, for every $a \in E$, and every $\overrightarrow{u}, \overrightarrow{v} \in \overrightarrow{E}$.

(A3) For any two points $a, b \in E$, there is a unique $\overrightarrow{u} \in \overrightarrow{E}$ such that $a + \overrightarrow{u} = b$.

Remark: The unique vector $\overrightarrow{u} \in \overrightarrow{E}$ such that $a + \overrightarrow{u} = b$ is denoted by $\overrightarrow{ab}$. Thus, we also write $b = a + \overrightarrow{ab}$. We also denote $\langle \mathbb{R}^n, \mathbb{R}^n, +\rangle$ by $\mathbb{A}^n$.

The fundamental operation on points in affine space is the *affine combination*. An affine combination is the equivalent of a linear combination of points in a vector space. A linear combination is frame dependent, and as we shall see, some restriction is necessary to achieve the desired frame independence for an affine combination. This notion of frame independence requires careful definition such as Lemma 2.4.1 of [Gal00a]. This lemma guarantees that given an affine space $\mathbb{A}^n$, for any family of points, $a_0, \ldots, a_n$, and any family of scalars, $\lambda_0, \ldots, \lambda_n$, such that $\sum_{i=0}^{n} \lambda_i = 1$, the point

$$x = a + \sum_{i=0}^{n} \lambda_i \overrightarrow{aa_i}$$

is independent of the choice of the origin $a \in \mathbb{A}^n$.

Definition 2.2.2 Given an affine space $\mathbb{A}^n$, for any family of points, $a_0, \ldots, a_n$, the *affine combination* of this family of points is denoted by

$$\sum_{i=0}^{n} \lambda_i a_i, \ \textit{with} \sum_{i=0}^{n} \lambda_i = 1, \ \textit{where} \ 0 \leq \lambda_i \leq 1$$

An *affine combination* is also called a *barycentric combination* or a *barycenter*.

In linear algebra, the notion of linear combination lends to the definition of a vector subspace as a non-empty set of a vector space closed under linear combinations. Similarly, affine combinations lead naturally to a definition of affine subspaces.

Definition 2.2.3 Given an affine space $\langle E, \overrightarrow{E}, + \rangle$, a subset V of E is an *affine subspace* (*of* $\langle E, \overrightarrow{E}, + \rangle$) if for every family of weighted points $((a_i, \lambda_i))_{i \in I}$ in V such that $\sum_{i \in I} \lambda_i = 1$, the affine combination $\sum_{i \in I} \lambda_i a_i$ belongs to V.

Lemma 2.2.4 *Given an affine space $\langle E, \overrightarrow{E}, +\rangle$, for any family $(a_i)_{i\in I}$ of points in E, the set V of barycenters $\sum_{i\in I}\lambda_i a_i$ (where $\sum_{i\in I}\lambda_i = 1$) is the smallest affine subspace containing $(a_i)_{i\in I}$.*

Proof. The proof is straight-forward, because if $(a_i)_{i\in I}$ is empty, then $V = \emptyset$, because of the condition $\sum_{i\in I}\lambda_i = 1$. If $(a_i)_{i\in I}$ is nonempty, then by definition the smallest affine subspace containing $(a_i)_{i\in I}$ must contain the set V of barycenters $\sum_{i\in I}\lambda_i a_i$, and thus, it is enough to show that V is closed under affine combinations, which is immediately verified. $\square$

Given a family $(a_i)_{i\in I}$ of points in E, the set V of barycenters $\sum_{i\in I}\lambda_i a_i$ (where $\sum_{i\in I}\lambda_i = 1$) is also called the *affine span* of $(a_i)_{i\in I}$. This naturally suggests the notion of an affine basis.

Let $\langle E, \overrightarrow{E}, +\rangle$ be a nonempty affine space, and let $(a_0, \ldots, a_m)$ be a family of $m + 1$ points in E. The family $(a_0, \ldots, a_m)$ determines the family of m vectors $(\overrightarrow{a_0a_1}, \ldots, \overrightarrow{a_0a_m})$ in $\overrightarrow{E}$. When $(\overrightarrow{a_0a_1}, \ldots, \overrightarrow{a_0a_m})$ is a basis of $\overrightarrow{E}$, then for every $x \in E$, since $x = a_0 + \overrightarrow{a_0x}$, there is a unique family $(x_1, \ldots, x_m)$ of scalars such that

$$x = a_0 + x_1\overrightarrow{a_0a_1} + \cdots + x_m\overrightarrow{a_0a_m}.$$

The scalars $(x_1, \ldots, x_m)$ may be considered as coordinates with respect to the basis $(a_0, (\overrightarrow{a_0a_1}, \ldots, \overrightarrow{a_0a_m}))$.

Definition 2.2.5 Given an affine space $\langle E, \overrightarrow{E}, +\rangle$, an *affine frame with origin* a_0 is a family $(a_0, \ldots, a_m)$ of $m+1$ points in E such that the list of vectors $(\overrightarrow{a_0a_1}, \ldots, \overrightarrow{a_0a_m})$

is a basis of $\overrightarrow{E}$. The pair $(a_0, (\overrightarrow{a_0a_1}, \ldots, \overrightarrow{a_0a_m}))$ is also called an *affine frame with origin* a_0.

Just like vector spaces come with linear maps, affine spaces come with affine maps.

Definition 2.2.6 Given two affine spaces $\langle E, \overrightarrow{E}, +\rangle$ and $\langle E', \overrightarrow{E'}, +'\rangle$, a map $f\colon E \to E'$ is *affine* iff for every family of points a_i in E and every family of coefficients λ_i such that $\sum_{i=0}^{n} \lambda_i = 1$,

$$f\left(\sum_{i=0}^{n} \lambda_i a_i\right) = \sum_{i=0}^{n} \lambda_i f(a_i).$$

That is, an affine map preserves affine combinations. Every affine map is a composition of a linear map and a translation.

2.3 Polynomial curves and surfaces

A mathematical representation of a curve or a curved surface is usually a system of equations. Frequently, a curve or a curved surface is thought of as consisting of a number of curve segments or surface patches that are joined together, with some degree of smoothness where they join. Such composite curves are called *splines*, and such composite surfaces are called *spline surfaces.*

The *parametric model* where a shape is viewed as the range of a system of equations, is generally preferred over the *implicit model*, where a shape is viewed as the zero locus of the system of equations, because the former allows more freedom, e.g., the possibilities of a *parametrization*, that is, the assignment of a point and a parameter in some given inteval is not unique. Furthermore, the *implicit model* assumes the existence of real solutions to the system of equations, which is not always possible.

Definition 2.3.1 Given the affine line $\mathbb{A}$, any affine space $\mathbb{A}^n$ of finite dimension n and any affine frame $(a_0, (\overrightarrow{a_0a_1}, \ldots, \overrightarrow{a_0a_n}))$ for $\mathbb{A}^n$, a (parametrized) polynomial curve of degree (at most) m is a map $F\colon \mathbb{A} \to \mathbb{A}^n$, such that for every $\overline{t} \in \mathbb{A}$,

$$F(\overline{t}) = a_0 + F_1(t)\overrightarrow{a_0a_1} + \ldots + F_n(t)\overrightarrow{a_0a_n},$$

where $\overline{t} = (1-t)0+t1$, and every $F_i(X)$ is a polynomial of degree $\leq m$, with $1 \leq i \leq n$.

Given any $r, s \in \mathbb{A}$, with $r < s$, a (parametrized) polynomial curve segment $F([r, s])$ of degree (at most) m is the restriction $F\colon [r, s] \to \mathbb{A}^n$ of a polynomial curve $F\colon \mathbb{A} \to \mathbb{A}^n$. The *trace* of a polynomial curve F is defined as $F(\mathbb{A})$, and therefore the trace of $F[r, s]$ is $F([r, s])$.

Polynomial curves can also be defined as a set of barycenters of a finite set of points. Since barycenters of a set of points can be obtained from repeated computations of barycenters of two points, this leads to the famous de Casteljau algorithm and the Bézier curves. This finite set (in fact, sequence) of points is called the *(Bézier) control points*.

2.4 Bézier and B-splines

Polynomial curves and surfaces are simple and easily computed using the de Casteljau algorithm (see Sections 2.6.2 and 2.7.4), but the degree of a polynomial curve or surface is directly related to the number of control points given. For example, a polynomial curve is of degree one less than the total number of control points in the control polygon. The lack of local control is also undesirable. Splines became the natural replacement to simple polynomial curves and surfaces.

The so-called “natural” splines are most often cubics, because they are the lowest degree true space (i.e. nonplanar) curves, as well as the lowest degree C^1 interpolating curves.

However, in design more freedom is often preferred, and therefore approximation is needed more than interpolation. The Bézier curves, named after Pierre Bézier, who developed them for designing automobiles for Renault, give such freedom. Bézier curves are splines of the *blended form*. Blended splines are splines whose curve segments are of the form

$$P(t) = \sum_{i=0}^{n} b_i F_i^k(t),\ where\ t \in [0,1],$$

where b_i are the control points and $F_i^k(t)$ are some basis functions, which together produce a weighted average of the control points. In the case of Bézier splines, the basis functions turn out to be the Bernstein polynomials [Far92, Gal00a].

Motivated by Bézier curves, Gordon and Riesenfeld [GR74b, GR74a] extended them to B-splines. The Bernstein basis functions present in the Bézier curves are replaced by B-spline basis functions, which then produce vector-valued B-spline approximation instead of the Bernstein approximations. The B-spline basis functions have some important advantages over the Bernstein polynomials, one of which is local support, which means B-spline curves have local control.

B-spline curves have been studied extensively and are well understood. The well-known de Boor algorithm provides an elegant way to compute points on a B-spline curve, and the de Boor control points allow the curve segments of polar degree m to join at C^k continuity, $0 \leq k \leq m$ [Gal00a]. The de Boor algorithm is also known as a progressive generalization of the de Casteljau algorithm for Bézier curves, because it has incrementality built in. The de Boor scheme can be generalized nicely to join tensor product spline surfaces (rectangular spline surfaces) of bidegree $< m, m >$, but unfortunately, there is no known method involving control points for joining triangular spline surfaces similar to the de Boor scheme for the tensor product surfaces [Ram87].

For more information on parametric curves and surfaces, including Bézier and B-spline curves and surfaces, please refer to Farin [Far92] and Gallier [Gal00a].

2.5 Multiaffine maps and polar forms

The principle of (affine) "blossoming" was first introduced by de Casteljau [dC86] and Ramshaw [Ram88]. Yet, it is in fact much older and well known to the algebraists. An example is Weyl's book on classical groups from 1946 [Wey46]. Blossoming can be viewed as a generalization of the well-known de Casteljau algorithm. In many cases, it is more convenient to study curves and surfaces from their polar forms or "blossoms"[Ram89], because it provides a way of labeling the Bézier or the de Boor control points with symmetric, multivariate labels. As we shall see in Sections 2.6 and 2.7, these labels provide excellent geometric intuitions.

Since their introduction, polar forms have gained considerable popularity because of the geometric insights they provided to Bézier and B-splines related algorithms. The theory of blossoming has been applied to B-spline knot insertion [Sei88, Sei89], triangular B-spline surfaces [MS92] and geometrically continuous spline curves [Sei93]. For a more in-depth reading on the theory of blossoming, the readers are referred to Ramshaw [Ram87] and Gallier [Gal00a].

Readers are assumed to have some degree of familiarity with the way polynomial curves and surfaces are handled through their control points. However, since polar forms are excellent tools to study this, we will also include discussions in Sections 2.6 and 2.7 for review purposes. Descriptions of the de Casteljau algorithm for polynomial curves and surfaces are also given in Sections 2.6.2 and 2.7.4 respectively.

Since polar forms are in fact symmetric multiaffine maps, we first discuss the representation of polynomial maps in terms of multiaffine maps.

Definition 2.5.1 A function $f\colon \underbrace{\mathbb{A}^d \times \cdots \times \mathbb{A}^d}_{m} \to \mathbb{A}^n$ is a *multiaffine map* (or an *m-affine map*), iff it is affine with respect to each of its argument, that is, for every i, with $1 \leq i \leq m$, considering $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n$ fixed, for all $a_i \in \mathbb{A}^d$, the map

$$a_i \to f(a_0, \ldots, a_i, \ldots, a_m)$$

is affine.

Definition 2.5.2 A *symmetric* map is a function that is invariant under permutation of its arguments, that is, the result of the function does not depend on any particular order of its arguments. In other words, given a map $f\colon \underbrace{\mathbb{A}^d \times \cdots \times \mathbb{A}^d}_{m} \to \mathbb{A}^n$,

$$f(a_{\pi(1)}, \ldots, a_{\pi(m)}) = f(a_1, \ldots, a_m),$$

for all $a_1, \ldots, a_m$, and all permutations π.

We can use multiaffine maps to define generalized polynomial maps between two affine spaces of arbitrary dimensions. In the special case where the multiaffine map maps from $\mathbb{A}^n$ to $\mathbb{A}$, it is equivalent to the notion of a polynomial function induced by a polynomial in n variables. In fact, every polynomial curve of degree m has a unique symmetric m-affine map associated with it. It is called the *m-polar* form of the curve, or its "blossom".

For example, consider the parabola $F\colon \mathbb{A} \to \mathbb{A}^2$, given by

$$\begin{aligned} x(t) &= 4t, \\ y(t) &= t^2 - 3t + 2. \end{aligned}$$

The polynomial map $F\colon \mathbb{A} \to \mathbb{A}^2$ comes from a unique symmetric biaffine map $f\colon \mathbb{A}^2 \to \mathbb{A}^2$, where

$$\begin{aligned} f_1(t_1, t_2) &= 2(t_1 + t_2), \\ f_2(t_1, t_2) &= t_1 t_2 - \frac{3}{2}(t_1 + t_2) + 2, \quad \text{such that} \end{aligned}$$

$$F(t) = f(t, t), \quad \text{for all } t \in \mathbb{A}.$$

2.6 Polynomial Curves and Polar Forms

2.6.1 Polar forms and control points

We now show how polynomial curves are handled in terms of control points. This is just a brief review. A comprehensive treatment of polynomial curves can be found in Gallier [Gal00a]).

The key to the treatment of polynomial curves in terms of control points is that polynomials can be multilinearized. Using definitions in Section 2.5, for every polynomial $F(t)$ of degree m, there is a unique symmetric and multiaffine map $f\colon \mathbb{R}^m \to \mathbb{R}$ such that

$$F(t) = f(\underbrace{t, \dots, t}_{m}), \quad \text{for all } t \in \mathbb{R}.$$

This is an old "folk theorem", probably already known to Newton. The proof is easy. By linearity, it is enough to consider a monomial of the form x^k, where $k \leq m$. The unique symmetric multiaffine map corresponding to x^k is

$$\frac{\sigma_k(t_1, \dots, t_m)}{\binom{m}{k}},$$

where $\sigma_k(t_1, \dots, t_m)$ is the kth symmetric function in m variables, i.e.,

$$\sigma_k = \sum_{\substack{I \subseteq \{1,\dots,m\} \\ |I|=k}} \left(\prod_{i \in I} t_i \right).$$

Given a polynomial curve $F\colon \mathbb{R} \to \mathbb{R}^n$ of degree m

$$x_1(t) = F_1(t),$$

$$\ldots = \ldots$$

$$x_n(t) = F_n(t),$$

where $F_1(t), \ldots, F_n(t)$ are polynomials of degree at most m, the curve $F\colon \mathbb{R} \to \mathbb{R}^n$ arises from a unique symmetric multiaffine map $f\colon \mathbb{R}^m \to \mathbb{R}^n$, the *polar form of* F, such that

$$F(t) = f(\underbrace{t, \ldots, t}_{m}),$$

for all $t \in \mathbb{R}$ (see Ramshaw [Ram87], Farin [Far92, Far95], Hoschek and Lasser [HL93], or Gallier [Gal00a]).

For example, consider the plane cubic defined as follows:

$$F_1(t) = \frac{3}{4}t^2 - \frac{3}{2}t - \frac{9}{4}, \qquad F_2(t) = \frac{3}{4}t^3 - \frac{3}{2}t^2 - \frac{9}{4}t.$$

We get the polar forms

$$f_1(t_1, t_2, t_3) = \frac{1}{4}(t_1t_2 + t_1t_3 + t_2t_3) - \frac{1}{2}(t_1 + t_2 + t_3) - \frac{9}{4}$$

$$f_2(t_1, t_2, t_3) = \frac{3}{4}t_1t_2t_3 - \frac{1}{2}(t_1t_2 + t_1t_3 + t_2t_3) - \frac{3}{4}(t_1 + t_2 + t_3).$$

Also, for $r \neq s$, the map $f\colon \mathbb{R}^m \to \mathbb{R}^n$ is determined by the $m+1$ *control points* $(b_0, \ldots, b_m)$, where

$$b_i = f(\underbrace{r, \ldots, r}_{m-i}, \underbrace{s, \ldots, s}_{i}),$$

since

$$f(t_1,\ldots,t_m)=\sum_{k=0}^{m}\sum_{\substack{I\cup J=\{1,\ldots,m\}\\ I\cap J=\emptyset,\,\mathrm{card}(J)=k}}\prod_{i\in I}\left(\frac{s-t_i}{s-r}\right)\prod_{j\in J}\left(\frac{t_j-r}{s-r}\right)f(\underbrace{r,\ldots,r}_{m-k},\underbrace{s,\ldots,s}_{k}).$$

For example, with respect to the affine frame $r=-1$, $s=3$, the coordinates of the control points of the cubic defined earlier are:

$$b_0=(0,0)$$
$$b_1=(-4,4)$$
$$b_2=(-4,-12)$$
$$b_3=(0,0).$$

Conversely, for every sequence of $m+1$ points $(b_0,\ldots,b_m)$, there is a unique symmetric multiaffine map f such that

$$b_i=f(\underbrace{r,\ldots,r}_{m-i},\underbrace{s,\ldots,s}_{i}),$$

namely

$$f(t_1,\ldots,t_m)=\sum_{k=0}^{m}\sum_{\substack{I\cup J=\{1,\ldots,m\}\\ I\cap J=\emptyset,\,\mathrm{card}(J)=k}}\prod_{i\in I}\left(\frac{s-t_i}{s-r}\right)\prod_{j\in J}\left(\frac{t_j-r}{s-r}\right)b_k.$$

Thus, there is a bijection between the set of polynomial curves of degree m and the set of sequences $(b_0,\ldots,b_m)$ of $m+1$ control points.

Figure 2.1 shows four control points b_0, b_1, b_2, b_3 specifying a polynomial curve of degree 3, where $b_0=f(r,r,r)$, $b_1=f(r,r,s)$, $b_2=f(r,s,s)$, $b_3=f(s,s,s)$.

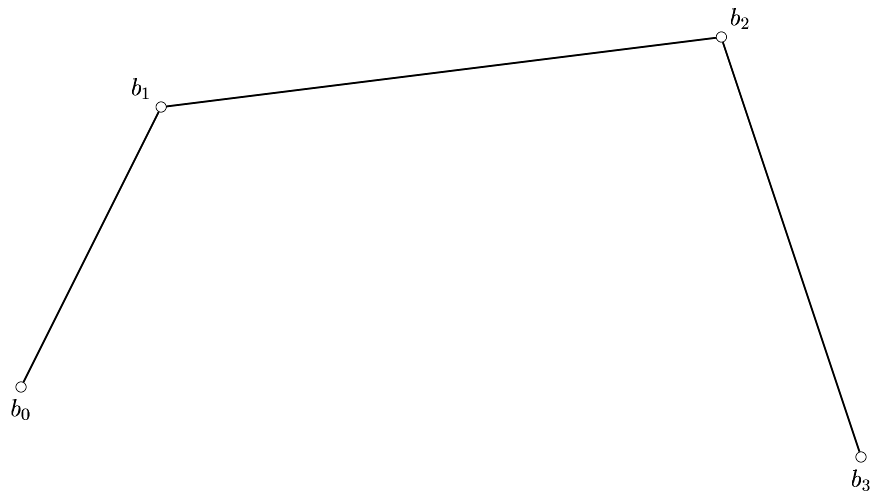


Figure 2.1: Control points and control polygon

The upshot of all this is that for algorithmic purposes, it is convenient to define polynomial curves in terms of polar forms. Recall that the canonical affine space associated with the field $\mathbb{R}$ is denoted as $\mathbb{A}$.

Definition 2.6.1 A (parameterized) *polynomial curve in polar form of degree m* is an affine polynomial map $F\colon \mathbb{A} \to \mathcal{E}$ of polar degree m, defined by its *m-polar form*, which is some symmetric m-affine map $f\colon \mathbb{A}^m \to \mathcal{E}$, where $\mathbb{A}$ is the real affine line, and $\mathcal{E}$ is any affine space (of dimension at least 2). Given any $r, s \in \mathbb{A}$, with $r < s$, a (parameterized) *polynomial curve segment $F([r, s])$ in polar form of degree m* is the restriction $F\colon [r, s] \to \mathcal{E}$ of an affine polynomial curve $F\colon \mathbb{A} \to \mathcal{E}$ in polar form of degree m. We define the the *trace of F* as $F(\mathbb{A})$, and the the *trace of $F[r, s]$* as $F([r, s])$.

Remark: Typically, the affine space $\mathcal{E}$ is the real affine space $\mathbb{A}^3$ of dimension 3. Note that we define a polynomial curve in polar form of degree at most m, rather than a polynomial curve in polar form of degree exactly m, because an affine polynomial map f of polar degree m may end up being degenerate, in the sense that it could be equivalent to a polynomial map of lower polar degree. For convenience, we will allow ourselves the abuse of language where we abbreviate "polynomial curve in polar form" to "polynomial curve".

We summarize the relationship between control points and polynomial curves in the following lemma:

Lemma 2.6.2 *Given any sequence of $m+1$ points $a_0, \ldots, a_m$ in some affine space $\mathcal{E}$, there is a unique polynomial curve $F\colon \mathbb{A} \to \mathcal{E}$ of degree m, whose polar form $f\colon \mathbb{A}^m \to \mathcal{E}$ satisfies the conditions*

$$f(\underbrace{r, \ldots, r}_{m-k}, \underbrace{s, \ldots, s}_{k}) = a_k,$$

(where $r, s \in \mathbb{A}$, $r \neq s$). Furthermore, the polar form f of F is given by the formula

$$f(t_1, \ldots, t_m) = \sum_{k=0}^{m} \sum_{\substack{I \cup J = \{1, \ldots, m\} \\ I \cap J = \emptyset,\, |J| = k}} \prod_{i \in I} \left(\frac{s - t_i}{s - r} \right) \prod_{j \in J} \left(\frac{t_j - r}{s - r} \right) a_k,$$

and $F(t)$ is given by the formula

$$F(t) = \sum_{k=0}^{m} B_k^m[r, s](t)\, a_k,$$

where the polynomials

$$B_k^m[r, s](t) = \binom{m}{k} \left(\frac{s - t}{s - r} \right)^{m-k} \left(\frac{t - r}{s - r} \right)^{k}$$

are the Bernstein polynomials of degree m over $[r, s]$.

Note that since the polar form f of a polynomial curve F of degree m is symmetric, the order of the arguments is irrelevant. Often, when arguments are repeated, we also omit commas between arguments. For example, we abbreviate

$$f(\underbrace{r,\ldots,r}_{i},\underbrace{s,\ldots,s}_{j})$$

as $f(r^i s^j)$. We will start using this convention from now on.

2.6.2 The de Casteljau algorithm

The definition of polynomial curves in terms of polar forms leads to a very nice algorithm known as the *de Casteljau algorithm*, to draw polynomial curves. Using the de Casteljau algorithm, it is possible to determine any point $F(t)$ on the curve, by repeated affine interpolations (see Farin [Far92, Far95], Hoschek and Lasser [HL93], Risler [Ris92], or Gallier [Gal00a]). Figure 2.2 below shows $F(1/2)$:

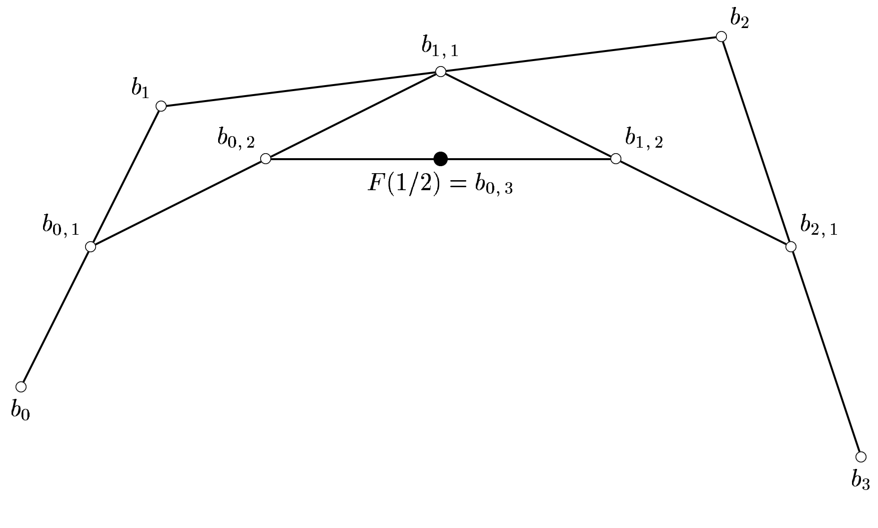


Figure 2.2: A de Casteljau diagram for $t = 1/2$

In the general case where a curve F is specified by $m+1$ control points $(b_0, \ldots, b_m)$ with respect to an interval $[r, s]$, let us define the following points $b_{i,j}$ used during the computation of $F(t)$ (where f is the polar form of F):

$$b_{i,j} = \begin{cases} b_i & \text{if } j = 0,\ 0 \leq i \leq m, \\ f(t^j r^{m-i-j} s^i) & \text{if } 1 \leq j \leq m,\ 0 \leq i \leq m - j. \end{cases}$$

Then, we have the following equations:

$$\boxed{b_{i,j} = \left(\frac{s-t}{s-r}\right) b_{i,j-1} + \left(\frac{t-r}{s-r}\right) b_{i+1,j-1}.}$$

The result is

$$\boxed{F(t) = b_{0,m}.}$$

The computation can be conveniently represented in the following triangular form, show in Table 2.1:

$$\begin{array}{ccccccccc}
0 & 1 & \dots & j-1 & j & \dots & m-k & \dots & m \\
b_{0,0} & & & & & & & & \\
 & b_{0,1} & & & & & & & \\
b_{1,0} & & \ddots & & & & & & \\
 & & & b_{0,j-1} & & & & & \\
 & & & \vdots & b_{0,j} & & & & \\
 & & & b_{i,j-1} & \vdots & \ddots & & & \\
 & & & & b_{i,j} & & b_{0,m-k} & & \\
 & & & b_{i+1,j-1} & \vdots & & & & \\
 & & & \vdots & b_{m-k-j,j} & & \vdots & & b_{0,m} \\
 & & & b_{m-k-j+1,j-1} & & & & & \\
 & & & & \vdots & & b_{k,m-k} & & \\
 & b_{m-k-1,1} & & \vdots & & & & & \\
b_{m-k,0} & & & & b_{m-j,j} & & & & \\
 & \vdots & & b_{m-j+1,j-1} & & & & & \\
\vdots & & & & & & & & \\
b_{m-1,0} & & & & & & & & \\
 & b_{m-1,1} & & & & & & & \\
b_{m,0} & & & & & & & &
\end{array}$$

Table 2.1: Computation of $F(t) = b_{0,m}$

When $r \leq t \leq s$, each interpolation step computes a convex combination, and $b_{i,j}$ lies between $b_{i,j-1}$ and $b_{i+1,j-1}$. In this case, geometrically the can be illustrated by a diagram consisting of the following m polylines:

$$(b_{0,0},\, b_{1,0}),\, (b_{1,0},\, b_{2,0}),\, (b_{2,0},\, b_{3,0}),\, (b_{3,0},\, b_{4,0}),\, \dots, (b_{m-1,0},\, b_{m,0})$$

$$(b_{0,1},\, b_{1,1}),\, (b_{1,1},\, b_{2,1}),\, (b_{2,1},\, b_{3,1}),\, \dots, (b_{m-2,1},\, b_{m-1,1})$$

$$(b_{0,2},\, b_{1,2}),\, (b_{1,2},\, b_{2,2}),\, \dots, (b_{m-3,2},\, b_{m-2,2})$$

$$\ldots$$

$$(b_{0,m-2},\, b_{1,m-2}),\, (b_{1,m-2},\, b_{2,m-2})$$

$$(b_{0,m-1},\, b_{1,m-1})$$

called *shells*, and with the point $b_{0,m}$, they form the *de Casteljau diagram*. Note that the shells are nested nicely. The polyline

$$(b_0,\, b_1),\, (b_1,\, b_2),\, (b_2,\, b_3),\, (b_3,\, b_4),\, \ldots, (b_{m-1},\, b_m)$$

is also called a *control polygon* of the curve. When t is outside $[r, s]$, we still obtain m shells and a de Casteljau diagram, but the shells are not nicely nested.

One of the best features of the de Casteljau algorithm is that it lends itself very well to recursion. Indeed, going back to the case of a cubic curve, it is easy to show that the sequences of points $(b_0, b_{0,1}, b_{0,2}, b_{0,3})$ and $(b_{0,3}, b_{1,2}, b_{2,1}, b_3)$ are also control polygons for the exact same curve (see Farin [Far92, Far95], Hoschek and Lasser [HL93], Gallier [Gal00a]).

Thus, we can compute the points corresponding to $t = 1/2$ with respect to the control polygons

$$(b_0, b_{0,1}, b_{0,2}, b_{0,3}) \quad \text{and} \quad (b_{0,3}, b_{1,2}, b_{2,1}, b_3),$$

and this yields a recursive method for approximating the curve. This method called the *subdivision method* applies to polynomial curves of any degree and can be used to render efficiently a curve segment F over $[r, s]$, as shown in Figure 2.3.

For much more on polynomial curves in terms of polar forms and control points, see Gallier [Gal00a].

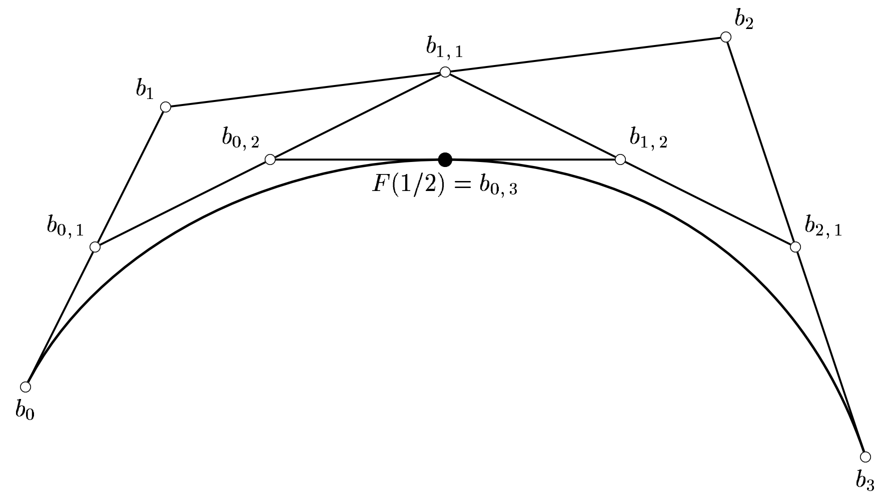


Figure 2.3: Approximating a curve using subdivision

2.7 Polynomial Surfaces and Polar Forms

2.7.1 Polarizing polynomial surfaces

A polynomial curve is easily generalized to a polynomial surface by having the polynomials take two parameters. Thus, we have a polynomial map $F(u, v)\colon \mathbb{A}^2 \to \mathbb{A}^3$.

For example, the following polynomials define a bipolynomial surface of degree $(3, 3)$, known as *Enneper's Surface*:

$$F_1(u, v) = u - \frac{u^3}{3} + uv^2$$

$$F_2(u, v) = v - \frac{v^3}{3} + u^2 v$$

$$F_3(u, v) = u^2 - v^2.$$

Recall that symmetric multi-affine maps can be used to define the notion of a polynomial function between two affine spaces of arbitrary dimension. We have seen in Section 2.5 that polynomial curves have associated polar forms. Similarly, we also have polar forms associated with polynomial surfaces.

To polarize a polynomial means to find the unique symmetric affine map associated with the given polynomial. Now, since we have polynomial surfaces, we have two parameter variables (u, v) involved in the polynomials, and there are two natural ways to polarize a given polynomial surface $F \colon \mathbb{A}^2 \to \mathbb{A}^3$.

The first approach is to polarize separately in the two variables, which yields bipolynomial surfaces, also commonly called tensor product surfaces. If p and q are the two degrees of the bipolynomial surface, then we get a $(p+q)$-multiaffine map which is symmetric in its first p arguments, and in its last q arguments, but not symmetric in all its arguments. This approach basically divides the parameter plane into rectangles.

The second approach treats the two variables (u, v) as a whole, namely, as coordinates of a point (u, v) in the affine plane. This approach produces total degree surfaces. If m is the degree of the total degree surface, then we get a m-affine map which is symmetric in all of its arguments. In this sense, this method is a natural generalization of the Bézier curves, and in fact, total degree surfaces (or Bézier triangular surfaces) were the first surfaces to be considered by de Casteljau himself. This approach triangulates the parameter plane.

A polynomial surface F of total degree m is completely specified by a (triangular) control net, or Bézier net consisting of control points.

2.7.2 Polar forms

We move on to show how polynomial surfaces are handled in terms of control points. A comprehensive treatment of polynomial surfaces can be found in Gallier [Gal00a]).

The deep reason why polynomial surfaces can be effectively handled in terms of control points is that multivariate polynomials arise from multiaffine symmetric maps (see Ramshaw [Ram87], Farin [Far92, Far95], Hoschek and Lasser [HL93], or Gallier [Gal00a]). Denoting the affine plane $\mathbb{R}^2$ as $\mathcal{P}$, traditionally, a *polynomial surface* in $\mathbb{R}^n$ is a function $F\colon \mathcal{P} \to \mathbb{R}^n$, defined such that

$$\begin{aligned} x_1 &= F_1(u, v), \\ \ldots &= \ldots \\ x_n &= F_n(u, v), \end{aligned}$$

for all $(u, v) \in \mathbb{R}^2$, where $F_1(U, V), \ldots, F_n(U, V)$ are polynomials in $\mathbb{R}[U, V]$.

We mentioned that there are two natural ways to polarize the polynomials defining F. The first way is to polarize *separately* in u and v. If p is the highest degree in u and q is the highest degree in v, we get a unique multiaffine map

$$f\colon (\mathbb{R})^p \times (\mathbb{R})^q \to \mathbb{R}^n$$

of degree $(p+q)$ which is symmetric in its first p arguments and symmetric in its last q arguments, such that

$$F(u, v) = f(\underbrace{u, \ldots, u}_{p}; \underbrace{v, \ldots, v}_{q}) \text{ or } F(u, v) = f(u^p v^q).$$

We get what is traditionally called a *tensor product surface*, or as we prefer to call it, a *bipolynomial surface of bidegree* $\langle p,\ q\rangle$ (or a *rectangular surface patch*). We also say that the multiaffine maps arising in polarizing separately in u and v are $\langle p,\ q\rangle$*-symmetric.*

The second way to polarize is to treat the variables u and v *as a whole*. This way, if F is a polynomial surface such that the maximum total degree of the monomials is m, we get a unique symmetric degree m multiaffine map

$$f\colon (\mathbb{R}^2)^m \to \mathbb{R}^n,$$

such that

$$F(u,v) = f(\underbrace{(u,v),\ldots,(u,v)}_{m}),\ \textit{or}\ F(u,v) = f((u,v)^m).$$

We get what is called a *total degree surface* (or a *triangular surface patch*).

Using linearity, all we have to do is to polarize a monomial $u^h v^k$. It is easily verified that the unique $\langle p,\ q\rangle$-symmetric multiaffine polar form of degree $p+q$

$$f^{p,q}_{h,k}(u_1,\ldots,u_p;v_1,\ldots,v_q)$$

of the monomial $u^h v^k$ is given by

$$f^{p,q}_{h,k}(u_1,\ldots,u_p;v_1,\ldots,v_q) = \frac{1}{\binom{p}{h}\binom{q}{k}} \sum_{\substack{I\subseteq\{1,\ldots,p\},|I|=h\\ J\subseteq\{1,\ldots,q\},|J|=k}} \left(\prod_{i\in I} u_i\right)\left(\prod_{j\in J} v_j\right).$$

The denominator $\binom{p}{h}\binom{q}{k}$ is the number of terms in the above sum.

It is easily verified that the unique symmetric multiaffine polar form of degree m

$$f^m_{h,k}((u_1, v_1), \dots, (u_m, v_m))$$

of the monomial $u^h v^k$ is given by

$$f^m_{h,k}((u_1, v_1), \dots, (u_m, v_m)) = \frac{1}{\binom{m}{h}\binom{m-h}{k}} \sum_{\substack{I \cup J \subseteq \{1,\dots,m\} \\ |I|=h, |J|=k, I\cap J=\emptyset}} \left(\prod_{i\in I} u_i\right)\left(\prod_{j\in J} v_j\right).$$

The denominator $\binom{m}{h}\binom{m-h}{k} = \binom{m}{h\;k\;(m-h-k)}$ is the number of terms in the above sum.

As an example, consider the following surface known as Enneper's surface:

$$F_1(U,V) = U - \frac{U^3}{3} + UV^2$$

$$F_2(U,V) = V - \frac{V^3}{3} + U^2V$$

$$F_3(U,V) = U^2 - V^2.$$

We get the polar forms

$$f_1((U_1,V_1),(U_2,V_2),(U_3,V_3)) = \frac{U_1+U_2+U_3}{3} - \frac{U_1U_2U_3}{3} + \frac{U_1V_2V_3 + U_2V_1V_3 + U_3V_1V_2}{3}$$

$$f_2((U_1,V_1),(U_2,V_2),(U_3,V_3)) = \frac{V_1+V_2+V_3}{3} - \frac{V_1V_2V_3}{3}$$

$$+\frac{U_1U_2V_3+U_1U_3V_2+U_2U_3V_1}{3}$$

$$f_3((U_1,V_1),(U_2,V_2),(U_3,V_3))=\frac{U_1U_2+U_1U_3+U_2U_3}{3}-\frac{V_1V_2+V_1V_3+V_2V_3}{3}.$$

2.7.3 Control points for triangular surfaces

Given an affine frame Δrst in the plane (where $r, s, t \in \mathcal{P}$ are affinely independent points), it turns out that any symmetric multiaffine map $f\colon \mathcal{P}^m \to \mathcal{E}$ is uniquely determined by a family of $\frac{(m+1)(m+2)}{2}$ points (where $\mathcal{E}$ is any affine space, say $\mathbb{R}^n$). Let

$$\Delta_m = \{(i,\, j,\, k) \in \mathbb{N}^3 \mid i+j+k=m\}.$$

The following lemma is easily shown (see Ramshaw [Ram87] or Gallier [Gal00a]).

Lemma 2.7.1 *Given a reference triangle Δrst in the affine plane $\mathcal{P}$, given any family $(b_{i,\,j,\,k})_{(i,j,k)\in\Delta_m}$ of $\frac{(m+1)(m+2)}{2}$ points in $\mathcal{E}$, there is a unique surface $F\colon \mathcal{P} \to \mathcal{E}$ of total degree m, defined by a symmetric m-affine polar form $f\colon \mathcal{P}^m \to \mathcal{E}$, such that*

$$f(r^i s^j t^k) = b_{i,\,j,\,k},$$

for all $(i,j,k) \in \Delta_m$. Furthermore, f is given by the expression

$$f(a_1,\ldots,a_m) =$$

$$\sum_{\substack{I\cup J\cup K=\{1,\ldots,m\}\\ I,J,K\ \text{pairwise disjoint}}} \left(\prod_{i\in I}\lambda_i\right)\left(\prod_{j\in J}\mu_j\right)\left(\prod_{k\in K}\nu_k\right) f(r^{|I|}s^{|J|}t^{|K|}),$$

where $a_i = \lambda_i r + \mu_i s + \nu_i t$, *with* $\lambda_i + \mu_i + \nu_i = 1$, *and* $1 \leq i \leq m$.

A point $F(a)$ *on the surface* F *can be expressed in terms of the Bernstein polynomials* $B^m_{i,j,k}(U, V, T) = \dfrac{m!}{i!j!k!}\, U^i V^j T^k$, *as*

$$F(a) = f(a^m) = \sum_{(i,\,j,\,k) \in \Delta_m} B^m_{i,j,k}(\lambda, \mu, \nu)\, f(r^i s^j t^k),$$

where $a = \lambda r + \mu s + \nu t$, *with* $\lambda + \mu + \nu = 1$.

Typically, the affine space $\mathcal{E}$ is the real affine space $\mathbb{A}^3$ of dimension three.

For example, with respect to the frame $\Delta rst = ((1, 0, 0), (0, 1, 0), (0, 0, 1))$, we obtain 10 control points for the Enneper surface, as illustrated in Table 2.2.

			$f(r,r,r)$ $\left(\frac{2}{3}, 0, 1\right)$			
		$f(r,r,t)$ $\left(\frac{2}{3}, 0, \frac{1}{3}\right)$		$f(r,r,s)$ $\left(\frac{2}{3}, \frac{2}{3}, \frac{1}{3}\right)$		
	$f(r,t,t)$ $\left(\frac{1}{3}, 0, 0\right)$		$f(r,s,t)$ $\left(\frac{1}{3}, \frac{1}{3}, 0\right)$		$f(r,s,s)$ $\left(\frac{2}{3}, \frac{2}{3}, -\frac{1}{3}\right)$	
$f(t,t,t)$ $(0,0,0)$		$f(s,t,t)$ $\left(0, \frac{1}{3}, 0\right)$		$f(s,s,t)$ $\left(0, \frac{2}{3}, -\frac{1}{3}\right)$		$f(s,s,s)$ $\left(0, \frac{2}{3}, -1\right)$

Table 2.2: Control points for Enneper's surface

A family $\mathcal{N} = (b_{i,\,j,\,k})_{(i,j,k)\in\Delta_m}$ of $\dfrac{(m+1)(m+2)}{2}$ points in $\mathcal{E}$ is called a *(triangular) control net, or Bézier net*. Note that the points in

$$\Delta_m = \{(i,\ j,\ k) \in \mathbb{N}^3 \mid i + j + k = m\},$$

can be thought of as a triangular grid of points in $\mathcal{P}$. For example, when $m = 5$, we have a grid of 21 points illlustrated in Table 2.3.

					500					
				401		410				
			302		311		320			
		203		212		221		230		
	104		113		122		131		140	
005		014		023		032		041		050

Table 2.3: Grid of control points for a quintic Bézier patch

We intentionally let i be the row index, starting from the left lower corner, and j be the column index, also starting from the left lower corner. The control net $\mathcal{N} = (b_{i,\,j,\,k})_{(i,j,k)\in\Delta_m}$ can be viewed as an image of the triangular grid Δ_m in the affine space $\mathcal{E}$. It follows from Lemma 2.7.1 that there is a bijection between polynomial surfaces of degree m and control nets $\mathcal{N} = (b_{i,\,j,\,k})_{(i,j,k)\in\Delta_m}$.

2.7.4 The de Casteljau algorithm and subdivision

In this section, we quickly review how the de Casteljau algorithm can be used to subdivide a triangular patch into three subpatches. For more details, see Farin [Far92, Far95], Hoschek and Lasser [HL93], Risler [Ris92], or Gallier [Gal00a]. There are also versions of the de Casteljau algorithm for rectangular patches, but we will not go into this topic. Again, readers are invited to consult Farin [Far92, Far95], Hoschek and Lasser [HL93], Risler [Ris92], or Gallier [Gal00a].

Given an affine frame Δrst, given a triangular control net $\mathcal{N} = (b_{i,\,j,\,k})_{(i,j,k)\in\Delta_m}$, recall that in terms of the polar form $f\colon \mathcal{P}^m \to \mathcal{E}$ of the polynomial surface $F\colon \mathcal{P} \to \mathcal{E}$ defined by $\mathcal{N}$, for every $(i,j,k) \in \Delta_m$, we have

$$b_{i,\,j,\,k} = f(r^r s^j t^k).$$

Given $a = \lambda r + \mu s + \nu t$ in $\mathcal{P}$, where $\lambda + \mu + \nu = 1$, to compute $F(a) = f(a,\ldots,a)$, the computation builds a sort of tetrahedron consisting of $m+1$ layers. The base layer consists of the original control points in $\mathcal{N}$, which are also denoted as $(b^0_{i,\,j,\,k})_{(i,j,k)\in\Delta_m}$. The other layers are computed in m stages, where at stage l, $1 \le l \le m$, the points $(b^l_{i,\,j,\,k})_{(i,j,k)\in\Delta_{m-l}}$ are computed such that

$$b^l_{i,\,j,\,k} = \lambda b^{l-1}_{i+1,\,j,\,k} + \mu b^{l-1}_{i,\,j+1,\,k} + \nu b^{l-1}_{i,\,j,\,k+1}.$$

During the last stage, the single point $b^m_{0,\,0,\,0}$ is computed. An easy induction shows that

$$b^l_{i,\,j,\,k} = f(a^l r^i s^j t^k),$$

where $(i,j,k) \in \Delta_{m-l}$, and thus, $F(a) = b^m_{0,\,0,\,0}$.

Assuming that a is not on one of the edges of Δrst, the crux of the subdivision method is that the three other faces of the tetrahedron of polar values $b^l_{i,\,j,\,k}$ besides the face corresponding to the original control net, yield three control nets

$$\mathcal{N}ast = (b^l_{0,\,j,\,k})_{(l,j,k) \in \in \Delta_m},\ \mathcal{N}rat = (b^l_{i,\,0,\,k})_{(i,l,k) \in \in \Delta_m},\ and\, \mathcal{N}rsa = (b^l_{i,\,j,\,0})_{(i,j,l) \in \in \Delta_m},$$

corresponding to the base triangle Δrsa, Δrat and Δrsa respectively. If a belongs to one of the edges, say rs, then the triangle Δrsa is flat, i.e. Δrsa is not an affine frame, and the net $\mathcal{N}rsa$ does not define the surface, but instead a curve. However, in such cases, the degenerate net $\mathcal{N}rsa$ is not needed anyway.

From an implementation point of view, we found it convenient to assume that a triangular net $\mathcal{N} = (b_{i,\,j,\,k})_{(i,j,k) \in \Delta_m}$ is represented as the list consisting of the concatenation of the $m + 1$ rows

$$b_{i,\,0,\,m-i},\ b_{i,\,1,\,m-i-1},\ \ldots,\ b_{i,\,m-i,\,0},$$

i.e.,

$$f(r^i t^{m-i}),\ f(r^i s t^{m-i-1}),\ \ldots,\ f(r^i s^{m-i-1} t),\ f(r^i s^{m-i}),$$

where $0 \leq i \leq m$. As a triangle, the net $\mathcal{N}$ is listed (from top-down) as

$$\begin{array}{c}
f(\underbrace{t,\ldots,t}_{m}) \quad f(\underbrace{t,\ldots,t}_{m-1},\, s) \quad \ldots \quad f(t,\, \underbrace{s,\ldots,s}_{m-1}) \quad f(\underbrace{s,\ldots,s}_{m}) \\
\ldots \qquad\qquad\qquad \ldots \\
\ldots \\
f(\underbrace{r,\ldots,r}_{m-1},\, t) \quad f(\underbrace{r,\ldots,r}_{m-1},\, s) \\
f(\underbrace{r,\ldots,r}_{m})
\end{array}$$

The main advantage of this representation is that we can view the net $\mathcal{N}$ as a two-dimensional array *net*, such that $net[i, j] = b_{i,j,k}$ (with $i + j + k = m$). In fact, only a triangular portion of this array is filled. This way of representing control nets fits well with the convention that the affine frame Δrst is represented as follows:

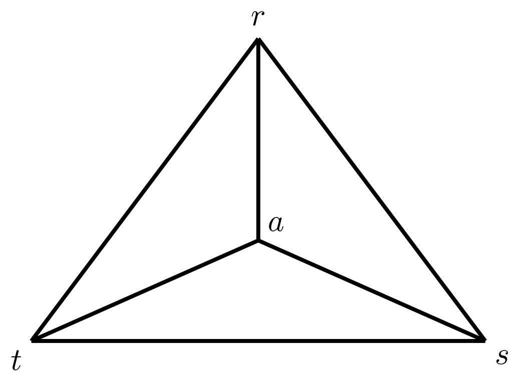


Figure 2.4: An affine frame

Instead of simply computing $F(a) = b^m_{0,0,0}$, the de Casteljau algorithm can be easily adapted to output the three nets $\mathcal{N}ast$, $\mathcal{N}rat$, and $\mathcal{N}rsa$.

Using the above version of the de Casteljau algorithm, it is possible to recursively subdivide a triangular patch. It would seem natural to subdivide Δrst into the three subtriangles Δars, Δast, and Δart, where $a = (1/3, 1/3, 1/3)$ is the center of gravity of the triangle Δrst, getting new control nets $\mathcal{N}ars$, $\mathcal{N}ast$ and $\mathcal{N}art$ using the functions described earlier, and repeat this process recursively. However, this process does not yield a good triangulation of the surface patch, because no progress is made on the edges rs, st, and tr, and thus, such a triangulation does not converge to the surface patch. Thus, to compute triangulations that converge to the surface patch, we need to subdivide the triangle Δrst in such a way that the edges of the affine frame are subdivided. There are many ways of performing such subdivisions, and we propose a method which has the advantage of yielding a very regular triangulation and of being very efficient.

The subdivision strategy that we propose is to divide the affine frame Δrst into four subtriangles Δabt, Δbac, Δcrb, and Δsca, where $a = (0, 1/2, 1/2)$, $b = (1/2, 0, 1/2)$, and $c = (1/2, 1/2, 0)$, are the middle points of the sides st, rt and rs respectively, as shown in the diagram below:

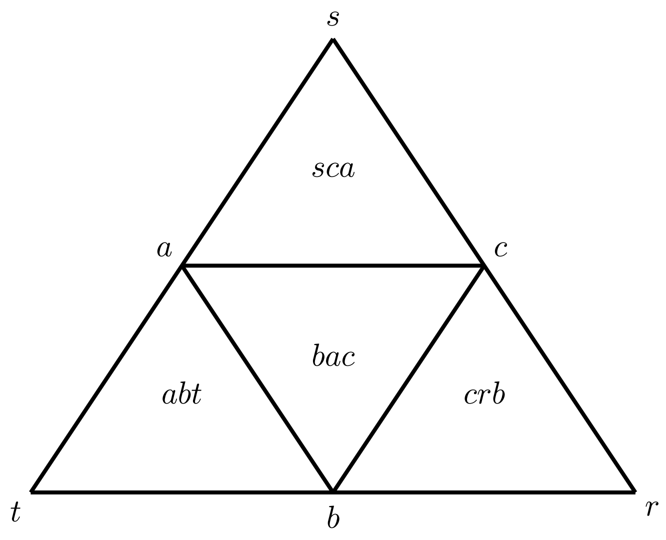


Figure 2.5: Subdividing an affine frame Δrst

It turns out that the four subpatches can be computed in four calls to the subdivision version of the de Casteljau algorithm. Details of such an algorithm can be found in Gallier [Gal00a], as well as subdivision algorithms for rectangular surfaces.

Example: For the following control net,

```
net = {{0, 0, 0}, {2, 0, 2}, {4, 0, 2}, {6, 0, 0},
        {1, 2, 2}, {3, 2, 5}, {5, 2, 2}, {2, 4, 2},
        {4, 4, 2},  {3, 6, 0}};
```

the surface patch associated with the affine frame (r, s, t) can be approximated using the subdivision version of the de Casteljau algorithm [Gal00a].

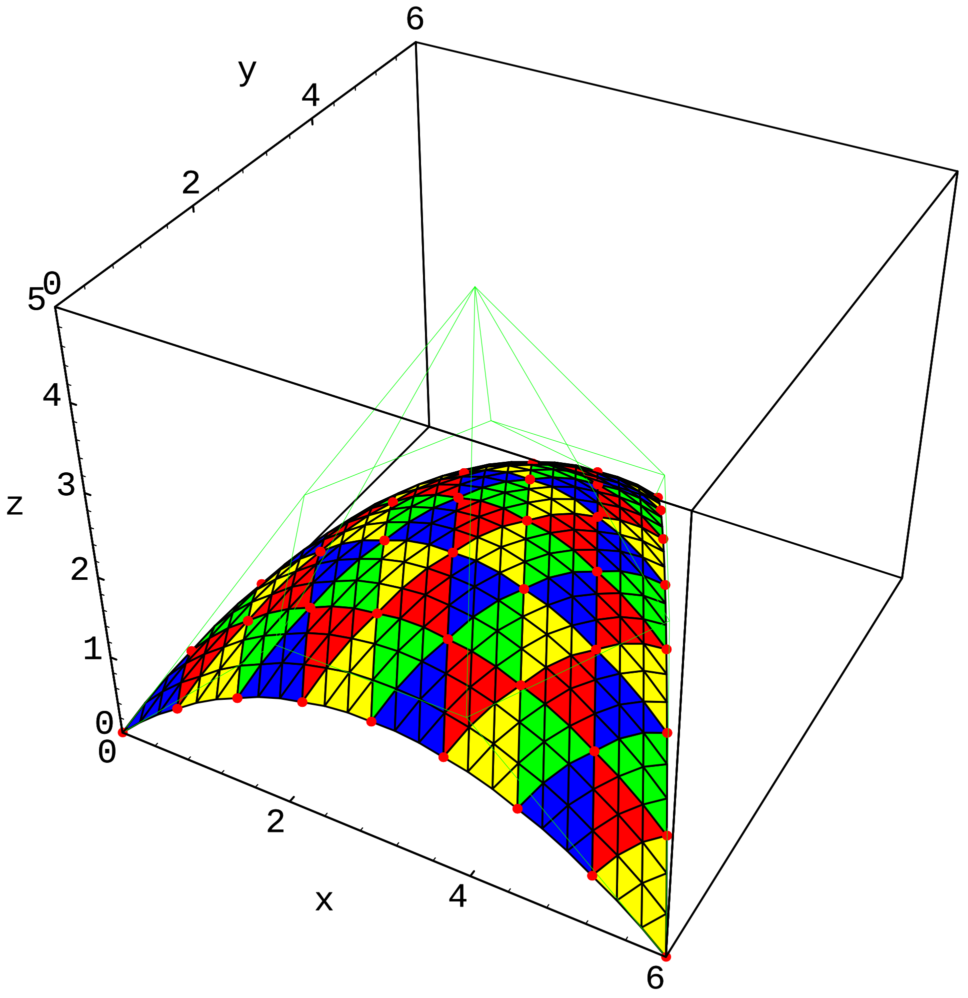


Figure 2.6: A triangular surface patch

Chapter 3

Continuity Conditions of Triangular Spline Surfaces

3.1 Continuity conditions on polar forms

The most important issue in joining polynomial surfaces is continuity along the boundaries. The question of exactly what we mean by "smooth" is of central importance. Today, the well-accepted standard is to look at *parametric continuity.* The parametric functions that define the surfaces are considered smooth if and only if their respective derivatives are well-defined up to some order.

Definition 3.1.1 Given two surface patches F and G, for any point $a \in \mathbb{A}^2$, F and G are said to join with C^k-continuity at a if and only if

$$F^{(i)}(a) = G^{(i)}(a),$$

for all i, $0 \leq i \leq k$. That is, their derivatives at a agree up to kth order.

We shall look at conditions forced on polar forms if two surfaces are to join with C^k-continuity. We focus our interest on spline surfaces based on a triangulation of the plane.

Definition 3.1.2 Let A and B be two adjacent convex polygons in the plane, and let (r, s) be the line segment along which they are adjacent (where $r, s \in \mathbb{A}^2$ are distinct vertices of A and B). Given two polynomial surfaces F_A and F_B of degree m, *F_A and F_B join with C^k continuity along the line segment (r, s)* iff F_A and F_B agree to kth order for all $a \in (r, s)$. That is, for any point a along the segment (r, s), all derivatives of F_A and F_B up to the kth order must agree at a.

Lemma 3.1.3 *For any $a \in (r, s)$, F_A and F_B agree to kth order at a iff their polar forms $f_A\colon (\mathbb{A}^2)^m \to \mathbb{A}^d$ and $f_B\colon (\mathbb{A}^2)^m \to \mathbb{A}^d$ agree on all multisets of points that contain at least $m - k$ copies of a, that is, iff*

$$f_A(u_1, \ldots, u_k, \underbrace{a, \ldots, a}_{m-k}) = f_B(u_1, \ldots, u_k, \underbrace{a, \ldots, a}_{m-k}),$$

for all $u_1, \ldots, u_k \in \mathbb{A}^2$.

Using the above lemma, we can prove the following crucial result:

Lemma 3.1.4 *Let A and B be two adjacent convex polygons in the plane, and let (r, s) be the line segment along which they are adjacent (where $r, s \in \mathbb{A}^2$ are distinct vertices of A and B). Given two polynomial surface F_A and F_B of degree m, F_A and F_B join with C^k continuity along (r, s) iff their polar forms $f_A\colon (\mathbb{A}^2)^m \to \mathbb{A}^d$ and*

$f_B\colon (\mathbb{A}^2)^m \to \mathbb{A}^d$ agree on all multisets of points that contain at least $m-k$ points on the line (r,s), that is, iff

$$f_A(u_1,\ldots,u_k,a_{k+1},\ldots,a_m) = f_B(u_1,\ldots,u_k,a_{k+1},\ldots,a_m),$$

for all $u_1,\ldots,u_k \in \mathbb{A}^2$, and all $a_{k+1},\ldots,a_m \in (r,s)$.

Proof. As Lemma 3.1.3 states, for every $a \in (r,s)$, F_A and F_B agree to kth order at a iff

$$f_A(u_1,\ldots,u_k,\underbrace{a,\ldots,a}_{m-k}) = f_B(u_1,\ldots,u_k,\underbrace{a,\ldots,a}_{m-k}),$$

for all $u_1,\ldots,u_k \in \mathbb{A}^2$. However, if we consider

$$a \mapsto f_A(u_1,\ldots,u_k,\underbrace{a,\ldots,a}_{m-k})$$

and

$$a \mapsto f_B(u_1,\ldots,u_k,\underbrace{a,\ldots,a}_{m-k})$$

as affine polynomial functions $F_A(u_1,\ldots,u_k)$ and $F_B(u_1,\ldots,u_k)$, if these functions agree on all points in (r,s), because of the uniqueness of the polar form associated with a polynomial function, the corresponding polar forms $f_A(u_1,\ldots,u_k)$ and $f_B(u_1,\ldots,u_k)$ agree for all points $a_{k+1},\ldots,a_m \in (r,s)$. Since this holds for all $u_1,\ldots,u_k \in \mathcal{P}$, we have shown that

$$f_A(u_1,\ldots,u_k,a_{k+1},\ldots,a_m) = f_B(u_1,\ldots,u_k,a_{k+1},\ldots,a_m),$$

for all $u_1,\ldots,u_k \in \mathbb{A}^2$, and all $a_{k+1},\ldots,a_m \in (r,s)$, as desired. □

3.2 Necessary and sufficient polar form conditions for C^1 continuity

As a consequence of Lemma 3.1.4, we obtain the necessary and sufficient conditions on triangular control nets for two surface patches F_A and F_B of degree m to join with C^n continuity along (r, s).

Let $A = \Delta prs$ and $B = \Delta qrs$ be two frames in the plane, sharing the edge (r, s).

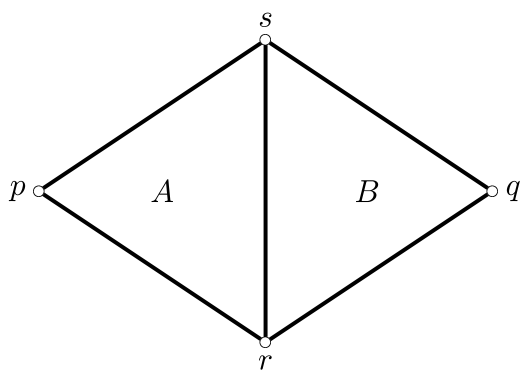


Figure 3.1: Two adjacent reference triangles

Then, Lemma 3.1.4 tells us that F_A and F_B join with C^k continuity along (r, s) iff

$$f_A(p^g q^h r^i s^j) = f_B(p^g q^h r^i s^j),$$

for all g, h, i, j such that $g + h + i + j = m$, and $i + j \geq m - k$ $(0 \leq k \leq m)$.

For C^0 continuity, we have $k = 0$. Thus, $i + j \geq m - 0 \Rightarrow i + j \geq m$, since

$g + h + i + j = m$, we have $g = h = 0$. Therefore, we just have

$$f_A(r^i s^{m-i}) = f_B(r^i s^{m-i}),$$

with $0 \leq i \leq m$, which means that the control points of the boundary curves along (r, s) must agree.

For C^1 continuity, we have $k = 1$. Thus, $i + j \geq m - 1$. Since $g + h + i + j = m$, we have three cases. Either $i + j = m$, then $g = h = 0$, or $i + j = m - 1$, in which case either $g = 0$ and $h = 1$, or $g = 1$ and $h = 0$. Thus, we have the conditions

$$f_A(r^i s^{m-i}) = f_B(r^i s^{m-i}),\ 0 \leq i \leq m, \tag{3.1}$$

$$f_A(pr^i s^{m-i-1}) = f_B(pr^i s^{m-i-1}),\ 0 \leq i \leq m-1, \tag{3.2}$$

$$f_A(qr^i s^{m-i-1}) = f_B(qr^i s^{m-i-1}),\ 0 \leq i \leq m-1. \tag{3.3}$$

This is a total of $3m + 1$ conditions. However, we will now show that in general, only $2m + 1$ of these conditions are independent. Let $q = \lambda p + \mu r + \nu s$, where $\lambda + \mu + \nu = 1$. Since we are assuming that A and B are distinct triangles, $\lambda \neq 0$, and

$$p = \frac{1}{\lambda} q - \frac{\mu}{\lambda} r - \frac{\nu}{\lambda} s,$$

and Condition 3.2

$$f_A(pr^i s^{m-i-1}) = f_B(pr^i s^{m-i-1})$$

becomes

$$f_A(pr^i s^{m-i-1}) = f_B\left(\left(\frac{1}{\lambda} q - \frac{\mu}{\lambda} r - \frac{\nu}{\lambda} s\right) r^i s^{m-i-1}\right),$$

and since f_B is multiaffine, this yields

$$f_A(pr^is^{m-i-1}) = \frac{1}{\lambda} f_B(qr^is^{m-i-1}) - \frac{\mu}{\lambda} f_B(r^{i+1}s^{m-i-1}) - \frac{\nu}{\lambda} f_B(r^is^{m-i}),$$

or equivalently

$$\lambda f_A(pr^is^{m-i-1}) + \mu f_B(r^{i+1}s^{m-i-1}) + \nu f_B(r^is^{m-i}) = f_B(qr^is^{m-i-1}),$$

for $0 \leq i \leq m-1$.

Now using Condition 3.1

$$f_A(r^is^{m-i}) = f_B(r^is^{m-i}),$$

we get

$$(*) \qquad \lambda f_A(pr^is^{m-i-1}) + \mu f_A(r^{i+1}s^{m-i-1}) + \nu f_A(r^is^{m-i}) = f_B(qr^is^{m-i-1}),$$

for $0 \leq i \leq m-1$. Similarly, since $q = \lambda p + \mu r + \nu s$, from Condition 3.3

$$f_A(qr^is^{m-i-1}) = f_B(qr^is^{m-i-1}),$$

we get

$$f_A((\lambda p + \mu r + \nu s)r^is^{m-i-1}) = f_B(qr^is^{m-i-1}),$$

which yields

$$\lambda f_A(pr^is^{m-i-1}) + \mu f_A(r^{i+1}s^{m-i-1}) + \nu f_A(r^is^{m-i}) = f_B(qr^is^{m-i-1}),$$

for $0 \leq i \leq m-1$. These are the conditions $(*)$ that we found earlier, and thus, the conditions for C^1-continuity are indeed the $2m+1$ conditions

$$f_A(r^is^{m-i}) \;=\; f_B(r^is^{m-i}),\ 0 \leq i \leq m, \tag{3.4}$$

$$f_B(qr^is^{m-i-1}) \quad = \quad \lambda f_A(pr^is^{m-i-1}) + \mu f_A(r^{i+1}s^{m-i-1}) + \nu f_A(r^is^{m-i}) \qquad (3.5)$$

$0 \le i \le m-1,$ *with* $q = \lambda p + \mu r + \nu s.$

These conditions show that the control points $f_A(pr^is^{m-i-1})$, $f_A(r^{i+1}s^{m-i-1})$, $f_A(r^is^{m-i})$ and $f_B(qr^is^{m-i-1})$ satisfy the same affine relation that p, r, s, q satisfy, i.e., the diamond formed by these control points is an affine image of the diamond (p, r, s, q) under some unique affine map h.

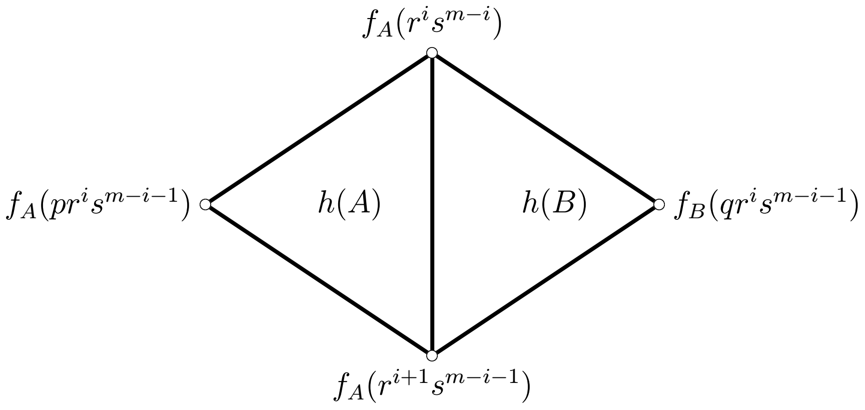


Figure 3.2: C^1-continuity conditions

In the special case where $q = r + s - p$, or equivalently $p + q = r + s$, which means that the line segments (p, q) and (r, s) have the same midpoint, the above Conditions 3.5 become

$$\frac{1}{2}(f_A(pr^is^{m-i-1}) + f_B(qr^is^{m-i-1})) = \frac{1}{2}(f_A(r^{i+1}s^{m-i-1}) + f_A(r^is^{m-i})), \qquad (3.6)$$

$0 \le i \le m-1.$

Now we will look at C^1 continuity constraints in more detail. We start from the special case where adjacent triangles form a parallelogram. This is true if the all triangles in the parameter plane are equilateral.

3.3 C^1 continuity constraints – a closer look

3.3.1 Around a vertex

Assume that we have a (finite) triangulation of the parameter plane $\mathbb{A}^2$ consisting of equilateral triangles. We know that to achieve C^1 continuity, the line segment between the apexes of any two triangular patches sharing an edge must have the same midpoint as the edge itself, as stated by equation 3.6. Let us take a closer look at what this means when three or more adjacent triangular patches have a common vertex. First of all, we introduce a definition that will simplify our notations somewhat.

Definition 3.3.1 Given $n \geq 2$ triangular patches of degree m that share a common vertex v_1, let $T_1, \ldots, T_n$ be the n triangles in the planar domain and $f_1, \ldots, f_n$ be the n polar forms associated with the patches. Let $v_1, \ldots, v_{n+2}$ denote the vertices of the domain triangles, so that $T_i = (v_1, v_{i+1}, v_{i+2})$, with $1 \leq i \leq n$. Let $z_1 = f_i({v_1}^m)$, then the control points $z_{i+1} = f_i({v_1}^{m-1}, v_{i+1})$, *with* $1 \leq i \leq n+1$, are called the *star of control points* around z_1, or simply, the *star* of z_1 (See Figure 3.3).

We call the set $\{z_2, \ldots, z_{n+2}\}$ a *complete star* if z_1 is completely surrounded by patches. A complete star has n control points. An incomplete star has $n+1$ control points.

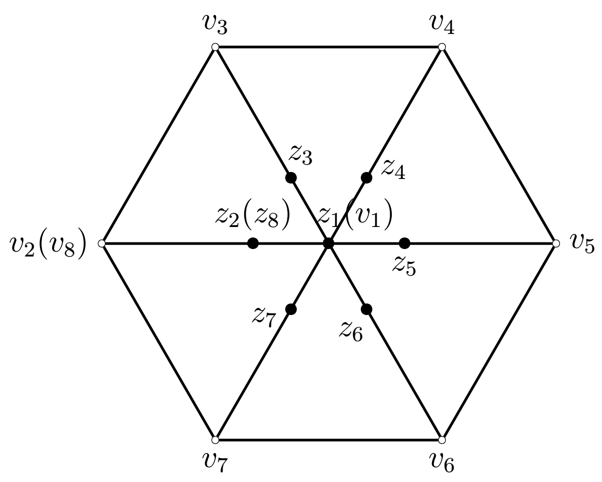


Figure 3.3: $z_2, \ldots, z_7$ is the complete star of z_1

Notice that vertex v_{n+2} coincides with v_2 if the star is complete.

We should also point out the v_i are in the planar domain and the z_i are control points in space. However, the corner point and its star, i.e. the z_i, are indeed coplanar and their convex hull is an affine image of the convex hull given by the v_i. One can think of Figure 3.3 as having the z_i superimposed onto the domain triangulation given by the v_i. Also, recall that the notation $f_i({v_j}^m)$ indicates that there are m copies of v_j in f_i's arguments, that is:

$$f_i({v_j}^m) = f_i(\underbrace{v_j, \ldots, v_j}_{m}).$$

The star completely determines the continuity conditions around a vertex, and therefore we will look at it closely. The necessary and sufficient polar form Conditions 3.6 given in Section 3.2 for C^1 continuity apply, as we have an equilateral template triangulation.

To simplify the notation for the equations given by Conditions 3.6, we would like to drop $\frac{1}{2}$. It turns out that it is all right to do so, because there is a contruction that will embed an affine space in a vector space. Thus, affine points can be turned into vectors, and then back to points. For more information on such a "homogenizing" construction, please refer to Chapter 4 of Gallier [Gal00b], or Berger [Ber90]. We begin with three patches sharing a common vertex.

Case 1: Three triangular patches.

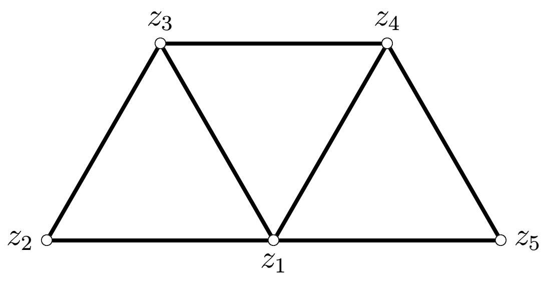


Figure 3.4: Three adjacent triangular patches

The two C^1-equations are

$$z_1 + z_3 = z_2 + z_4,$$

$$z_3 + z_5 = z_1 + z_4.$$

Subtracting the second equation from the first, we get $z_1 - z_5 = z_2 - z_1$, or

$$z_2 + z_5 = 2z_1.$$

Thus, z_1 is the middle of the line segment (z_2, z_5). There are three degrees of freedom, for example, z_1, z_3, z_4.

To avoid redundancy, we will skip over the next two cases, e.g. four and five triangular patches, and go straight to six patches.

Case 2: Six triangular patches.

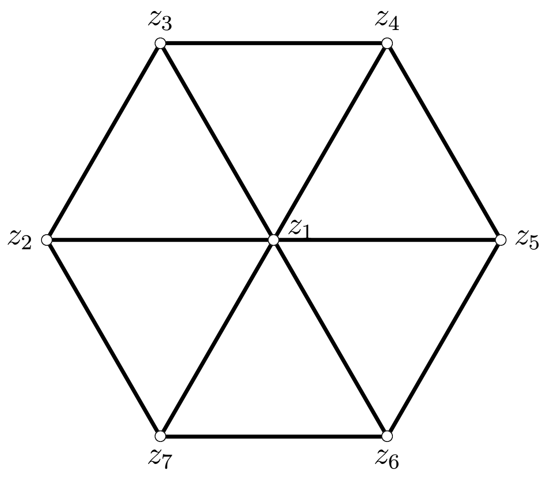


Figure 3.5: Six adjacent triangular patches

There are two more equations, and the six C^1-equations are

$$z_1 + z_3 = z_2 + z_4,$$

$$z_3 + z_5 = z_1 + z_4,$$

$$z_1 + z_5 = z_4 + z_6,$$

$$z_5 + z_7 = z_1 + z_6,$$

$$z_2 + z_1 = z_3 + z_7,$$

$$z_2 + z_6 = z_1 + z_7.$$

Remarkably, the last two equations follow from the first four. Indeed, as in the previous case, we get

$$z_2 + z_5 = 2z_1,$$

$$z_3 + z_6 = 2z_1,$$

$$z_4 + z_7 = 2z_1,$$

and by adding the two equations

$$z_1 + z_3 = z_2 + z_4,$$

$$z_4 + z_7 = 2z_1,$$

we get

$$z_1 + z_3 + z_4 + z_7 = 2z_1 + z_2 + z_4$$

which reduces to

$$z_3 + z_7 = z_1 + z_2,$$

that is,

$$z_2 + z_1 = z_3 + z_7.$$

Similarly, by adding the two equations

$$z_2 + z_1 = z_3 + z_7,$$

$$z_3 + z_6 = 2z_1,$$

we get

$$z_2 + z_1 + z_3 + z_6 = 2z_1 + z_3 + z_7$$

which reduces to

$$z_2 + z_6 = z_1 + z_7.$$

Thus, the rank of the system is four, and it is easily seen that the equations

$$z_2 + z_5 = 2z_1,$$

$$z_3 + z_6 = 2z_1,$$

$$z_4 + z_7 = 2z_1,$$

$$z_2 + z_4 + z_6 = 3z_1,$$

are linearly independent. There are three degrees of freedom, for example, z_2, z_4, z_6 (or z_3, z_5, z_7).

In all cases, there are three degrees of freedom among the vertex and its star, that is, any three control points that are not collinear completely determine the continuity condition around that vertex.

3.3.2 Along an edge

When two degree m patches meet, the C^1 continuity constraints along the common edge are completely determined by the parallelograms formed by the control points across the common edge. Figure 3.6 shows two patches of degree four, which contains three such parallelograms. In general, there are $m - 1$ such parallelograms for any two degree m patches that share an edge.

We must have

$$v_A^k = v_B^k$$

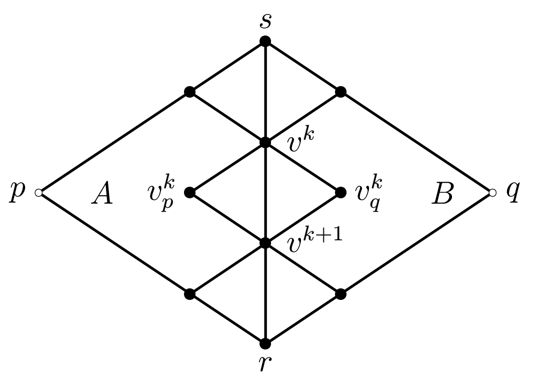


Figure 3.6: Two cubic patches that share an edge

along the common edge (r, s) and every diamond $(v_p^k, v^k, v_q^k, v^{k+1})$ must be the image of (p, s, q, r) under a bijective affine map, as shown in Figure 3.2. We can pick any three non-collinear points among (p, q, r, s). Assuming (p, r, q) are not collinear and $s = \lambda p + \mu r + \nu q$, we get

$$v^k = \lambda v_p^k + \mu v^{k+1} + \nu v_q^k.$$

as shown in Conditions 3.5 in Chapter 3.

Similarly, in the special case where (p, s, q, r) is a parallelogram, Conditions 3.5 become Conditions 3.6, that is, the line segments (p, q) and (r, s) have the same midpoint, and we get

$$\frac{1}{2}(v^k + v^{k+1}) = \frac{1}{2}(v_p^k + v_q^k),$$

written simply as:

$$v^k + v^{k+1} = v_p^k + v_q^k.$$

Thus, any three non-collinear control points among the four control points that make up a parallelogram will determine the fourth.

3.3.3 Arbitrary triangulation

So far, we have seen that with a equilateral triangulation, there are three degrees of freedom around a vertex control point and its star, and also in any parallelogram across any shared edge. This is a nice result, but in obtaining the result, we relied on diagonals of the parallelograms having the same midpoint. This is no longer true when we have an arbitrary triangulation.

Much to our delight, the case of an arbitrary triangulation is not much more difficult. It turns out that we still have three degrees of freedom among the vertex and its star, regardless of how many patches come together at that vertex.

Recall Conditions 3.5 given in Section 3.2 for the general case:

$$f_B(qr^is^{m-i-1}) = \lambda f_A(pr^is^{m-i-1}) + \mu f_A(r^{i+1}s^{m-i-1}) + \nu f_A(r^is^{m-i}),$$

where $\lambda + \mu + \nu = 1$.

These conditions show that the control points $f_A(pr^is^{m-i-1})$, $f_A(r^{i+1}s^{m-i-1})$, $f_A(r^is^{m-i})$, and $f_B(qr^is^{m-i-1})$, satisfy the same affine relation that p, r, s, q satisfy, i.e., the diamond formed by these control points is an affine image of the diamond (p, r, s, q) under some affine map h.

Lemma 3.3.2 *Given $n \geq 2$ triangular patches that share a common vertex, let $T_1, \ldots, T_n$ be the $n \geq 2$ triangles in the planar domain associated with the patches, with $T_i = (v_{i+1}, v_1, v_{i+2})$, so that T_i and T_{i+1} share the vertex v_1 and are adjacent along the edge (v_1, v_i) ($3 \leq i \leq n+1$). Let z_1 be the control point associated to v_1, and $z_2, \ldots, z_{n+2}$ the star of control points of z_1 (note that $v_{n+2} = v_2$ and $z_{n+2} = z_2$*

if the star is complete). Also let $F_1, \ldots, F_n$ be the polynomial maps associated with the patches. If the patches $F_i(T_i)$ and $F_{i+1}(T_{i+1})$ meet with C^1 continuity, for all $1 \leq i \leq n$, then there is a unique affine map $h\colon \mathbb{A}^2 \to \mathbb{A}^3$ such that

$$h_i(v_i) = z_i,$$

with $1 \leq i \leq n+2$.

As a consequence, if (v_i, v_j, v_k) are any three affinely independent points, for any v_l with $l \neq i, j, k$, if $v_l = \lambda v_i + \mu v_j + \nu v_k$, we also have the equation

$$z_l = \lambda z_i + \mu z_j + \nu z_k,$$

where $\lambda + \mu + \nu = 1$.

Proof. We proceed with induction on n. Base case: $n = 2$. There are two triangles, $T_1 = (v_2, v_1, v_3)$, and $T_2 = (v_3, v_1, v_4)$ that meet with C^1 continuity. According to Conditions 3.5 obtained in Chapter 3, the points z_4, z_1, z_2 and z_3 are an image of v_4, v_1, v_2 and v_3 under a unique affine map.

Assume that Lemma 3.3.2 holds for n. Now, we prove it for $n+1$. According to the induction hypothesis, patches $F_i(T_i)$ and $F_{i+1}(T_{i+1})$ meet with C^1 continuity for all $1 \leq i \leq n$, thus there is a unique affine map h such that

$$h_i(v_i) = z_i,$$

with $1 \leq i \leq n+2$. Now, we have a new patch $F_{n+1}(T_{n+1})$ that meets with patch $F_n(T_n)$ with C^1 along (v_1, v_{n+2}). According to Conditions 3.5, there is also an unique affine map g_i such that $g_i(v_i) = z_i$, with $i \in [1, n+1, n+2, n+3]$. Thus we have the affine maps h_i and g_i overlap on the triangle $T_n = (v_1, v_{n+1}, v_{n+2})$. Because the triangle T_n is a proper triangle with three affinely independent vertices which form an affine frame, this forces h_i and g_i to be the same affine map. $\square$

Since any three affinely independent points form an affine basis, any other point is completely determined by a unique affine combination of these three points. When any three non-collinear control points among z_1 and its star are determined, to compute any other control point in the star, we can immediately find the corresponding vertices of the four control points involved, compute the barycentric coefficients for the unique affine combination between the vertices, and use the same coefficients to compute the fourth control point. Thus, regardless of how many patches come together, any three non-collinear control points among z_1 and its star determine the continuity conditions around a vertex completely. The same also holds for similar quadrilaterals along the edges.

Chapter 4

An Algorithm for Designing C^1 Triangular Spline Surfaces

4.1 Algorithm to choose prescribed (free) control points based on equilateral triangulations

The challenge is to come up with a way to systematically prescribe a certain set of control points, so that the rest of the control points can be computed efficiently, and the resulting surface has guaranteed C^1 continuity. We restrict ourselves for now to the special case where all triangles in the template triangulation are equilateral. This guarantees that the two lines formed by the opposite apexes of any two adjacent triangles (line segments (p, q) and (r, s)) have the same midpoint. Notice that restricting all triangles to be isosceles, with the third, non-equilateral edge as the shared common edge between two triangles achieves the same effect, but the condition makes the template triangulation a lot less regular, and harder to draw.

4.1.1 Degree 5 patches

It is generally impossible to construct a triangular spline surface of degree m with C^k continuity, if $2m \leq 3k + 1$. The proof of this is quite involved, and a detailed account can be found on pages 317−320 in Gallier's *Curves and Surfaces in Geometric Modeling* [Gal00b].

Thus, we must have $2m \geq 3k+2$. For C^1 continuity, we have $k = 1$, and therefore m must be at least three. Because there are three degrees of freedom in all cases where adjacent triangles share some common vertex, the systems of equations to solve for control points are over-determined for $m \leq 4$. It turns out that in order to have local flexibility and a reasonably symmetric method, $m = 5$ is the smallest possible degree to allow enough freedom. Therefore we turn our attention to quintic patches.

Since we have degree five, each patch has 21 control points. We divide these control points into corner points, edge points and inner points. Recall that the control points

$$b_{i,j,k} = f(\underbrace{r, \ldots, r}_{i}, \underbrace{s, \ldots, s}_{j}, \underbrace{t, \ldots, t}_{t})$$

where $i + j + k = m$ can be viewed as an image of the triangular grid Δ_m defined by

$$\Delta_m = \{(i, j, k) \in \mathbf{N}^3 \quad | i + j + k = m\}.$$

As shown in Table 2.3, the 21 control points of a quintic patch forms a triangular grid. We repeat the table here for the reader's convenience.

					500					
				401		410				
			302		311		320			
		203		211		221		230		
	104		113		122		131		140	
005		014		023		032		041		050

Table 4.1: Control points grid for a quintic triangular patch

Definition 4.1.1 Given a triangular control net of degree m,

1. *Corner points* are control points whose corresponding polar values have either $i = m$, $j = m$ or $k = m$, which means any two of i, j, k are 0.

2. *Edge points* are control points whose corresponding polar values have either $i + j = m$, $j + k = m$ or $i + k = m$, which means any one of i, j, k is 0.

3. *Inner points* are control points whose corresponding polar values have $i+j+k = m$ with $i \neq 0, j \neq 0, k \neq 0$.

When $m = 5$, the corner points are $b_{0,0,5}$, $b_{5,0,5}$ and $b_{5,0,0}$, the edge points are $b_{4,0,1}$, $b_{3,0,2}$, $b_{2,0,3}$, $b_{1,0,4}$, $b_{0,1,4}$, $b_{0,2,3}$, $b_{0,3,2}$, $b_{0,4,1}$, $b_{1,4,0}$, $b_{2,3,0}$, $b_{3,2,0}$ and $b_{4,1,0}$ and the inner points are $b_{3,1,1}$, $b_{1,3,1}$, $b_{1,1,3}$, $b_{2,1,2}$, $b_{2,2,1}$, $b_{1,2,2}$. It is clear that the corner points correspond to points on the three corners of the triangular surface patch, the edge points correspond to points on the three edges of the triangular surface patch, and the inner points correspond to points not on the boundary of the triangular surface patch.

4.1.2 Corner point region

First of all, we need to specify a uniform way to prescribe the corner points. As we have seen earlier in Chapter 3, the continuity conditions around a corner point depends on the number of patches (or the number of edges) coming together at that particular point, although there are three degrees of freedom in all cases.

In fact, a corner point that is adjacent to exactly two edges is common to only one face, and therefore does not contribute to or constrain the continuity conditions of the patch. Such points and its two edges are commonly called an “ear” in many literatures on triangulation, and we will refer to such corner points and edge points on the two edges associated as *ear points*.

Because we are dealing with equilateral triangles, there can be at most six patches coming together at a corner point. Thus, we need only to specify prescription methods for corner points adjacent to two to six patches. They are shown in Figure 4.1, where a square represents a corner point, a circle represents an edge point and a green square or circle indicates that particular control point is prescribed.

In theory, there are three degrees of freedom in all of the above cases, and therefore we can pick any three control points that are not collinear. We chose the above schemes simply because they are more symmetric, and therefore are less likely to cause undesired compensation effect due to excessive tilting of the affine reference frame. In the case of a corner point adjacent to six faces, either of the two schemes is permitted. We simply pick any three alternate points out of the six adjacent to the corner point.

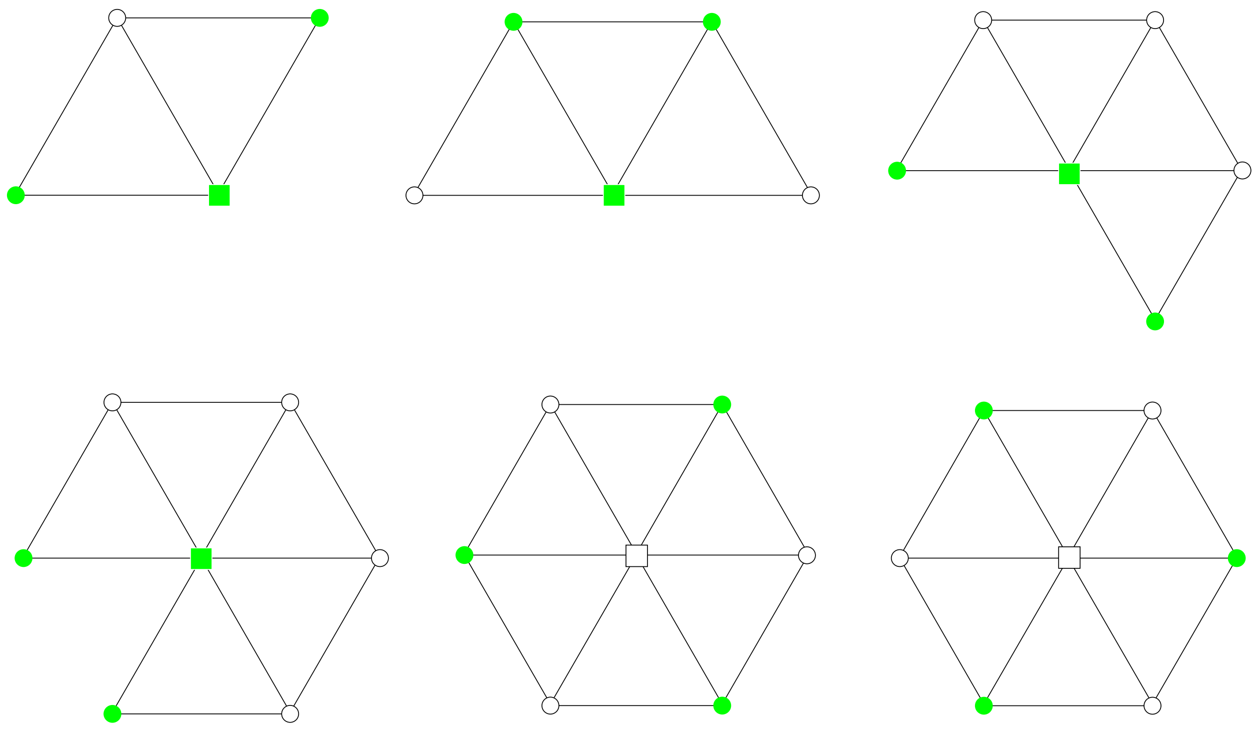


Figure 4.1: Prescribing corner points

Once a corner point region is prescribed, propagation will determine the corner point and its star completely. Corner point region propagations are computed based on the fact that the midpoints of the two diagonals of the parallelograms must agree. Because of the regularity of the equilateral triangles, the corner point is the midpoint of three pairs of star points (in the case of six patches), and the center of gravity of any three alternate star points. These special properties simplify propagation somewhat and we take advantage of that. Propagation computations are simple. They consist of only one addition and one subtraction in most cases, and are instantaneous.

As shown in Figure 4.2, where a corner point region consisting of z_1 and its complete star $z_2, \ldots, z_7$ is given. We notice that z_2, z_4 and z_6 are prescribed. We first

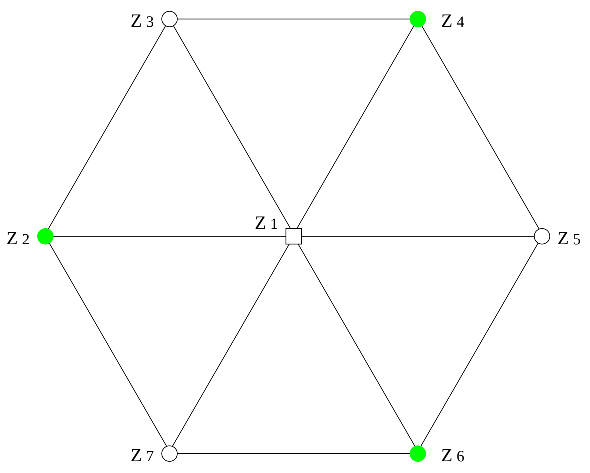


Figure 4.2: Prescribing corner points, example

compute z_1. Since it is the center of gravity of z_2, z_4 and z_6,

$$z_1 = \frac{z_2 + z_4 + z_6}{3}.$$

Once z_1 is obtained, z_3, z_5 and z_7 follow immediately:

$$z_3 = 2z_1 - z_6,$$

$$z_5 = 2z_1 - z_2$$

$$z_7 = 2z_1 - z_4,$$

z_1 is obtained with two additions and one division, and the rest one addition ($2z_1 = z_1 + z_1$) and one subtraction.

4.1.3 Along the edges

What remains are continuity conditions along an edge, which are controlled by the parallelograms formed by inner points and edge points across that edge. Clearly all edge points directly affect the continuity conditions across that particular edge, but for every patch, only some of its inner points affect its C^1 continuity conditions. These are the inner points that form the aforementioned parallelograms.

For every two patches that share an edge, once the two corner point regions of the shared edge are computed, we need only to prescribe two inner points per patch, plus one inner point per shared edge to determine all the edge points.

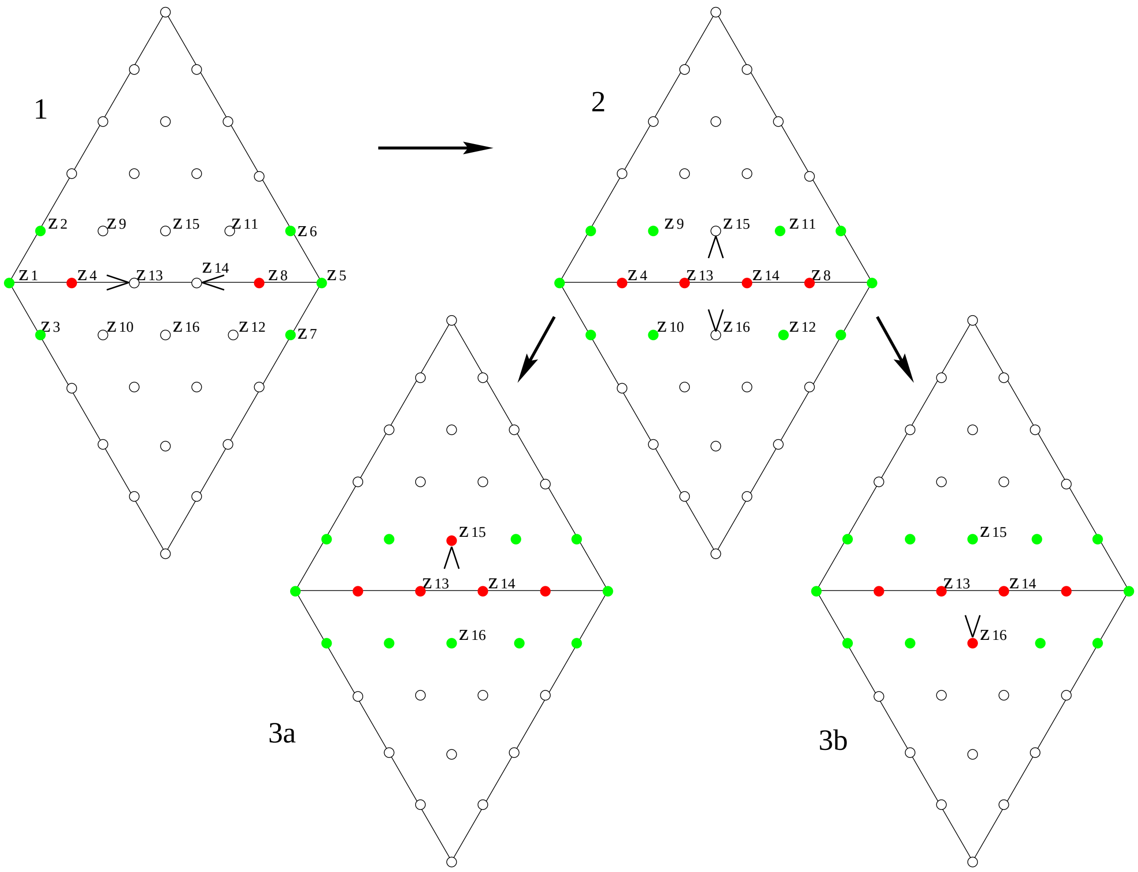


Figure 4.3: Prescriptions and propagations along the edges

Edge propagations are computed the same way as corner point region propagations. We force propagations along the shared edge from both ends to the center as shown in step $1-2$. Step 1 shows an edge whose two endpoint corner-point-region propagations $(z_1, z_2, z_3 \rightarrow z_4)$, $(z_5, z_6, z_7 \rightarrow z_8)$ are already completed. We adopt the convention of drawing prescribed control points in green, and propagated control points in red.

Next, following the directions of propagation as indicated by the two arrows, we prescribe two inner points at each side of the edge, namely, (z_9, z_{10}) and (z_{11}, z_{12}). Edge points z_{13} and z_{14} are then immediately computed via propagation in the following way:

$$z_{13} = z_9 + z_{10} - z_4,$$

$$z_{14} = z_{11} + z_{12} - z_8$$

At this stage we have z_{15} and z_{16} still to be propagated. It's clear that one of them needs to be prescribed, and the other propagated. Step 3_a and 3_b are both legal moves allowed by the algorithm. Prescribing either one of the inner points will lock the other in. We therefore call them *flip points*. The flip point is propagated either as $z_{15} = z_{13} + z_{14} - z_{16}$, or $z_{16} = z_{13} + z_{14} - z_{15}$, depending on which one we choose to prescribe. The two directions of propagations are caught and stopped by the flip points in the middle of the shared edge. All control points remaining (only inner and ear points) are free and they do not constrain continuity conditions.

4.1.4 Six patches

When we have a large surface with many patches, the case where six patches share a common corner point is typical except on the boundaries. In this case, three inner points per patch plus one flip point every two adjacent patches will suffice to determine continuity conditions across the patches. We require that corner point region propagations always precede edge propagations. All propagations are computed based on diagonal mid-point agreement of affected parallelograms.

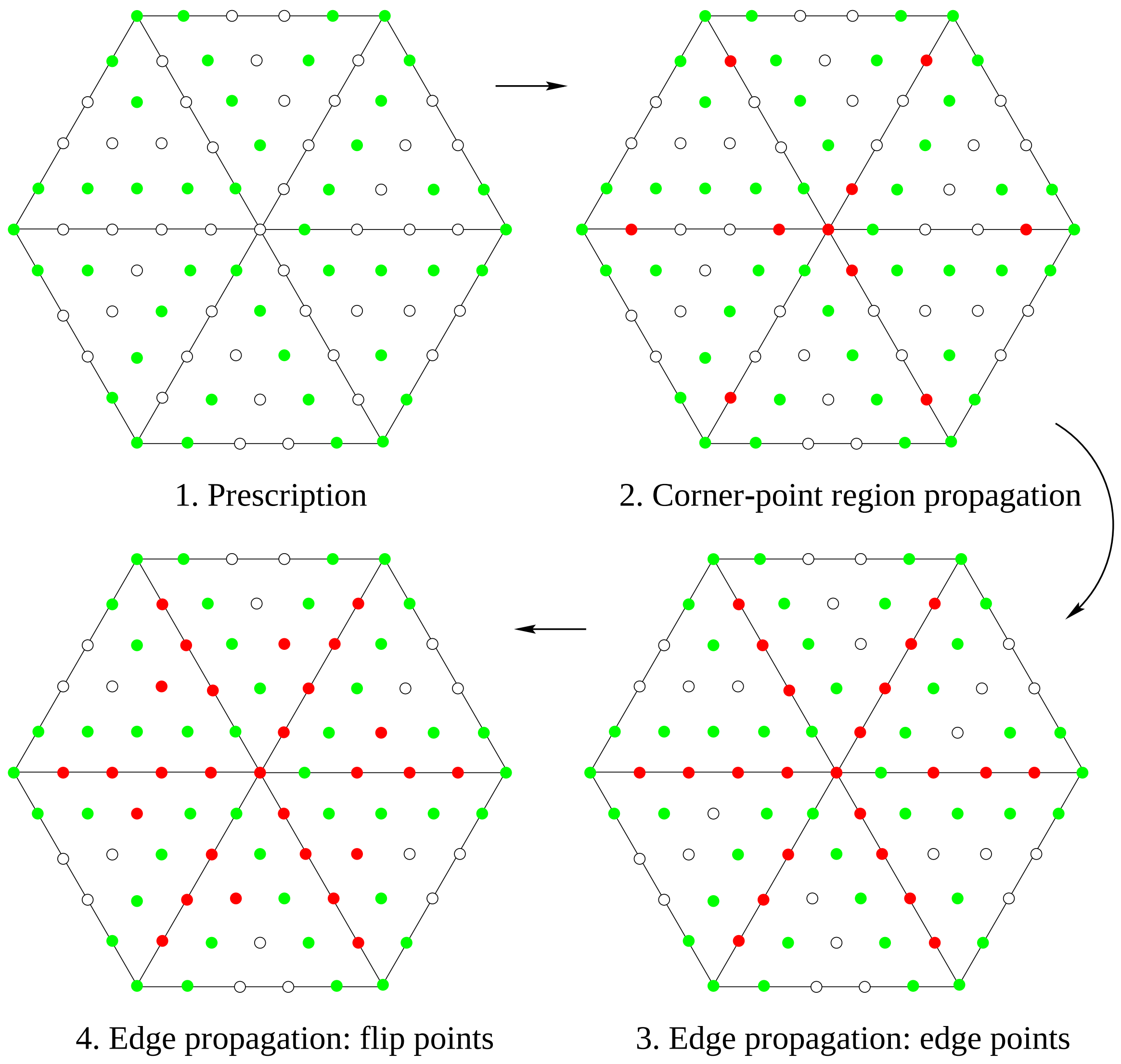

1. Prescription
2. Corner-point region propagation
3. Edge propagation: edge points
4. Edge propagation: flip points

Figure 4.4: When six patches come together

4.2 Irregular triangulations

So far, we have based our algorithm on equilateral triangulations, which are quite restrictive. The kind of surfaces they can represent tend to be geometrically regular and are not very realistic. We started with equilateral triangulations due to the nice properties brought by their regularity, but as we shall see, the properties of the equilateral triangles are not necessary, and our algorithm needs minimal modifications to adapt to irregular triangulations. In fact, equilateral triangulation is just a special case of irregular triangulation.

First, let's look at corner point region. From Lemma 3.3.2, we know that around the corner point region, there are always three degrees of freedom regardless how many patches come together. Therefore any three non-collinear control points among the corner point and its star can be prescribed. The three prescribed control points then form an affine frame. Therefore, propagation for any fourth point in this corner region is computed based on affine relations with respect to that particular frame.

Along the edges, we have similar quadrilaterals instead of parallelograms because the triangulation is no longer regular. Propagations still proceed in the same manner and directions, and involve control points with the same positions (or polar values) in their respective patches. The only thing that is different is in the actual computation. Just like in the corner point region, instead of mid-point calculations, we compute using affine relations.

The complexity of our algorithm stays the same, because the only thing that is different in the irregular case is the actual computation step of the propagation.

The exact same sets of control points are prescribed and propagated, respectively. The computation based on affine relations is only slightly more complicated, which requires the computation of 2×2 determinants.

In Figure 4.5, following our convention, all the colored points, some labeled z_i, are control points in space, and the v_i belong to the domain triangulation, and are therefore in the plane.

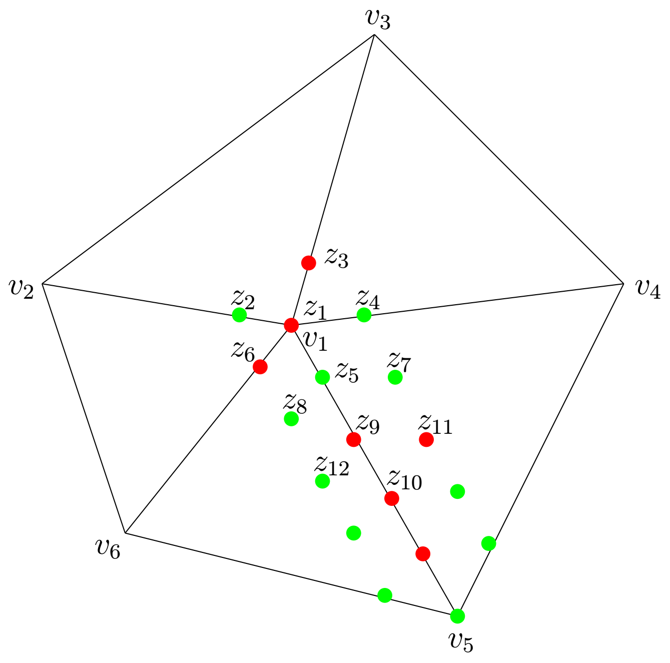


Figure 4.5: Irregular triangulation

In general, it is easily shown that given three affinely independent point (a, b, c) in $\mathbb{A}^2$, for any point $x \in \mathbb{A}^2$, if $x = \lambda a + \mu b + \nu c$, where λ, μ, ν are the barycentric coordinates of x with respect to (a, b, c),

$$\lambda = \frac{det(\overrightarrow{xb}, \overrightarrow{bc})}{det(\overrightarrow{ab}, \overrightarrow{ac})}, \quad \mu = \frac{det(\overrightarrow{ax}, \overrightarrow{ac})}{det(\overrightarrow{ab}, \overrightarrow{ac})}, \quad \nu = \frac{det(\overrightarrow{ab}, \overrightarrow{ax})}{det(\overrightarrow{ab}, \overrightarrow{ac})}.$$

As shown in the above example, the corner point region of z_1 and its star $z_2, \ldots, z_6$ has the three points (z_2, z_4, z_5) chosen as the affine frame. Computations for the rest of the control points in the corner point region is given as follows:

$$z_1 = \frac{det(\overrightarrow{v_1v_4}, \overrightarrow{v_4v_5})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_2 + \frac{det(\overrightarrow{v_2v_1}, \overrightarrow{v_2v_5})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_4 + \frac{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_1})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_5,$$

$$z_3 = \frac{det(\overrightarrow{v_3v_4}, \overrightarrow{v_4v_5})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_2 + \frac{det(\overrightarrow{v_2v_3}, \overrightarrow{v_2v_5})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_4 + \frac{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_3})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_5,$$

$$z_6 = \frac{det(\overrightarrow{v_6v_4}, \overrightarrow{v_4v_5})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_2 + \frac{det(\overrightarrow{v_2v_6}, \overrightarrow{v_2v_5})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_4 + \frac{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_6})}{det(\overrightarrow{v_2v_4}, \overrightarrow{v_2v_5})} z_5,$$

Along the edge, (z_5, z_7, z_8) will be the affine frame used to compute z_9, and similarly, (z_9, z_{10}, z_{11}) will give us the flip point z_{12}.

We can handle most irregular triangulations, but not arbitrary triangulations. We are limited by one case. We will discuss that in some detail in Section 4.4.

4.3 Local control

Locality is an important property for interactive design systems. The patches affected by the modification of a single control vertex, the less recomputation is needed, and therefore the faster the rendering speed. When it comes to locality, *complete locality* is highly desired, which means modification of a control vertex only affects patches that are incident to it.

Our algorithm generates propagations that are completely local. Local control is achieved because propagations always end at the flip points. Modification of any control point on any patch F will affect at most those control points associated with the patches that are adjacent to F. Only some of the control points of these incident patches are affected at all times, and furthermore, not all incident patches are affected under most circumstances.

Depending on the nature of the modified control point (corner, inner or flip), more or less repropagation is needed. Only control points that were prescribed in the original design are allowed to be modified.

4.3.1 Corner point region

Modification of a corner point or any of its star will result in repropagations of the entire corner point region and along all of its affected adjacent edges.

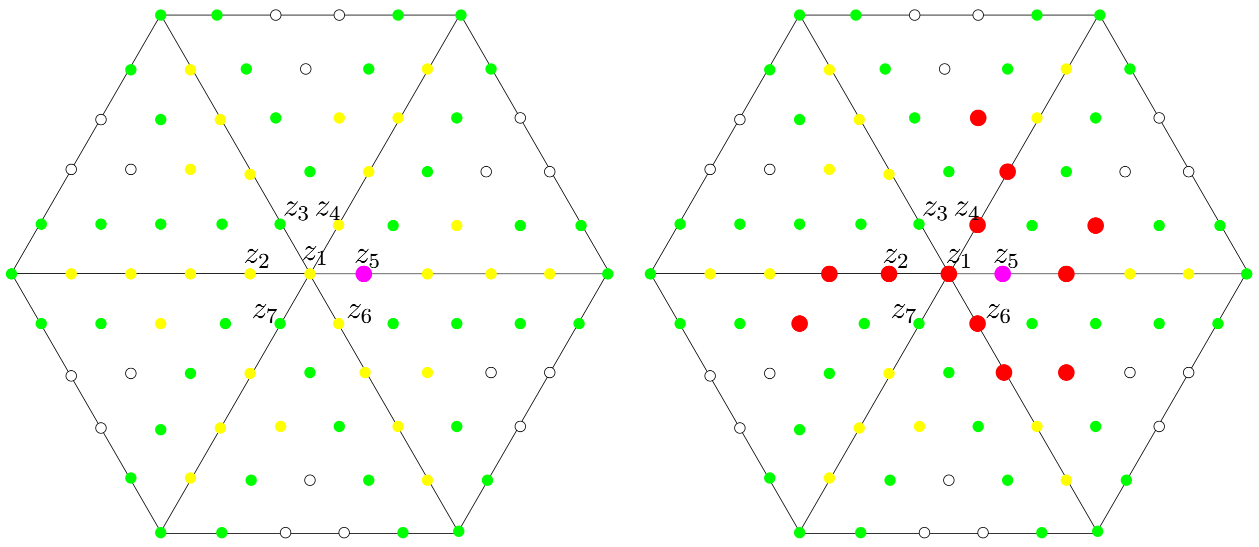


Figure 4.6: Local propagation in corner point region

In Figure 4.6, control point z_5 is modified. We adopt the convention of drawing modified control points in magenta. Since z_5 was one of the three points that formed the affine frame in that particular corner point region, once the affine frame changed, all the other control points in the same corner point region that do not belong to the affine frame, namely z_1, z_2, z_4 and z_6 need to be recomputed with respect to the new frame.

Notice that the example shows only one modified control point. All three control points that make up the frame are allowed to be modified. The changes made to z_2, z_4, z_6 and also the original z_5 will in turn affect the edges associated to them, e_2, e_4, e_6 and e_5 respectively. Edges e_3 and e_7 are not affected, as z_3 and z_7 are not modified.

Repropagation along any of the above edges is in fact only propagated through half of the edge. We know that there are two directions of propagation along any edge, but from any one corner point region, only one side of an edge is affected. Either direction is stopped by the flip point, and cannot proceed further.

Repropagation of any one corner region and its adjacent edges does not affect any other corner regions, as the flip points stops the propagation before it can reach any other corner region. This means that all corner point region propagations can be carried out in parallel, or when programmed sequentially, in any arbitrary order.

4.3.2 Inner points

Modification of an inner point (not a flip point) will result in repropagations along its two incident edges, as illustrated in Figure 4.7.

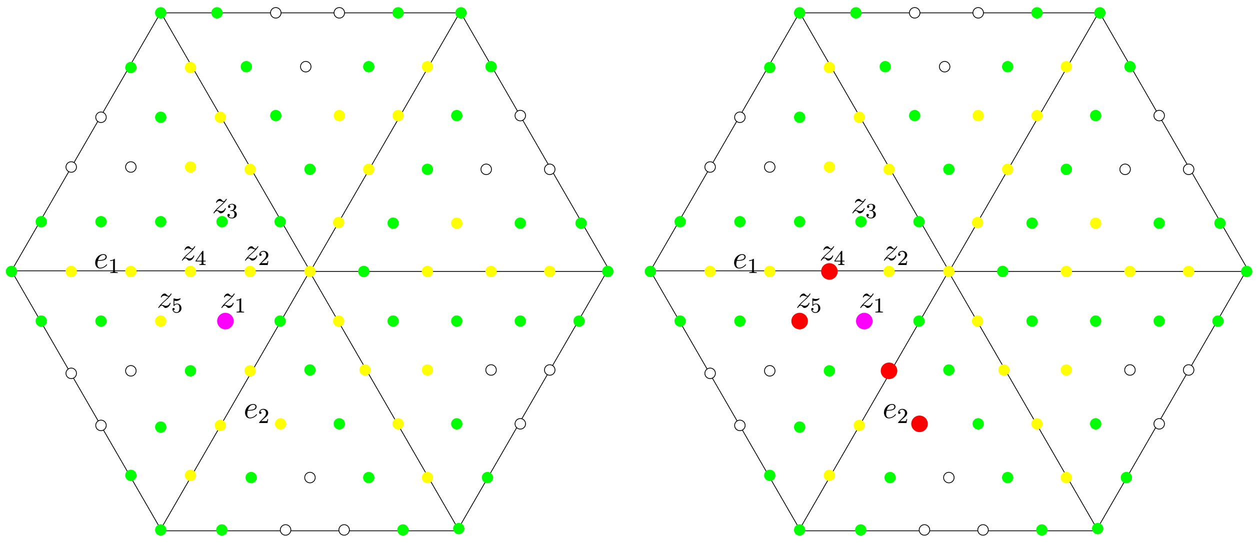


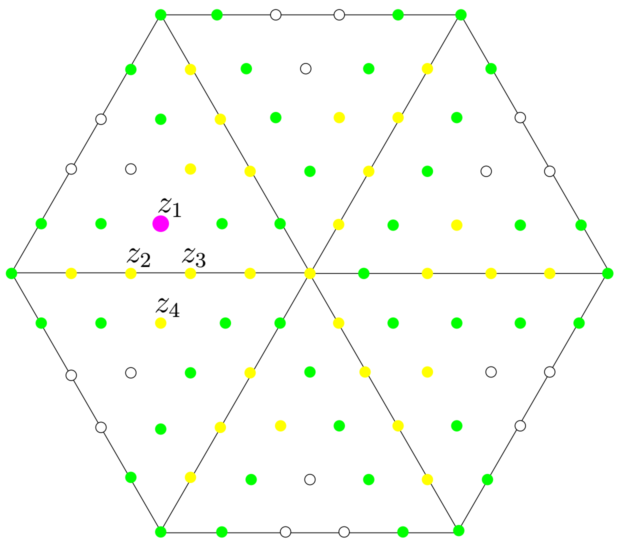
Figure 4.7: Local propagation along the edges

In Figure 4.7, inner point z_1 is modified. As a result, on edge e_1, the affine frame (z_1, z_2, z_3) has changed, and therefore z_4 needs to be recomputed. The changes to z_4 in turn affect the affine frame used to compute flip point z_5, therefore z_5 has to be recomputed as well. No other changes occurred on e_1. Similarly on edge e_2, one edge point and one flip point need to be recomputed.

Repropagation along an edge will not affect any other edges or corner point regions, again because the flip points "sinks" the propagations from both ends of the edge. All edge propagations can also be carried out in parallel, or in any arbitrary order.

4.3.3 Flip points

Modification of a flip point will result in repropagation of the other flip point across the edge, as illustrated in Figure 4.8.



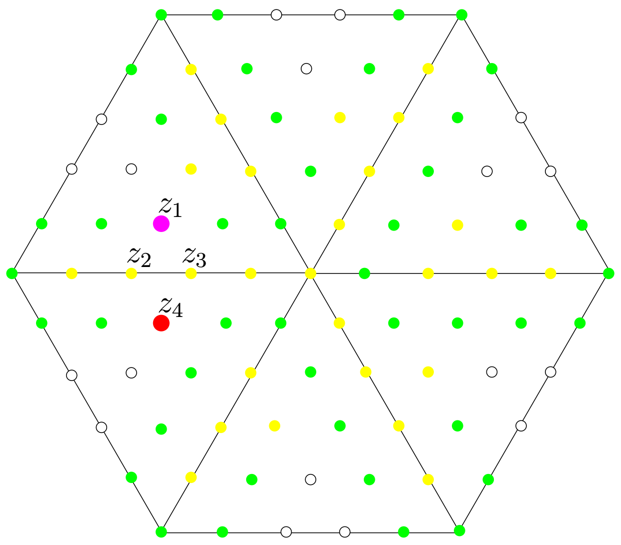


Figure 4.8: Local propagation of flip points

In Figure 4.8 flip point z_1 is modified. As a result, the affine frame (z_1, z_2, z_3) is changed, and therefore z_4 needs to be recomputed with respect to the new frame. No other control points are affected.

Repropagation of a flip point does not affect any corner point regions or any edges. Therefore, when a set of control points is modified, first repropagate all affected corner regions in any order, then along the affected edges in any order, finally recompute affected flip points in any order. This order (corner first, edge second, flip last) assures that all modifications take effect as they should.

4.4 Summary and future work

We believe that our algorithm provides a simple, fast way to design surfaces with built-in C^1 continuity. Since our method is based on triangular patches, there is no restriction on how many patches may come together at a control point, and thus it offers more freedom in terms of modeling capability. Furthermore, there are enough free (manipulatable) control points per patch (typically about $5-6$) such that very local changes might be made.

The triangular control net the algorithm generates lends very well to the subdivision version of the de Casteljau algorithm for evaluation. Given the minimum degree we require, 5, usually two iterations are enough to generate a good approximation of the surface. To manipulate and modify surface splines, it suffices to locally move control points. Recomputation of control points is limited to the local vicinity, and is therefore simple and easily implemented.

Complex topologies involving holes or sharp corners are automatically handled by the algorithm without extra work, because they do not arise as special cases. As long as the template triangulation reflects the existence of such topologies, they are handled properly. All the computations involved in the algorithm are just the propagations of the control points, which in most cases consist of just one addition and one subtraction.

Bellow are some directions we are taking to extend this work.

4.4.1 C^2

Preliminary experiment results have shown that our algorithm can likely be extended to construct C^2 triangular surfaces. The extension is in theory straight-forward, but working out an actual local incremental algorithm analogous to what we have shown so far is not trivial.

It turns out that at least degree 8 patches are needed everywhere for C^2, and in fact we need to move to degree 9 to have enough freedom locally to design a propagation scheme. Here, in addition to prescribing the first row cross-boundary control points, some second row cross-boundary control points will need to be prescribed as well. A balanced prescription scheme needs to be worked out to avoid shape defects.

4.4.2 Interpolation

The extension of our scheme to interpolate instead of approximate is almost instantaneous. Because we have three degrees of freedom around any corner-point region, including the corner point itself, we have enough freedom to simply force the corner point to pass through desired coordinates.

However, this does have the potential to lead to shape defects due to the problem mentioned in the last subsection, tilting of local affine frame, that is, if the choice of the corner-point's placement is unfortunate, it will result in undesired undulations in that corner point region. We will discuss this more in Subsection 4.4.5.

Also, recall that our C^1 algorithm requires the existence of a parameterization in the plane. If we assume that such a parameterization has been given, the extension to handle interpolation is indeed just as decribed above. We have also run some sucessful tests based on this assumption. However, in most cases, it is unrealistic to assume a given parameterization for interpolation purposes. The input will be a collection of data points given in space, although it is probably reasonable to assume that it is already meshed. We will need to come up with a reasonable parameterizaion of this mesh in the plane.

This is a non-trivial problem. A simple projection to a plane is not enough. Even with planes that are reasonably fitted to the data points, such as a least-squares fitting, it is still not good enough to generate a parameterizaion with no self-intersection. The problem arises when the mesh describe a surface that has folds (but no self-intersection). Any projection will not deal with the folds correctly. A reasonable parameterization in the plane will need to stretch out the folds in the mesh.

Gu, Gortler and Hoppe [GGH02] recently worked on remeshing irregular triangular meshes with meshes with (semi)-regular connectivity. In their work of representing surfaces with geometry images, they have worked on methods of cutting a mesh to open it into a topological disk, which then gives a parameterization of that mesh within this disk. This is quite possibly exactly what we can use to find a parameterization of our input mesh. Closer examination of Gu et al's methods is needed.

Of course, we are also aware of some restrictions of our method, some of which we are working on to resolve in the near future.

4.4.3 Shape parameters and automated shape control

Our algorithm is highly local and leaves quite a bit of freedom in the interior of the patches and the boundaries. The disadvantage of this is that, in general, a fair number of control points need to be modified before a change to a single patch will be uniformly distributed. This is definitely not a task a user should handle. An automatic method is needed to prescribe a set of appropriate control points, given a request for shape change. That is, some type of shape parameter needs to be set up.

A related issue has already been worked on (see Section 7.3), that is, to develop an automatic method to prescribe all the control points needed for propagation when given large sets of data to approximate. In this case, it is important to set good initial values for optimal first approximation so that less modification is needed from the user. It is still current research to set the initial free parameters to incorporate as much of the known discrete surface geometry as possible.

4.4.4 Degenerate frames along the edges

First, we mentioned in Section 4.2 that we cannot handle just any arbitrary triangulation, we are limited by one case. Recall that all propagations need the existence of at least three affinely independent points. Around a corner point region, this is guaranteed. But along the edges, because we have fixed directions of propagation, we do not have the freedom to choose which three points from the four control points that form the similar quadrilaterals will be our affine frame. They are fixed. In the unfortunate cases when these three points are collinear, the algorithm breaks down.

As shown in Figure 4.9, the points v_2, v_1, v_4 and v_3, v_1 v_5 are respectively collinear in the domain triangulation. The corner point region of z_1 and its star $z_2, \ldots, z_5$ is not affected, and in the example, z_3, z_4, z_5 are chosen as the affine frame, and z_1 and z_2 are then obtained through propagation. But then, the edge propagations immediately present a problem. For example, along e_1, the algorithm dictates that

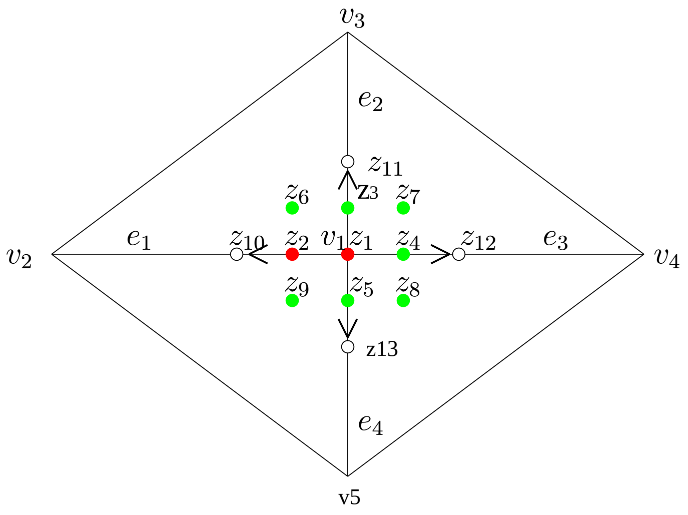


Figure 4.9: Frames along the edges are collinear

z_6, z_2, z_9 shall form an affine frame, resulting in z_{10}'s propagation. However, z_6, z_2, z_9 are given by the same affine relations as v_3, v_1, v_5, which are collinear. So are frames (z_6, z_3, z_7), (z_7, z_4, z_8), (z_8, z_5, z_9) along edges e_2, e_3 and e_4, respectively. Thus, control points z_{10}, z_{11}, z_{12} and z_{13} cannot be computed. In this case, we have two straight lines going through v_1/z_1. In general, any straight line that goes through a corner point will cause similar problem for two edge propagations adjacent to that particular corner point.

There are two possible fixes for this problem. We can detect it and then change the direction of propagations locally where such degenerate frames exist. Notice that because our domain is a triangulation, there is always at least one direction along an edge that does not contain degenerate frames.

Another approach is to leave the algorithm alone, but preprocess to remove the degenerate frames. We simply need to move v_1 slightly off to the side, to disrupt the straight lines. The move is therefore a small one, and will not cause noticeable difference in the final surface generated. Any irregular triangulation will need to be preprocessed to remove such degenerate cases.

4.4.5 Excessive tilting of the affine frames

This is the most serious problem that we have encountered so far. Recall that both around the corner point region and across the edges, propagation places control points on a plane determined by the affine frame formed by the three affinely independent control points that we prescribed. This means that the placement of this plane relative to the normals of the surface in that region is extremely important to the final shape of the surface.

In general, we want this plane to be tangent to the surface as much as possible. That is why in the G^1 literature it is often called the "cross-boundary tangent plane". If by some unfortunate choice (most likely due to interpolation), the affine frame is tilted towards the normal of the surface, then the resulting surface will contain severe undulations.

Fortunately, even in the case of interpolation, we still have two other control points which we can prescribe to mitigate the tilt, if it occurs. The best method of fairing is currently being researched.

Chapter 5

Geometric Continuity

5.1 Background and related work

The algorithm given in Chapter 4 works only on planar triangulations, which means that it cannot produce closed surfaces. Closed surfaces will need triangulated polyhedra as parameter space. We recognize that our problem is fundamentally that of going from one topological space to another. Algebraic topology has already developed tools to study such problems.

We know that surfaces are topologically 2-manifolds (see definition 5.4.1), which are topological spaces in which each point has a neighborhood (an open set that contains the point) homeomorphic to a portion of the Euclidean plane. Naturally, these neighborhoods form an *open cover* of the manifold. An open cover of a topological space $\mathcal{X}$ is a family of open subsets whose union is $\mathcal{X}$. One can think of a manifold as a structure imposed on a set, dividing the set into overlapping regions that covers the set completely, with each of the region behaving much like a portion of the Euclidean plane. Now since the regions overlap, there are families of homeomorphisms called *charts*, that describe how the regions overlap.

The difficulty of mapping a triangulated polyhedron to a surface is that we cannot use open sets, but only closed sets, e.g. triangles. Therefore instead of loosely "overlapping", we must conduct precise boundary matching. (Incidentally, there have been attempts to represent a surface with a topological 2-manifold thereby allowing the surface patches to overlap [GH95].)

A common method to study closed triangulatable compact surfaces in algebraic topology is to cut a surface open along the edges of the triangulation and "flatten" it out, i.e. to realize the surface as a polygon in the plane, with certain identifications on the boundary edges. The following diagram shows a triangulation of the *sphere* which has been opened up:

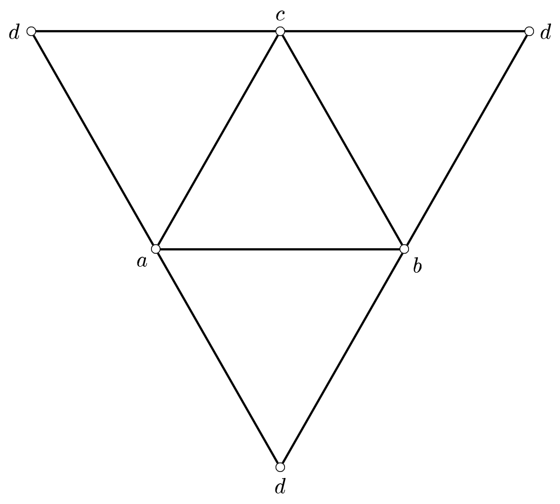


Figure 5.1: A triangulation of the sphere

The geometric realization of the above triangulation is obtained by pasting together the pairs of edges labeled (a, d), (b, d), (c, d). The geometric realization forms a tetrahedron. We recognize this as exactly the triangulated polyhedron we had in mind as a parameter space. Algebraic topology states that if we view the triangulated polyhedron as a geometric realization of the actual surface we are fitting on top of it, they will be topologically equivalent.

However, it is well known that fitting a parametrically continuous surface over triangulated polyhedra of arbitrary topology can not always be done. As Herron showed in [Her85], requiring C^1 continuity along the patch boundaries is only possible if the corresponding domain triangles are adjacent. Instead, a common solution to the problem is to relax the continuity constraints to what is know as *geometrically continuous*, and fit G^k-continuous surfaces instead.

Geometric continuity is a different class of continuity as its definition is parameterization independent. Geometric continuity only concerns itself with the *shape* of a curve or surface. Various early definitions of geometric continuity were given in Farin [Far82], Sabin [Sab83], DeRose [DeR85], Herron [Her87], etc. There are also more recent surveys on geometric continuity such as Veltkamp [Vel92] and Peters [Pet02]. Despite different interpretations, geometric continuity is basically defined as follows:

Definition 5.1.1 *Two surface patches F and G are said to be G^k-continuous at the joining point a if and only if there exist two reparameterizations f and g, such that $F \circ f$ and $G \circ g$ are C^k-continuous at a.*

It was also pointed out in the literature that although the definition suggests that both F and G require reparameterizations, it is equivalent if only one of them is reparameterized, reducing the other reparameterization function to the identity.

The majority of the literature does not clearly specify what these reparameterization functions f and g are. With the exception of Peters [Pet02], most of them describe them as *change of variables* or *change of parameters* functions.

In this dissertation a reparameterization map is a C^k *diffeomorphism* $f\colon \mathbb{A}^2 \to \mathbb{A}^2$, that is, f is a bijection such that both f and its inverse are C^k. Recall that a *diffeomorphism* f between manifolds (subsets of $\mathbb{R}^n$) is an invertible map, such that f and its inverse are infinitely differentiable.

Now, finding such reparametrizations is of course no easy task. It turns out that for G^1 and G^2, they are easily worked out: two adjacent patches F and G are G^1-continuous at a if normal vectors at $f(a)$ and $g(a)$ are in the same direction. Therefore G^1-continuous is also called tangent plane continuous. Similarly, G^2 requires that two patches are tangent plane continuous and curvature continuous.

Many methods of building triangular G^1 surfaces have been proposed. These techniques differ mainly in the way they simultaneously satisfy smoothness conditions for many patches around a common corner. This is also known as the "vertex consistency" or "twist compatibility" problem. The difficulty arises when joining an even number of C^2 patches sharing a common vertex with G^1 continuity, as the system of linear equations involved may be singular.

There are three major approaches to deal with the vertex consistency problem.

1. *Subdivision algorithms.*

These schemes average arbitrary input meshes to isolate irregularities. Examples on triangular meshes (i.e. subdivision of abstract complexes) are Loop [Loo87], Cavaretta et al [CDM91], Hoppe et al [HDD$^+$94] and Peters [Pet95], and more recently, [BLZ00] and [Kob01]. For a survey of continuity conditions of subdivision surfaces, please refer to Zorin [Zor96].

2. *Split-patch methods.*

   These are Clough-Tocher like split-domain schemes. Earlier approaches including Farin [Far82, Far83], Jensen [Jen87], Piper [Pip87] and Shirman and Sequin [SS88]. A more recent approach due to Hahmann et al [HBT00, HBT01] applies a new 4-split method. Split-domain schemes generally produce lower degree surfaces.

3. *Convex combination schemes.*

   This type of methods such as Hagen [Hag86] and Hagen and Pottmann [HH89] construct higher degree triangular patches with a single patch per parameter triangle. They are named convex combination because n patches are constructed for n-sided patches, and then a blended convex combination of the patches make the final patch.

In addition to the above approaches, there are C^2-consistent boundary curve schemes such as Loop [Loo94] and Peters [Pet91], variational method using the theory of manifolds by Sarraga [Sar00] and B-patches by Seidel [GS93, PS95]. The surveys by Du and Schmitt [DS90] and Mann et al [MLM$^+$92] provide more detailed description of the above methods. It is pointed out in [MLM$^+$92] that most of these methods suffer from shape defects.

In the next sections we present the foundation of an extension of our incremental algorithm to handle G^1 and possibly G^2 surfaces based on triangulated polyhedra of arbitrary topology. We differ from the standard approaches in that we do not require presetting information (normals or partial derivatives) at vertex regions for interpolation purposes first, and thus end up with extra degrees of freedom. In fact this freedom around the vertices should allow easier future extensions to handle interpolation.

We also completely bypass the vertex consistency problem since we do not fit boundary curves first [1]. Constructing boundary curves is a common aspect of most of the G^1 schemes proposed so far. Usually, boundary curves are first computed to fit the cross-boundary tangent field. Once the boundary curves are fixed, the interiors of the patches are then filled with Bézier points computed from these boundary curves. This leads to the well-known vertex consistency problem. In our approach, vertex regions are fitted FIRST, and boundaries are considered thereafter. Therefore the vertex consistency problem does not exist.

We believe that our approach has a number of other advantages:

- *Parametric patches.* Our approach fits Bézier patches, whose parametric nature enables well known algorithms like subdivision and intersection.
- *Single patch per domain triangle.* We construct one patch only at all times, without intermediate splitting or degree raising.

[1]We thank Stephen Mann for pointing out this fact.

- *Acceptably low degree.* We use quintic Bézier patches, one degree lower than for example Loop [Loo94]. While there exist schemes that produce lower degree patches, most of those schemes do not offer complete local control.

- *Local Control.* This is one of the most important advantages of this approach. The complete local control present in our C^1 algorithm generalizes to the more general method.

- *Interpolation.* While approximation was the first goal, there is plenty of freedom left in the vertex region for an extension to interpolation.

- *Extension to G^2.* Again, as discussed in our C^1 algorithm, G^2 extension is most likely achieved with degree 8 patches, degree 9 if local control is desired.

5.2 A definition of a triangulated polyhedron

Since our domain is a triangulated polyhedron, we first give a formal definition of what we mean by a triangulated polyhedron. Arbitrary spaces are not generally triangulatable. However, it can be shown that any compact space can be expressed as a limit of triangulatable spaces in a reasonable sense, thus, triangulatable spaces are dense in the family of compact spaces. Furthermore, any compact surface is triangulatable. According to Massey [Mas87] the proof was first given by Radó in 1925.

Note that most of the definitions and concepts in this section are taken out of [Xu99], inspired by [Gal97]. We will give formal definitions of simplices and complexes, the geometric realization of a simplicial complex and general triangulation. This is done in standard affine geometry. Readers may also assume standard algebraic topology definitions for open and closed sets, compactness and connectedness. For further readings on this topic, please refer to Bredon [Bre93], Fulton [Ful95] and Massey [Mas91].

5.2.1 Simplices and Complexes

Definition 5.2.1 Let $\mathcal{E}$ be any affine space with the standard affine basis $e_0, \ldots, e_n$ in $\mathcal{E}$. An *n-simplex (or simplex)* Δ^n is defined by:

$$\Delta^n = \left\{ \sum_{i=0}^{n} \lambda_i e_i \mid \sum_{i=0}^{n} \lambda_i = 1,\ 0 \le \lambda_i \le 1 \right\}.$$

The families of coefficients λ_i are called *barycentric coordinates*. The *dimension* of an n-simplex Δ^n is n.

The n-simplex is quite simply just the convex hull of the points $e_0, \ldots, e_n$, that is, the set of all convex combinations $\lambda_0 e_0 + \cdots + \lambda_n e_n$, where $\lambda_0 + \cdots + \lambda_n = 1$, and $\lambda_i \ge 0$, for all i, with $0 \le i \le n$.

Definition 5.2.2 Given an n-simplex Δ^n, the points $e_0, \ldots, e_n$ are called the *vertices* of Δ^n. A k-simplex generated by any k-subset of the vertices $\{e_0, \ldots, e_n\}$ is called a *k-face* or simply *face* of Δ^n. A face T of Δ^n is a *proper face* iff $T \neq \Delta^n$ ($\emptyset$ is a face of any simplex). Every face which is an $(n-1)$-simplex is called a *boundary face*.

The union of the boundary faces is the *boundary of* Δ^n, denoted as $\partial\Delta^n$, and the complement of $\partial\Delta^n$ in Δ^n is the *interior*, $\overset{\circ}{\Delta}{}^n$, of Δ^n.

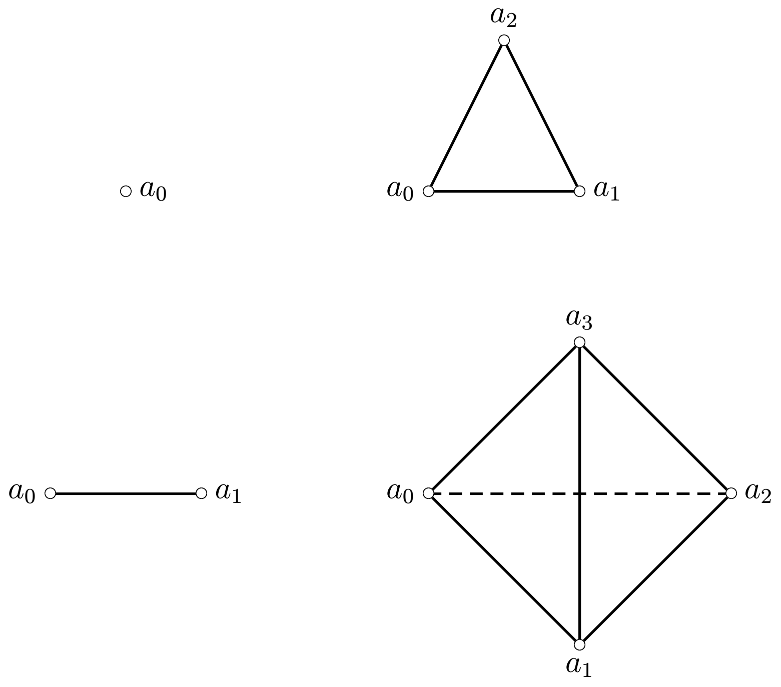


Figure 5.2: Examples of simplices

Clearly, a point x belongs to the boundary of Δ^n iff at least one of its barycentric coordinates $(\lambda_0, \ldots, \lambda_n)$ is zero, and a point x belongs to the interior of Δ^n iff all of its barycentric coordinates are positive.

A simplex Δ is convex, connected, compact, and closed. The interior of a complex is convex, connected, open, and Δ is the closure of $\overset{\circ}{\Delta}{}^n$. A simplex Δ^n is the smallest convex set in $\mathbb{R}^{n+1}$ containing all its vertices, thus, it is said to *span* its vertices.

We now need to put simplices together to form more complex shapes. Ahlfors and Sario [AS60], define abstract complexes and their geometric realizations rather formally. The idea is to construct a combinatorial object called an *abstract simplicial complex* K by taking a (finite) set of simplices as vertices, and a collection of subsets from this set, with the property that every subset of a simplex is a simplex. However, to understand its *geometric realization*, one would need to interpret the simplicial complex in terms of a vector space. An alternative definition called a *geometric simplicial complex* was given by Bredon [Bre93], which seems a bit more intuitive, and we will present it here:

Definition 5.2.3 A *(geometric) simplicial complex* in $\mathbb{A}^n$ (for short *complex*), K, is a finite family of simplices in $\mathbb{A}^n$ satisfying the following conditions:

(A1) Given a simplex $\Delta \in K$, any face of Δ is also in K.

(A2) Given two simplices, Δ_1 and Δ_2, either Δ_1 and Δ_2 are disjoint, or $\Delta_1 \cap \Delta_2$ is a face.

The highest dimension of the simplices in K is the *dimension* of K. A 0-simplex in K is a *vertex* of K.

Example: Sato [Sat99] shows a simplex Δ (in fact a tetrahedron) whose boundary is a simplicial complex. Given four points in $\mathbb{R}^3$ as follows:

$$p_0 = (0,0,0),\ p_1 = (1,0,0),\ p_2 = (1,2,0),\ p_3 = (2,3,4),$$

from these four points, there are

$\binom{4}{3} = 4$ 2-simplices:

$$\Delta_0^2 = (p_0, p_1, p_2),\ \Delta_1^2 = (p_1, p_2, p_3),\ \Delta_2^2 = (p_0, p_2, p_3),\ \Delta_3^2 = (p_0, p_1, p_3),$$

$\binom{4}{2} = 6$ 1-simplices:

$\Delta_0^1 = (p_0, p_1),\ \Delta_1^1 = (p_0, p_2),\ \Delta_2^1 = (p_0, p_3),\ \Delta_3^1 = (p_1, p_2),\ \Delta_4^1 = (p_1, p_3),\ \Delta_5^1 = (p_2, p_3)$

and $\binom{4}{1} = 4$ 0-simplices:

$$\Delta_0^0 = (p_0),\ \Delta_1^0 = (p_1),\ \Delta_2^0 = (p_2),\ \Delta_3^0 = (p_3).$$

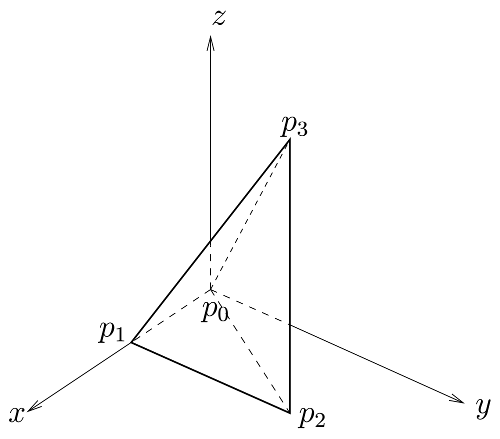


Figure 5.3: Example of a simplicial complex

It is clear that the 2-simplices are the faces, the 1-simplices the edges, and the 0-simplices the vertices, and the union of all 14 simplices is a set K which satisfies the conditions given in Definition 5.2.3, and therefore is a simplicial complex. It is also the boundary of the 3-simplex Δ.

Definition 5.2.4 Given a simplicial complex, K, its *geometric realization* (also called the *polyhedron of* K) is the subspace $|K|$, defined as $|K| = \bigcup_{\Delta \in K} \Delta$.

We define a topology on $|K|$ by defining a subset U of $|K|$ to be closed iff $U \cap |\Delta|$ is closed in $|\Delta|$ for all $\Delta \in K$. It is immediately verified that the axioms of a topological space are indeed verified. The space $|K|$ has the subspace topology as a subset of $\mathbb{R}^n$. We also note that for any two simplices Δ_1, Δ_2 of K, we have

$$|(\Delta_1 \cap \Delta_2)| = |(\Delta_1)| \cap |(\Delta_2).$$

5.2.2 Triangulation

We now return to surfaces and define the notion of triangulation. The name "triangulation" is a bit misleading in the way that it implies "triangles", which we automatically assume are 2-dimensional. We have already seen that a triangle is a 2-simplex, therefore triangulation of higher-dimension spaces cuts the spaces up into higher dimension triangles, i.e., n-simplices. Triangulation can be thought as a method of cutting a topological space into a finite number of warped simplices, which are homeomorphic to n-simplices, and since the space is a finite collection of such simplices, the space is thus identified to a simplicial complex using homeomorphisms. Applying this to compact surfaces means that we divide the surface into a finite number of faces, each of which is homeomorphic to a planar triangle.

Definition 5.2.5 Given a topological space X, we say X is *triangulatable*, if there is a homeomorphism identifying X with a simplicial complex K. Such a homeomorphism $g\colon |K| \to X$, is called a *triangulation* of X.

Definition 5.2.6 Given a surface M, a *triangulation* of M is a pair (K, h), where K is a 2-simplex and h is map $h\colon K \to 2^M$ from the 2-dimensional simplicial complex K into the set of subsets of the surface M, assigning a closed subset $h(s)$ of M to every simplex $s \in K$, and satisfying the following conditions:

(C1) $h(s_1 \cap s_2) = h(s_1) \cap h(s_2)$, $\forall s_1, s_2 \in K$.

(C2) For every $s \in K$, there is a homeomorphism φ_s from the geometric realization $|s|$ of s to $h(s)$, such that $\varphi_s(|s'|) = h(s')$, for every $s' \subseteq s$.

(C3) $\bigcup_{s \in K} h(s) = M$, that is, the sets $h(s)$ cover M.

(C4) For every point $x \in M$, there is some neighborhood of x which meets only finitely many of the $h(s)$.

Condition C2 basically guarantees that every 2-simplex (triangle) in K is homeomorphic to a closed subset in M, and also every face of every 2-simplex, that is, every edge, is homeomorphic to a closed interval, and every vertex is homeomorphic to a point.

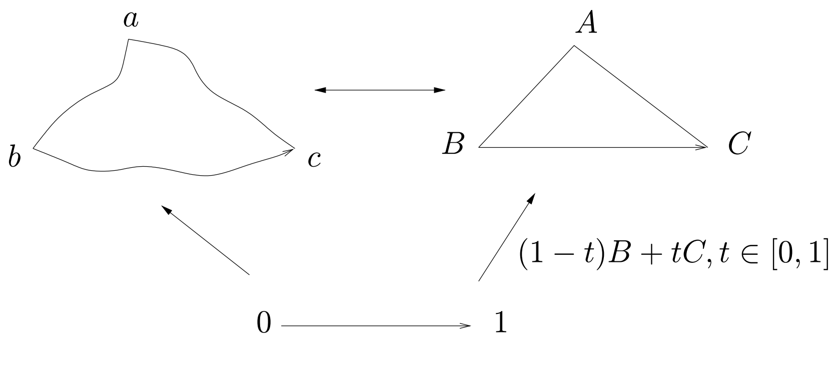


Figure 5.4: Mapping an edge onto an interval

We have already seen an example of the triangulation of a sphere in Figure 5.1. The following diagram shows a triangulation of a surface called a *torus*:

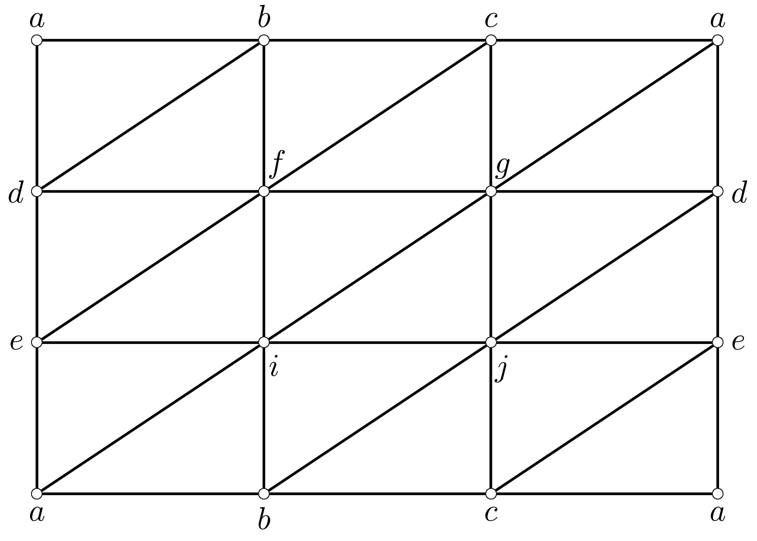


Figure 5.5: A triangulation of the torus

The geometric realization of the above triangulation is obtained by pasting together the pairs of edges labeled (a, d), (d, e), (e, a), and the pairs of edges labeled (a, b), (b, c), (c, a).

We are now going to state a lemma characterizing the complexes K that correspond to triangulations of surfaces. It gives a more combinatorial notion of a 2-complex corresponding to a triangulation. The following notational conventions will be used: vertices (or nodes, i.e., 0-simplices) will be denoted by α, edges (1-simplices) will be denoted by a, and triangles (2-simplices) will be denoted by A. We will also denote an edge by $a = (\alpha_1\alpha_2)$, and a triangle by $A = (a_1a_2a_3)$, or by $A = (\alpha_1\alpha_2\alpha_3)$, when we are interested in its vertices.

Lemma 5.2.7 *A 2-complex K corresponds to a triangulation $h: K \to 2^M$ of a surface M iff the following properties hold:*

(D1) Every edge a is contained in exactly two triangles A.

(D2) For every vertex α, the edges a and triangles A containing α can be arranged as a cyclic sequence $a_1, A_1, a_2, A_2, \ldots, A_{m-1}, a_m, A_m$, in the sense that $a_i = A_{i-1} \cap A_i$ for all i, $2 \leq i \leq m$, and $a_1 = A_m \cap A_1$, with $m \geq 3$.

(D3) K is connected, in the sense that it cannot be written as the union of two disjoint nonempty complexes.

Proof. A proof can be found in Ahlfors and Sario [AS60]. The proof is non-trivial and requires the notion of the winding number of a closed curve in the plane with respect to a point, and the concept of homotopy. □

A 2-complex K that satisfies the conditions of Lemma 5.2.7 will be called a *triangulated complex*, and its geometric realization is called a *triangulated polyhedron*.

5.3 AG^k continuity on triangulated polyhedra

We present a mathematically rigorous definition for a triangular spline surface based on a triangulated polyhedron. Although there are various efforts to fit G^k surfaces in the literature we surveyed, there have not been any attempt to formulate the problem formally. From our definition of a (triangulated) polyhedron-based spline surface, we also introduce a new kind of geometric continuity for such surfaces.

A triangulation of a compact surface $\mathcal{X}$ has a finite number of faces, each of which is a subset of $\mathcal{X}$ homeomorphic to an ordinary closed triangle in the plane. There are many homeomorphisms between a face of a triangulated compact surface and a planar triangle. To simplify matters, we can think of the mapping between a triangulated compact surface and planar triangles as a two-step process. First, the triangulated compact surface is mapped to a triangulated polyhedron, through a polynomial map. Then, the triangulated polyhedron is in turn mapped to ordinary (closed) planar triangles (with identifications). When we identify sides of triangles with edges of a triangulated polyhedron, the homeomorphisms must be affine. Figure 5.6 shows this process, where the triangulated polyhedron is cut along the red edges.

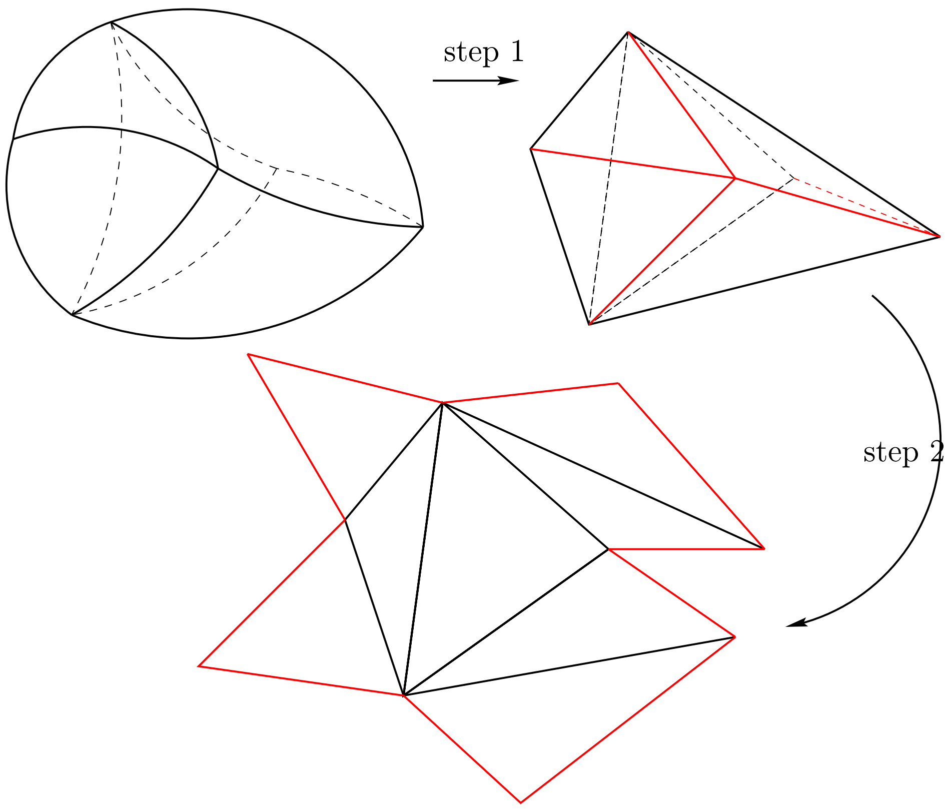


Figure 5.6: From surface to triangulated polyhedron, then to planar triangles

Thus we call our version of geometric continuity *affine geometric continuity* , or AG^k-continuity.

Definition 5.3.1 A *piecewise polynomial surface $\mathcal{S}$ of degree m based on a triangulated polyhedron* is a triple $(P, \{\phi_T\}_{T\in\mathcal{T}}, \{F_T\}_{T\in\mathcal{T}})$ where,

1. $P = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ is a triangulated polyhedron, with $\mathcal{V}$ denoting the subset of 0-simplices, i.e., vertices, $\mathcal{E}$ denoting the subset of 1-simplices, i.e., edges and $\mathcal{T}$ denoting the subset of 2-simplices, i.e., face triangles.

2. Each $\phi_T : T \to \mathbb{A}^2$ is an affine bijection mapping the face triangle $T \in \mathcal{T}$ to the affine plane.

3. Letting $T' = \phi_T(T)$, each $F_T : T' \to \mathbb{A}^3$ is a polynomial surface patch of degree m.

4. For every edge $e \in \mathcal{E}$ adjacent to two faces T_i and T_j, we require the restrictions of F_i and F_j to this edge to agree. That is,

$$F_{T_i} \circ \phi_{T_i} | e = F_{T_j} \circ \phi_{T_j} | e.$$

In short, we call such a surface a *polyhedron-based surface.* See Figure 5.7.

Remark: The family of bijective affine maps $\{\phi_T\}_{T\in\mathcal{T}}$ is indexed by the face triangles in $\mathcal{T}$. When dealing with distinct faces T_i and T_j, for simplification of notation, we allow ourselves to abbreviate expressions such as ϕ_{T_i} and ϕ_{T_j} as simply ϕ_i and ϕ_j respectively. From now on, T_i will be consistently replaced by i when used

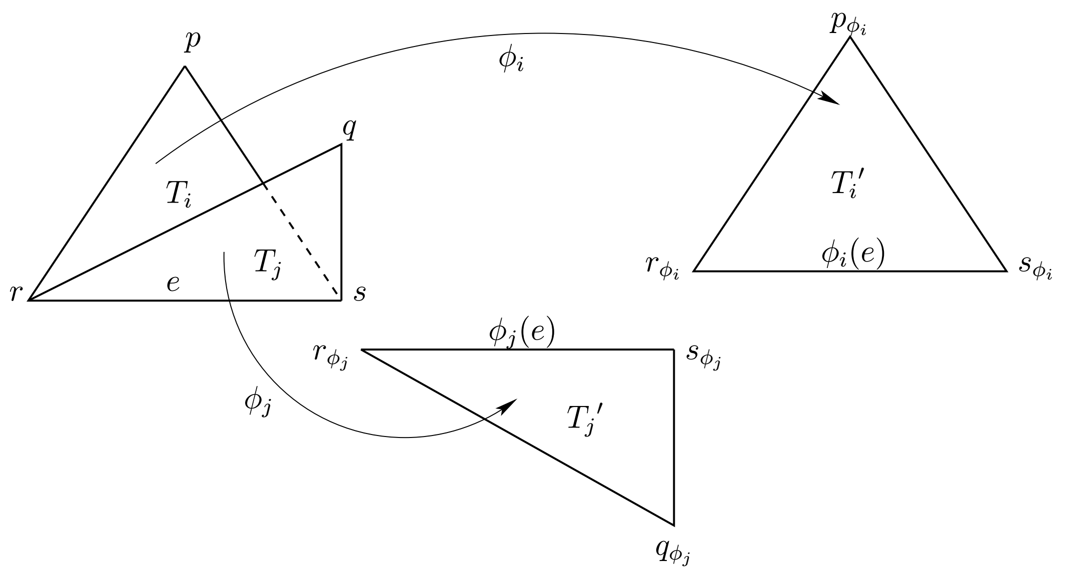


Figure 5.7: The maps ϕ_T send faces on a polyhedron to triangles in $\mathbb{A}^2$

to index maps. The maps $\{\phi_T\}_{T\in\mathcal{T}}$ send the faces of the polyhedron P to the affine plane $\mathbb{A}^2$. We also index points with maps when convenient, that is, we denote $\phi_i(p)$ as p_{ϕ_i}, as shown in Figure 5.7.

Notice that two faces adjacent on the polyhedron are not necessarily mapped to adjacent triangles in $\mathbb{A}^2$. Since P is closed, it is not possible to send all adjacent polyhedral faces to adjacent faces in the plane without identifications. Because we require C^0 continuity in condition 4 in the previous definition, we need reparameterization maps to define AG^k-continuity.

Definition 5.3.2 Given a *polyhedron-based surface* $\mathcal{S} = (P, \{\phi_T\}_{T\in\mathcal{T}}, \{F_T\}_{T\in\mathcal{T}})$ of degree m, for every edge $e \in \mathcal{E}$ adjacent to two faces T_i and $T_j \in \mathcal{T}$, let $p, r, s \in \mathcal{V}$ denote the vertices of T_i and $q, r, s \in \mathcal{V}$ denote the vertices of T_j, with $e = (r, s)$. An *affine reparametrization map* is a bijective affine map $\rho_j\colon \mathbb{A}^2 \to \mathbb{A}^2$, mapping $T_j{}'$ to

$\mathbb{A}^2$, $(T_j{}' = \phi_j(T_j))$, such that

1. $\phi_i(e) = \rho_j(\phi_j(e))$.
2. The line segment $(\phi_i(p), \rho_j(\phi_j(q))$ (or $(p_{\phi_i}, \rho_j(q_{\phi_j})))$ has an intersection with the line segment $\phi_i(e)$ or its extension. In other words, p_{ϕ_i} and $\rho_j(q_{\phi_j})$ lie on opposite sides of $\phi_i(e)$.

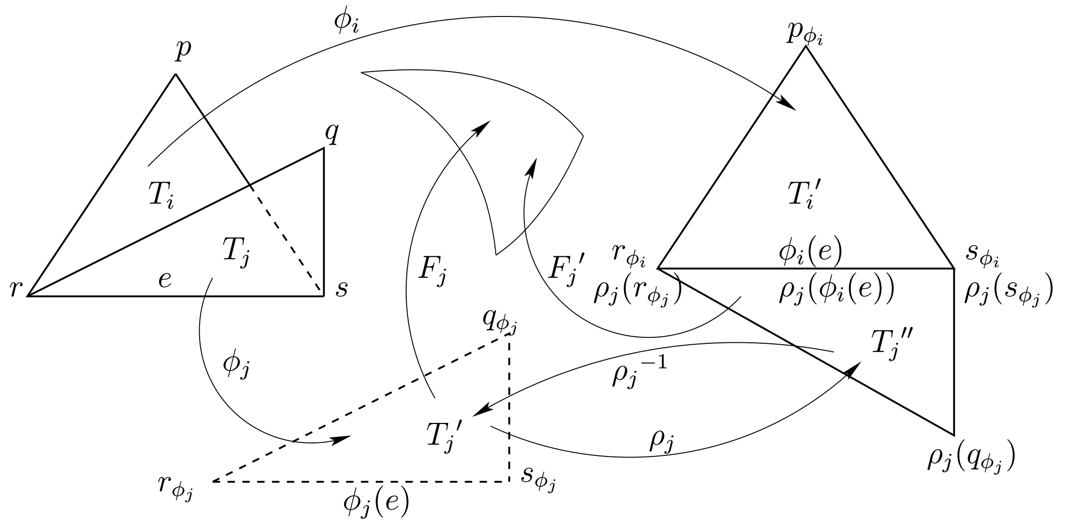


Figure 5.8: The map ρ_j reparameterizes so that the two triangles $T_i{}'$ and $\rho_j(T_j{}')$ are adjacent in $\mathbb{A}^2$, where $T_j{}' = \phi_j(T_j)$.

Remark: Let $T_j{}''$ denote $\rho_j(T_j{}')$. Notice that the reparameterization maps send potentially non-adjacent triangles (triangles that are scattered into the affine plane via maps ϕ_T) in the plane to adjacent triangles.

Now we show that the control points attached to any reparameterized triangular face $T_j{}''$ are in fact the same control points attached to the original face $T_j{}'$.

Given the polynomial map $F_j : T_j{}' \to \mathbb{A}^3$, let $F_j{}' : T_j{}'' \to \mathbb{A}^3$ be the polynomial map so that $F_j{}' = F_j \circ \rho_j{}^{-1}$. Let $f_j : T_j{}' \to \mathbb{A}^3$ and $f_j{}' : T_j{}'' \to \mathbb{A}^3$ be the corresponding polar forms of F_j and $F_j{}'$, respectively. Also, let $b_{\alpha\beta\gamma}$ and $b'_{\alpha\beta\gamma}$ be the control points of F_j and $F_j{}'$, with respect to $T_j{}'$ and $T_j{}''$, respectively.

Lemma 5.3.3 *With the notations above, we have*

$$b_{\alpha\beta\gamma} = b'{}_{\alpha\beta\gamma}$$

, i.e. the control points attached to any reparameterized triangular face $T_j{}'$ *are not affected by the reparameterization map* $\rho_j : T_j{}' \to T_j{}''$.

Proof: Now the control point $b_{\alpha\beta\gamma}$ of $f_j{}'$ with respect to $T_j{}''$ must be

$$f_j{}'(\underbrace{\rho_j(r_{\phi_j}), \ldots, \rho_j(r_{\phi_j})}_{\alpha}, \underbrace{\rho_j(s_{\phi_j}), \ldots, \rho_j(s_{\phi_j})}_{\beta}, \underbrace{\rho_j(q_{\phi_j}), \ldots, \rho_j(q_{\phi_j})}_{\gamma}),$$

that is,

$$f_j{}'((\rho_j(r_{\phi_j}))^{\alpha}(\rho_j(s_{\phi_j}))^{\beta}(\rho_j(q_{\phi_j}))^{\gamma}). \qquad (5.1)$$

We also know that $f_j{}' = f_j \circ \rho_j{}^{-1}$. That is,

$$f_j{}'(a_1, \ldots, a_m) = f_j(\rho_j{}^{-1}(a_1), \ldots, \rho_j{}^{-1}(a_m)).$$

Substituting in (5.1) for the arguments $a_1, \ldots, a_m$, we have

$$f_j{}'((\rho_j(r_{\phi_j}))^{\alpha}(\rho_j(s_{\phi_j}))^{\beta}(\rho_j(q_{\phi_j}))^{\gamma}) = f_j((\rho_j\rho_j{}^{-1}(r_{\phi_j}))^{\alpha}(\rho_j\rho_j{}^{-1}(s_{\phi_j}))^{\beta}(\rho_j\rho_j{}^{-1}(q_{\phi_j}))^{\gamma}).$$

The term $\rho_j {\rho_j}^{-1}$ cancels and the above equation becomes:

$$b'_{\alpha\beta\gamma} = {f_j}'((\rho_j(r_{\phi_j}))^\alpha(\rho_j(s_{\phi_j}))^\beta(\rho_j(q_{\phi_j}))^\gamma) = f_j((r_{\phi_j})^\alpha(s_{\phi_j})^b eta(q_{\phi_j})^\gamma) = b_{\alpha\beta\gamma},$$

where $b'_{\alpha\beta\gamma}$ is the control point with respect to ${T_j}'$.

This proves that control points with respect to ${T_j}''$ are exactly the same as those with respect to ${T_j}'$. $\square$

Definition 5.3.4 Using Definition 5.3.1, if we let ${T_j}'' = \rho_j({T_j}')$, we say that $\mathcal{S}$ is affine G^k-continuous or AG^k if for every edge e adjacent to two faces T_i and T_j, the patches F_i and ${F_j}' = F_j \circ {\rho_j}^{-1}$ meet with C^k continuity along the common edge curve $\phi_i(e)$.

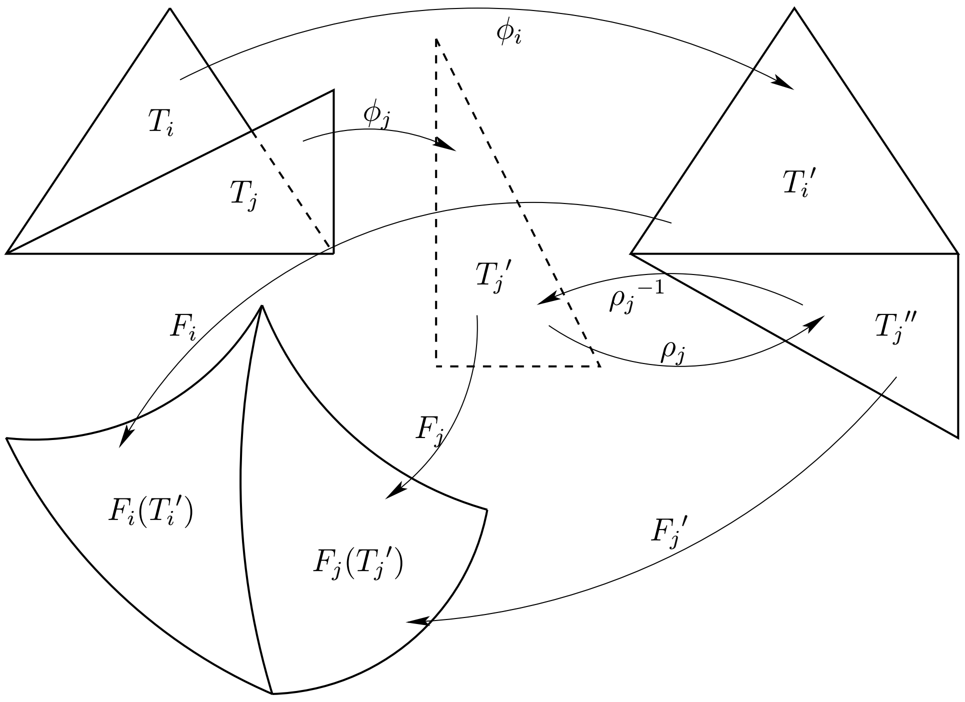


Figure 5.9: $F_i({T_i}')$ and ${F_j}'({T_j}'')$ are AG^k-continuous.

5.4 Approaches to fit AG^1 surfaces

Now we proceed to the construction of an actual algorithm that will fit such AG^1 surfaces over a triangulated polyhedron.

We have looked at the problem from two angles. The first is an obvious, but somewhat more global and brute-force approach. We propose to open up the triangulated polyhedron along selected edges and lay it flat on the plane with proper identifications among pairs of edges that were cut. This is what we have seen in Section 5.2 as a 2-complex corresponding to a triangulation. What we have done is to go from its geometric realization back to the complex. Since now all domain triangles are planar, we can proceed to write out the necessary conditions on polar forms for C^1 continuity as shown in Section 3.2. Solving this system of equations will give us an idea of at least the existence of the reparameterization maps ρ_i.

The next idea is to leave the triangulated tetrahedron in three dimensional space, and find local parameterizations around each corner point region and across each edge boundary so that the immediate neighborhood is mapped to the affine plane $\mathbb{A}^2$. We know that such local parameterization exists because the standard algebraic topology definition of a 2-manifold states:

Definition 5.4.1 A *(topological)* 2*-manifold* is a second-countable Hausdorff space M, together with an open cover $(U_i)_{i\in I}$ and a family $(\varphi_i)_{i\in I}$ of homeomorphisms $\varphi_i\colon U_i \to \Omega_i$, such that each Ω_i is some open subset of $\mathbb{R}^2$. Each pair (U_i, φ_i) is called a *coordinate system*, or *chart* (or local chart) of M, each homeomorphism $\varphi_i\colon U_i \to \Omega_i$

is called a *coordinate map*, and its inverse $\varphi_i^{-1}\colon \Omega_i \to U_i$ is called a *parameterization* of U_i. For any point $p \in M$, for any coordinate system (U, φ) with $\varphi\colon U \to \Omega$, if $p \in U$, we say that (Ω, φ^{-1}) is a *parameterization of M at p*. The family $(U_i, \varphi_i)_{i\in I}$ is often called an *atlas* for M.

Remark: Intuitively, the family $(\varphi_i)_{i\in I}$ of homeomorphisms map each point and its neighborhood in M to the open 2-dimensional disc.

This approach is much more attractive from an algorithmic point of view, because it proposes the idea of "walking" on the surface of the polyhedron, along the edges from corner point to corner point. This is basically the three dimensional version of our C^1 algorithm. However, the additional element of these local parameterizations do complicate matters, as it is clear that the choice of these parameterizations is critical, and will greatly affect the outcome of the final surface shape quality. As there is ample freedom in how to map these local regions to the affine plane, a method that will ideally incorporate as much known surface geometry as possible and losing as little information as possible is nontrivial.

We have done some preliminary implementations to get a feeling of these local parameterizations. The local parameterizations are chosen to be as simple as possible, and in this cases, they even violate AG^1 constraints across the edges, that is, they have edge consistency problems. However, as we shall see in the examples below, the quality of the approximations are surprisingly good.

5.4.1 Tetrahedron

As show in Figures 5.10 and 5.11 below, the tetrahedron given in green wireframe is the input triangulated polyhedron, and the smaller, smooth tetrahedron in blue (also given in wireframe in the fourth figure) is the output AG^1 surface approximation of the input tetrahedron.

The AG^1 surface, although entirely contained within the original input tetrahedron, is nevertheless a fairly good smooth surface approximation of the original shape. The shrinkage is largely due to the fact that since interpolation is not required, lowering the surface into the interior allows the reduction of certain shape defects, which we will discuss in detail in Chapter 6, when actual algorithms for fitting AG^1 surfaces are presented.

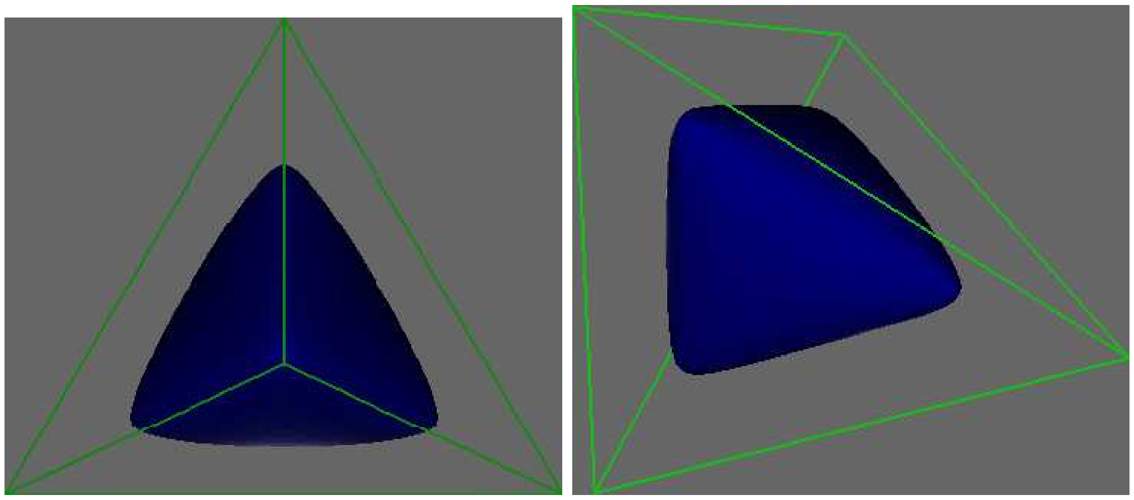


Figure 5.10: A smooth tetrahedron.

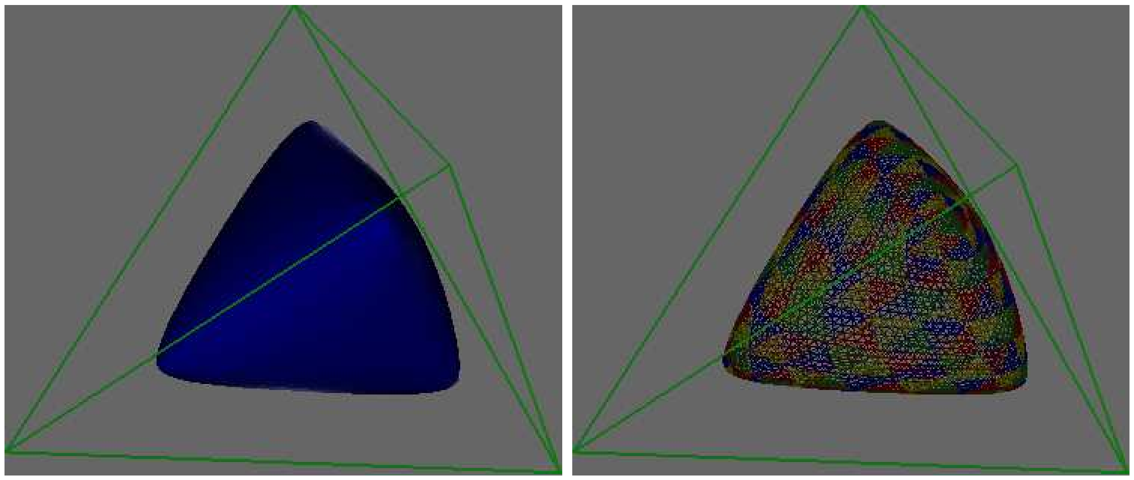


Figure 5.11: A smooth tetrahedron, ctd.

5.4.2 Icosahedron

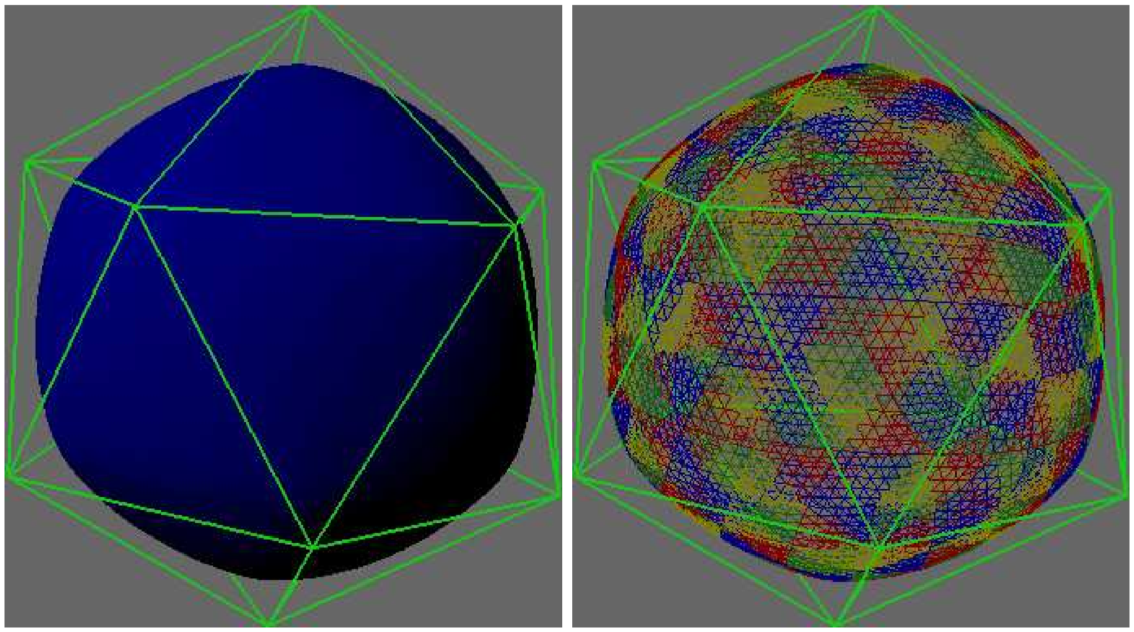


Figure 5.12: A smooth icosahedron

As stated before, the simplest possible reparametrizations maps were chosen for both the tetrahedron and the icosahedron. The corner region is flattened to a regular n-gon consisting of the n triangles that come together at the corner point. Because the tetrahedron and icosahedron are both regular polyhedra, the same number of triangles meet at all corner-point regions. That is, three triangles for each corner-point region for the tetrahedron and five for the icosahedron. The reparameterization maps will be the same for all corner-point regions, and each will be flattened to a equilateral triangle (for the tetrahedron), or a pentagon (for the icosahedron). The corner-point regions look as follows after reparameterization:

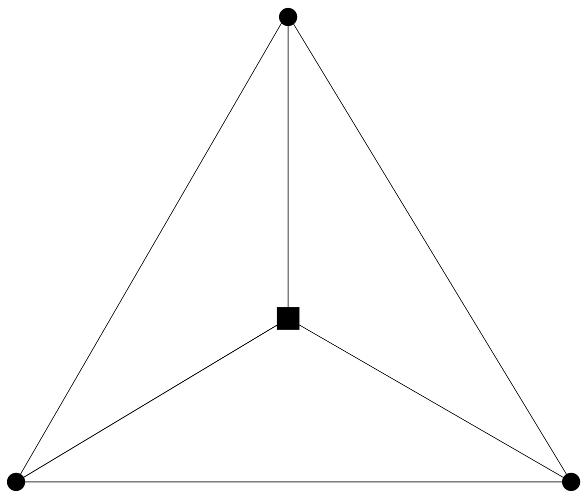
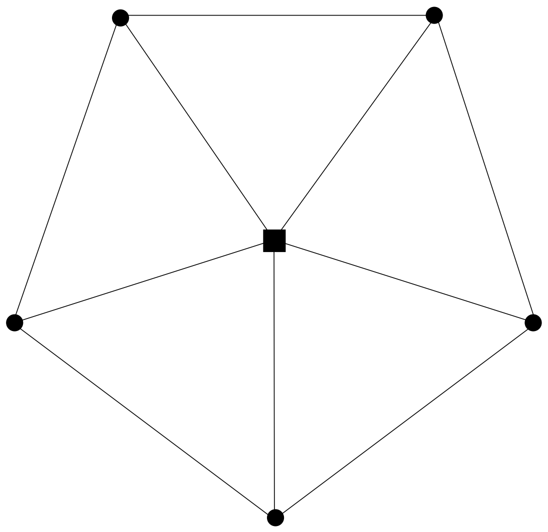



Figure 5.13: Corner-point region of tetrahedron   Corner-point region of icosahedron

Across the edges, we choose maps to form simple parallelograms, so that the propagations can take advantage of the fact that the midpoints of the diagonals agree. This is probably not the best choice of maps, and it causes clear violation of G^1 continuity constraints across the edges, because the three diamonds are not in any way similar to the two on either end of the edge, who are specified by the corner-point region maps. For example, an edge on the tetrahedron will look like this:

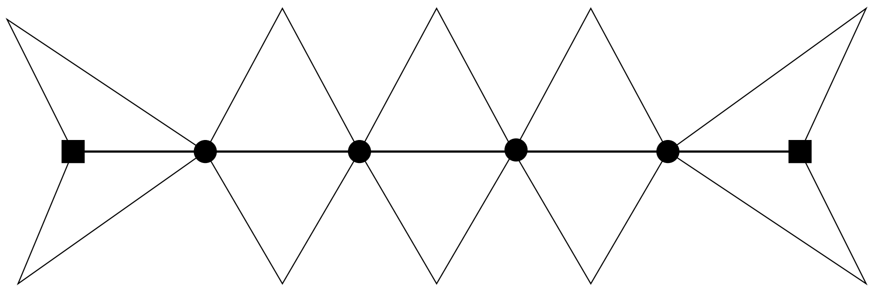


Figure 5.14: Edge maps of a tetrahedron.

Now we move on to develop an actual algorithm for fitting AG^1 surfaces for triangulated polyhedrons, and at the same time, keeping to our goal to retain as much incrementality and local control as possible.

Chapter 6

Designing AG^1 Triangular Spline Surfaces

The results of the preliminary experimentations in Chapter 5 show that even with simple-minded (and in this case, probably poor) choices of the reparameterization maps, the results are still quite good. Of course, the polyhedra we chose to fit our surfaces on are mathematical and regular, which definitely contributed towards the final surface quality.

6.1 Listing AG^1 equations

We would like to look more closely at the equations that dictate the AG^1 conditions around the corner region and across the edges. We believe that opening up the triangulated polyhedra, writing out systems of equations and attempting to solve/reduce them will bring crucial insights. While doing so does not contribute to the construction of an actual algorithm, as the approach is quite global, it nevertheless offers the important proof of existence for the reparameterization maps. We hope that the system of constraints might even give us some information on how these reparameterization maps might look like.

6.1.1 AG^1 equations on a tetrahedron

We choose to start with the tetrahedron because it is the smallest triangulated polyhedron, and therefore has the fewest number of total equations around the corners and across the edges.

Let $(\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a tetrahedron, where $\mathcal{V} = \{a, b, c, d\}$, $\mathcal{E} = \{ab, bc, cd, ac, bc, bd\}$ and $\mathcal{T} = \{abc, adb, bdc, cda\}$. Let the corner control points be indexed by the vertices of the tetrahedron they are associated with. That is, $z_a \to a$, $z_b \to b$ and so on.

Figure 6.1 shows such a tetrahedron and after it has been opened and laid flat in the plane, with proper identifications. The closed tetrahedron drawing does not accurately reflect the placements of the control points associated with the vertices, i.e. z_a, z_b, z_c and z_d, under AG^1 constraints, as z_a should be coplanar with z_1, z_2 and z_3, and z_b coplanar with z_7, z_8 and z_9 and so on. However, we do not yet know the corner point's affine relation with respect to its star. For the purpose of listing equations, we find it easier to imagine the control points when drawn this way.

Recall Lemma 5.3.3 from Chapter 5, which states that control points attached to any reparameterized triangular face ${T_j}'$ are not affected by the reparameterization maps. Thus, although not drawn in Figure 6.1, we can list the equations after reparameterization with the same control points. This greatly simplifies things.

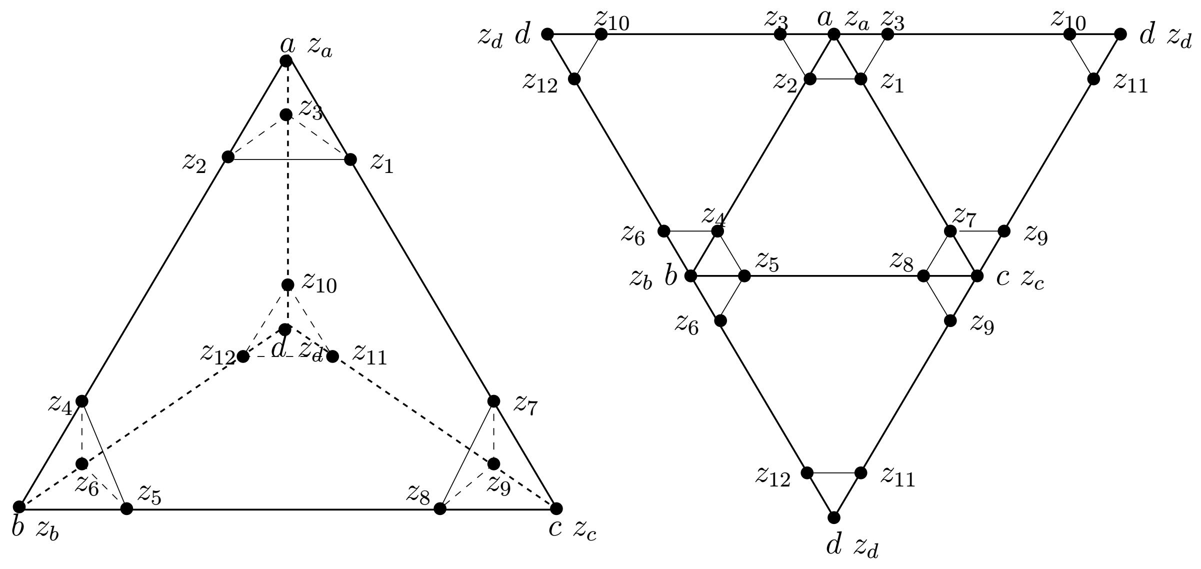


Figure 6.1: An opened-up tetrahedron

AG^1 constraints state that all diamonds across any edge must observe the same affine relations (because they are given by the same domain diamond). More precisely, given the domain diamond $\langle p, q, r, s\rangle$ consisting of two adjacent triangles Δprs and Δqrs sharing the edge (r, s), if p, q, r, s has the relation $\lambda p + \mu r + \nu s = q$ with respect to frame Δprs, then any four control points forming a diamond across the edge (r, s) must respect the same affine relations given by λ, μ and ν. This condition is in fact Condition 3.5 given in Section 3.2 as the necessary and sufficient polar form condition for C^1 continuity.

Thus, we can index such relations with each edge, as $(\lambda_{e_i}, \mu_{e_i}, \nu_{e_i})$. Now we proceed to list all the AG^1 equations for the corner regions. For each corner point, there is one equation for each edge that is adjacent to that corner point. For example, for vertex a, and its corresponding corner control point z_a, there are three equations:

$$z_a = \lambda_{ac} z_3 + \mu_{ac} z_1 + \nu_{ac} z_2, \tag{6.1}$$

$$z_a = \lambda_{ab} z_1 + \mu_{ab} z_2 + \nu_{ab} z_3, \tag{6.2}$$

$$z_a = \lambda_{ad} z_1 + \mu_{ad} z_3 + \nu_{ad} z_2. \tag{6.3}$$

Similarly, the equations for the other three corner-point regions are:

$$z_b \begin{cases} z_b = \lambda_{bc} z_4 + \mu_{bc} z_5 + \nu_{bc} z_6, \\ z_b = \lambda_{bd} z_4 + \mu_{bd} z_6 + \nu_{bd} z_5, \\ z_4 = \lambda_{ab} z_5 + \mu_{ab} z_b + \nu_{ab} z_6, \end{cases}$$

$$z_c \begin{cases} z_c = \lambda_{cd} z_8 + \mu_{cd} z_9 + \nu_{cd} z_7, \\ z_4 = \lambda_{ac} z_9 + \mu_{ac} z_c + \nu_{ac} z_8, \\ z_5 = \lambda_{bc} z_7 + \mu_{bc} z_c + \nu_{bc} z_9, \end{cases}$$

$$z_d \begin{cases} z_{10} = \lambda_{ad} z_{11} + \mu_{ad} z_d + \nu_{ad} z_{12}, \\ z_{11} = \lambda_{cd} z_{12} + \mu_{cd} z_d + \nu_{cd} z_{10}, \\ z_{12} = \lambda_{bd} z_{10} + \mu_{bd} z_d + \nu_{bd} z_{11}. \end{cases}$$

In addition, there are also the six equations (one per edge) stating the affine relations of the λs, μs and νs:

$$\lambda_{ab} + \mu_{ab} + \nu_{ab} = 1,$$

$$\lambda_{ac} + \mu_{ac} + \nu_{ac} = 1,$$

$$\lambda_{ad} + \mu_{ad} + \nu_{ad} = 1,$$

$$\lambda_{bc} + \mu_{bc} + \nu_{bc} = 1,$$

$$\lambda_{bd} + \mu_{bd} + \nu_{bd} = 1,$$

$$\lambda_{cd} + \mu_{cd} + \nu_{cd} = 1.$$

Thus, once the z_i are fixed, there are 18 equations and 18 variables, and as we will prove, the above system has a unique solution. However, the solution turns out to be highly unsatisfactory.

Because we have a tetrahedron, there are only three control points in the star of each corner point, which forces the same three points to be the affine frame for every equation written for that corner point region. See Equations 6.1, 6.2 and 6.3. This forces

$$\lambda_{ab} = \mu_{ac} = \lambda_{ad},$$

$$\nu_{ac} = \mu_{ab} = \nu_{ad},$$

$$\lambda_{ac} = \nu_{ab} = \mu_{ad}.$$

Together with

$$\lambda_{ab} + \mu_{ab} + \nu_{ab} = 1,$$

$$\lambda_{ac} + \mu_{ac} + \nu_{ac} = 1,$$

$$\lambda_{ad} + \mu_{ad} + \nu_{ad} = 1,$$

it is clear that $\lambda_{ab} = \mu_{ab} = \nu_{ab} = \lambda_{ac} = \mu_{ac} = \nu_{ac} = \lambda_{ad} = \mu_{ad} = \nu_{ad} = \dfrac{1}{3}$.

Substituting into any of the Equations 6.1, 6.2 or 6.3, we get

$$z_a = \frac{1}{3}z_1 + \frac{1}{3}z_2 + \frac{1}{3}z_3. \tag{6.4}$$

That is, z_a is the center of gravity of z_1, z_2 and z_3.

Now we look at the equations of the other corner regions. Recall the equations for corner point z_b:

$$z_b = \lambda_{bc}z_4 + \mu_{bc}z_5 + \nu_{bc}z_6, \tag{6.5}$$

$$z_b = \lambda_{bd}z_4 + \mu_{bd}z_6 + \nu_{bd}z_5, \tag{6.6}$$

$$z_4 = \lambda_{ab}z_5 + \mu_{ab}z_b + \nu_{ab}z_6. \tag{6.7}$$

Notice that since all diamonds across the same edge must respect the same affine relations given by the domain diamond, we are not able to pick only the star points as affine frames for the other three corner region. For example, around z_b, edge ab's frame has already been locked in by z_a's Equation $z_a = \lambda_{ab}z_1 + \mu_{ab}z_2 + \nu_{ab}z_3$ 6.2, therefore the z_b Equation corresponding to edge ab has to be written as $z_4 = \lambda_{ab}z_5 + \mu_{ab}z_b + \nu_{ab}z_6$ 6.7. Substituting in λ_{ab}, μ_{ab} and ν_{ab}, we get the following affine relations associated with edge bc, $z_4 = \frac{1}{3}z_5 + \frac{1}{3}z_b + \frac{1}{3}z_6$, that is

$$z_b = 3z_4 - z_5 - z_6. \tag{6.8}$$

Similar to what happens around z_a, this also locks in the affine relations associated with edges bc and bd through Equations 6.5 and 6.6 as

$$\lambda_{bd} = \lambda_{bc} = 3$$

$$\mu_{bc} = \nu_{bd} = -1$$

$$\nu_{bc} = \mu_{bd} = -1.$$

These computations can also be conducted around corner-point regions z_c and z_d, leading to similar results. Clearly, z_b's position with respect to its star points z_4, z_5 and z_6 as given by Equation 6.8 is unreasonable. Corner point z_b lies way out of the triangle formed by z_4, z_5 and z_6, which will lead to severe surface defect. In fact, once corner point z_a is forced to be $\frac{1}{3}$ of z_1, z_2 and z_3 (Equation 6.4), the corner point region is then flattened to an equilateral triangle (incidentally, it coincides with the guess we took in our preliminary experiments, see Figure 5.13):

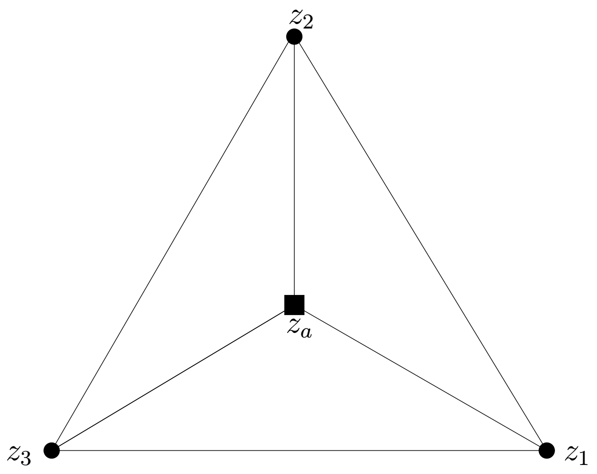


Figure 6.2: Corner-point region of z_a

This forces highly undesirable affine shapes to be propagated along all incident edges ab, ac and ad, that is, instead of the general diamond-shaped quadrilateral consisting of two acute triangles, we have an arrow-shaped quadrilateral consisting of two obtuse triangles. The resulting edge maps are indeed strangely shaped, as shown in Figure 6.3.

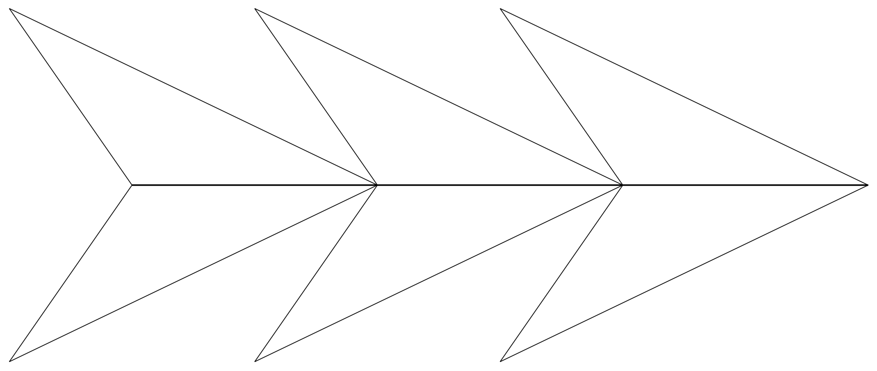


Figure 6.3: Edge maps for edges ab, ac and ad.

The edge maps then in turn force undesirable maps in the corner-point regions on the other end of these edges, namely, those of z_b, z_c and z_d.

In conclusion, we can obtain an AG^1 surface based on a tetrahedron, but only with a highly undesirable surface shape, and the resulting surfaces will not represent a tetrahedron in any reasonable way.

6.1.2 AG^1 in general

The results we obtained in Section 6.1.1 were highly unsatisfactory. However, being the smallest and simplest triangulated polyhedron, the tetrahedron is also the most restrictive. The main problem is that on a tetrahedron each vertex only has three adjacent edges, and as we have seen in the last section, because there are only three star points in a corner-point region, AG^1 constraints then force a unique and very unfortunate corner-point region map, which in turn propagates along the adjacent edges to affect other corner-point regions.

Now the question is what will happen if we have a more general triangulated polyhedron with more adjacent edges to each vertex. In general, given a triangulated polyhedron $(\mathcal{V}, \mathcal{E}, \mathcal{T})$, letting $|\mathcal{E}|$ denote the number of edges, we write two equations per edge, one for each corner-point region at each end of the edge, and an extra equation per edge for the affine constraints of the λs, μs and νs.

We chose to write equations this way because this system gives privileges to the corner control points over the rest. This allows for freedom around the corner point region, and in turn local control and an incremental algorithm, we hope.

The above system has a total of $2|\mathcal{E}| + |\mathcal{E}| = 3|\mathcal{E}|$ equations. There are three unknowns per edge, one each of λ, μ and ν, and therefore also $3|\mathcal{E}|$ total unknowns. This means that the system has a unique solution in the generic case. In all cases there is at least some solution, although not necessarily satisfactory, as we noticed from the case of the tetrahedron.

Now, if we assume there are more than three adjacent edges, there are then more than three star points around the corner-point region. We can write the equations in the same fashion as we have done for the tetrahedron in Section 6.1.1.

$$\begin{aligned}
z_{v_i} &= \lambda_{e_{i(i+1)}} z_1 + \mu_{e_{i(i+1)}} z_2 + \nu_{e_{i(i+1)}} z_3 \\
z_{v_i} &= \lambda_{e_{i(i+2)}} z_2 + \mu_{e_{i(i+2)}} z_3 + \nu_{e_{i(i+2)}} z_4 \\
&\ \ \cdots \\
z_{v_i} &= \lambda_{e_{i(i+n)}} z_n + \mu_{e_{i(i+n)}} z_{n+1} + \nu_{e_{i(i+n)}} z_{n+2}
\end{aligned}$$

Because there are more than three star points, there is no unique solution to this system. However, recall Lemma 3.3.2 from Chapter 3, which states that any three affinely independent points form an affine basis, and any other point in the corner point region is completely determined by a unique affine combination of these three points. That is, once we determine z_{v_i} with respect to any three of its star points, this locks down the entire corner-point region, which in turn locks down the affine relations for all adjacent edges, which then propagates through to the other corner-point regions of the adjacent edges and so on.

What this means is that we in fact only have some freedom in the first corner-point region we chance upon. Once that is determined, it locks down everything. This is not very good news either. We would need more flexibility than that if we were to have local control. In particular, we need to be able to determine each corner-point region map independently and the propagate the edge.

As we have seen so far, because the three edge diamonds need to respect the the same affine relations as the two one each end of the edge which are part of the corner-point regions, we have excessive propagation and resulting in all corner-point regions being locked in. The only alternative then is to relax AG^1 even further, so that the three edge diamonds do not have to respect the same affine relations as the two end diamonds.

We have now in fact gone back to the kind of maps we tried in our preliminary tests (see Figure 5.14). We had hoped for better maps, but AG^1 constraints are too restrictive to allow maps with more freedom.

6.2 Pseudo-AG^1 with local methods

We now relax AG^1 constraints to allow the three edge diamonds no longer have to agree with the two end diamonds that are part of the corner-point region on each end of an edge. We do, however, still require the three edge diamonds to respect the same ratios. We call this continuity pseudo-AG^1.

This allows enough freedom to have individual reparameterization maps applied to each corner-point region, as well as individual edge reparameterization maps applied to each edge.

In contrast to the C^1 work we have done, in the AG^1 case, we have much more freedom in the local reparameterizations. In C^1, because we start from a template triangulation, we have more information, and locally, much is dictated by the layout of the triangulation. However, in the case of a triangulated polyhedron, when we reparameterize, in essence we have to come up with a local "flattening" of the corner-point regions first. It is clear that the choice of how this "flattening" is done is of crucial importance. It will greatly affect the shape of final the surface. Also, unfortunate choices might make the across-the-edge reparameterizations much more difficult.

6.2.1 Reparameterization maps: corner-point regions

In a corner-point region, the corner point and all of its star points are necessarily coplanar. The plane onto which we send the corner point and its start points is called the *corner plane*. The first choice to make is where to fit the corner plane. For now, we assume that all the star points already have assigned coordinates that give a reasonable first approximation of their final locations. We will see how these coordinates are assigned in Subsection 6.2.4.

The corner plane

Take any corner point region, let v_a be the vertex, z_a its corresponding corner point and $z_1 \ldots z_n$ be z_a's star points.

- **Approach** 1

  We begin by taking the center of gravity $\mathcal{C}$ of all star points, that is,

  $$\mathcal{C} = \frac{\sum_{i=0}^{n} z_i}{n}.$$

  Draw a line from v_a to $\mathcal{C}$, we take any plane $\mathcal{P}$ perpendicular to this line as our corner plane. There is freedom to slide this plane along $z_a\mathcal{C}$ according to some shape parameter. Project z_a onto $\mathcal{P}$ gives us z_a and projections of $z_1 \ldots z_n$ onto $\mathcal{P}$ are also taken to obtain final values of all star points, as shown in Figure 6.4.

  However, this approach turns out to unsatisfactory, because unfortunate locations of the center of gravity with respect to the vertex can lead to undesired tilting of the corner plane, which will result in surface defect.

- **Approach** 2

  Next we take the average of the normals of all faces adjacent to the vertex, and use a plane $\mathcal{P}$ perpendicular to this average of face normals as the corner plane.

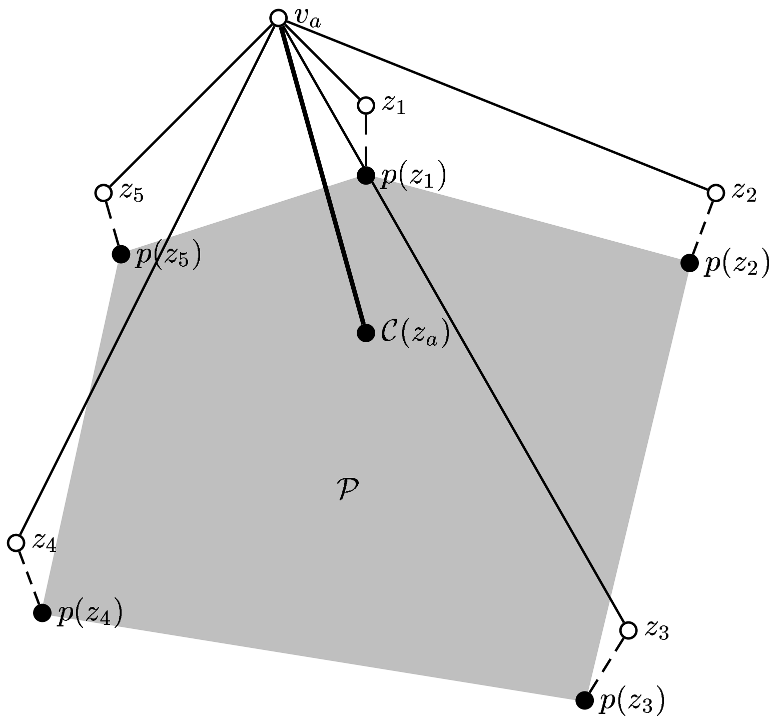


Figure 6.4: Corner plane by taking centroid of star points

This method gives better position of the corner plane. However, if some of the adjacent faces happen to be on the same plane, the average of the normals tend to favor these faces, because there is more weight. This effect is especially prominent when there are a lot of consecutive small (thin) coplanar faces in a row.

- **Approach** 3

  We go on to take the weighted average of the normals $\mathcal{N}$ of all faces adjacent to the vertex. The weight is assigned according to the area of the face. That is, if $f_1, \ldots, f_n$ are the adjacent faces and $N(f_i)$ denotes the normals of f_i and

$A(f_i)$ the area of f_i, then

$$\mathcal{N} = \sum_{i=0}^{n} \frac{A(f_i)}{\sum_{i=0}^{n} A(f_i)} * N(f_i).$$

Again, we use a plane perpendicular to $\mathcal{N}$ as our corner plane.

The position of the corner plane

Now that we have chosen our corner plane, there are two questions remaining. First, because there is an infinite number of planes perpendicular to a given vector (weighted normal average), there is a choice of where to "hang" this plane along the vector. A reasonable position seems to be somewhere between the vertex and the centroid of the star points.

It is clear that the closer the corner plane is placed towards the vertex, the sharper the final surface will be around the vertex. Conversely, the further away the corner plane is placed from the vertex, the more "rounded" the final surface will be around that vertex. Therefore the decision of the corner plane position with respect to the vertex seems to call for a shape parameter, which indicates, for the lack of a better word, the "spikiness" of the final surface.

Currently we start out by positioning the corner plane half way between the vertex and the centroid of its star points. However, in the event of a very sharp vertex, placement closer to or even beyond the centroid might be considered as well. Also, the vertical position of the plane is greatly influenced by the existence of edge undulations, which will be discussed in Subsection 6.2.3.

Projecting onto the corner plane

The second decision we need to make is how to send the vertex and the star points onto the corner plane. A simple projection is of course the first thing that comes to mind and was also the first thing we tried. However, projections do not preserve lengths and unfortunate positions of the corner plane with respect to the control points can lead to bizarre final placements.

The position of z_a in the corner plane from v_a in space is of great importance. Instead of projecting v_a, we need to send v_a to a centered position on the corner plane with respect to the final placements of its star in the corner plane. Control point z_a also needs to be placed inside of the polygon consisting of the final placements of z_i in the corner plane.

Normally, sending v_a to the intersection point of $\overrightarrow{v_a\mathcal{C}}$ and the corner plane is a good choice. However, if the projection of $\mathcal{C}$ onto the corner plane falls outside of the polygon P consisting of the projections of all z_i onto the corner plane, then we need to pick another point that is inside P and is also relatively centered. The method we use to find this point is described below and illustrated in Figure 6.5.

If the aforementioned happens, then P must be non-convex. We find the first non-convex vertex v (by walking along the boundary of P counter-clock wise and finding the first right-turn). Extend both edges that are adjacent to v until they each intersect with P at v_1 and v_2. These two extended edges cut P into three separate parts. Let P equal the the portion of P that contains both $\overrightarrow{v_1v}$ and $\overrightarrow{vv_2}$. Recurse on P until P is convex. Now we can send v_a to the centroid $\mathcal{C}'$ of this new convex P, which is contained in the original P.

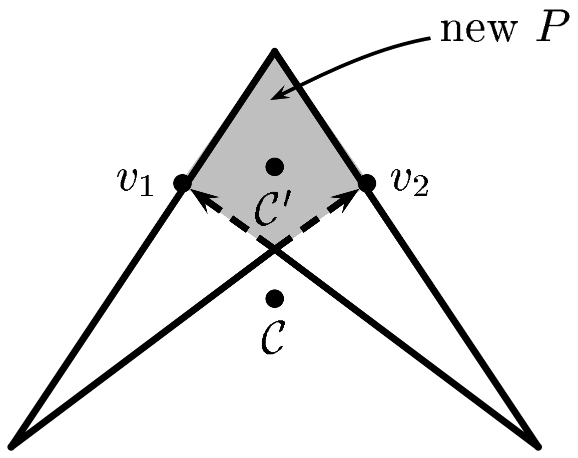


Figure 6.5: $\mathcal{C}$ lies outside of P, $\mathcal{C}'$ is the new choice.

fThen we recover the lengths after projecting z_i onto the corner plane. We first calculate the lengths of all star points to the vertex. Then we lengthen or shorten the vector $\overrightarrow{z_a p(z_i)}$ according to the corresponding length $|\overrightarrow{v_a z_i}|$. Depending on the actual input mesh, this can be also done to maintain lengths proportionally.

We also try to recover part of the affine relations. It is not possible to recover all of the original affine relations, as we are losing a dimension. However, the angles between the adjacent edges around the vertex can at least be proportionally maintained. That is, we position the $p(z_i)$ around z_a so that the angle $\mathcal{A}_{p(z_i)z_a p(z_{i+1})}$ between $\overrightarrow{z_a p(z_i)}$ and $\overrightarrow{z_a p(z_{i+1})}$ is given by

$$\mathcal{A}_{p(z_i)z_a p(z_{i+1})} = \frac{\mathcal{A}_{z_i v_a z_{i+1}}}{\sum_{i=0}^{n} \mathcal{A}_{z_i v_a z_{i+1}}} * 360.$$

In fact, if we do the angle adjustments before the length adjustments, there is no need to do any projections for the star points z_i. The angle adjustments will place the $p(z_i)$ in the plane in proper directions around z_a. The length of the vectors can then be determined. Instead of simply flattening out the corner-point region via projections, what we achieve through the angle and length adjustments is analogous to an umbrella opening, that is, keeping the lengths rigid and the angles proportional.

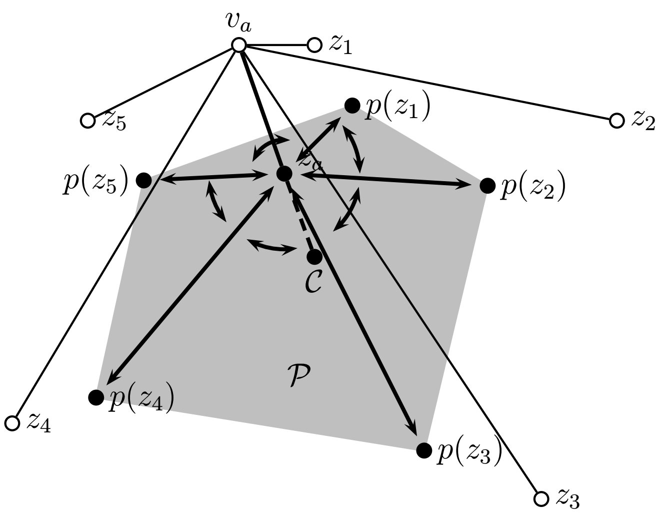


Figure 6.6: Corner plane adjustments, lengths are adjusted proportionally

6.2.2 Reparameterization maps: edges

Now we move on to edge reparameterization maps. The three edge diamonds no longer need to agree with the two that are part of the corner-point regions, but they do need to agree with each other. From the point of view of the final surface shape, any map we use across the edge needs to at least keep the diamonds convex. The arrow-shaped diamonds we have seen in the case of the tetrahedron will clearly destroy any hope of a reasonable shape approximation.

The easiest convex map is the one we used in our experiments with the tetrahedron and the icosahedron, that is, to have the diamonds form parallelograms. This leads to easy propagation computation.

We have also experimented with applying the average of the ratios of the two diamonds at each end of the edge. We did not see significant improvement over the parallelogram map, and in fact, in the cases where one or both of the diamonds have unreasonable ratios, the averaging only made things worse.

We believe that in the event of two clearly conflicting and unreasonable ratios like the corner-point regions of the tetrahedron, sticking to the parallelogram map on the edges is as good a solution as any. However, if both diamonds are at least convex, then a middle shape between the two might be the best ratio to keep. However, the problem of finding the best middle shape of two convex quadrilaterals does not seem to offer easy immediate solutions. For the time being, we have chosen the parallelogram map as edge maps.

6.2.3 Undulations

The edge diamonds must be coplanar. For any diamond, if the plane formed by the three prescribed control points (the affine frame) is not parallel to the edge it sits across, there is unavoidable “dipping” or “raising” of the fourth point, see Figure 6.7. In general, dipping happens more often, as the star points at the end of the edge tend to go higher out of the surface of the polyhedral mesh then the two inner points at each side of the edge. The sharper the original vertex, the more so.

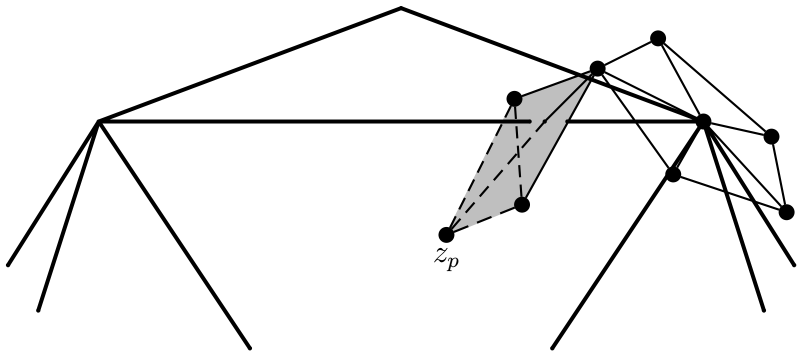


Figure 6.7: Control point z_p is dipping.

One solution is to raise the inner points higher out of the triangulated polyhedron or lower the star points/corner plane into the interior of the polyhedron. Recall that we do have the freedom to slide the corner plane higher or lower. See Subsection 6.2.1. Inner points need to be raised substantially to counteract the dipping, and at that point, they will cause the ballooning of the inner surface of the triangular patch. Therefore, lowering the star points are necessary.

How much to lower or raise the corner plane depends on the sharpness of the vertex in the original mesh. For each vertex, we calculate the angles formed by two adjacent faces across each edge adjacent to that vertex. Then we take the average of these angles over all edges adjacent to the vertex. This average angle is a fair indicator of the sharpness of the vertex. In case of many crowded edges with flat angles and a few edges with sharp angles, the average will not indicate the sharpness correctly. In that case we also measure the distance between the vertex and the center of gravity of the star points to make sure. Clearly, the longer the distance, the sharper the corner.

Currently we raise the plane if the average angle is greater than $\frac{4}{5}\pi$, and lower otherwise. We also raise and lower proportional to the angle.

If lowering the corner plane is not enough to eliminate the undulations along the edges, we can also shrink the star region proportionally. This also brings the star points towards the interior of the triangulated polyhedron.

Aside of the global raising or lowering, one can make more local changes to fine-tune the surface shape. Once the star points are determined, we have the freedom to prescribe two inner points for each of the two edge diamonds containing a star point as the third prescribed point. We can calculate the average of the normals of the two faces adjacent to this edge, thereby obtaining the cross-edge normal, and then assign the two inner points so that together with the star point, they form an affine frame that is perpendicular to the cross-edge normal.

6.2.4 First approximation of control points

Because the final surface is to approximate the given triangulated polyhedron, we first calculate a rough approximation of the locations of certain control points based on the input polyhedral mesh. We assign a simple weighted average of the vertices of the polyhedral mesh to these control points, according to each control point's polar value.

Recall that every control point has a corresponding polar value which is a triple (i, j, k), $i + j + k = m$, where m is the degree of the patches. For each triangular face of the input triangulated polyhedron, given the coordinates of the three vertices, we simply obtained the coordinates of any control point by blending the coordinates of these three vertices using that control point's polar values as weights.

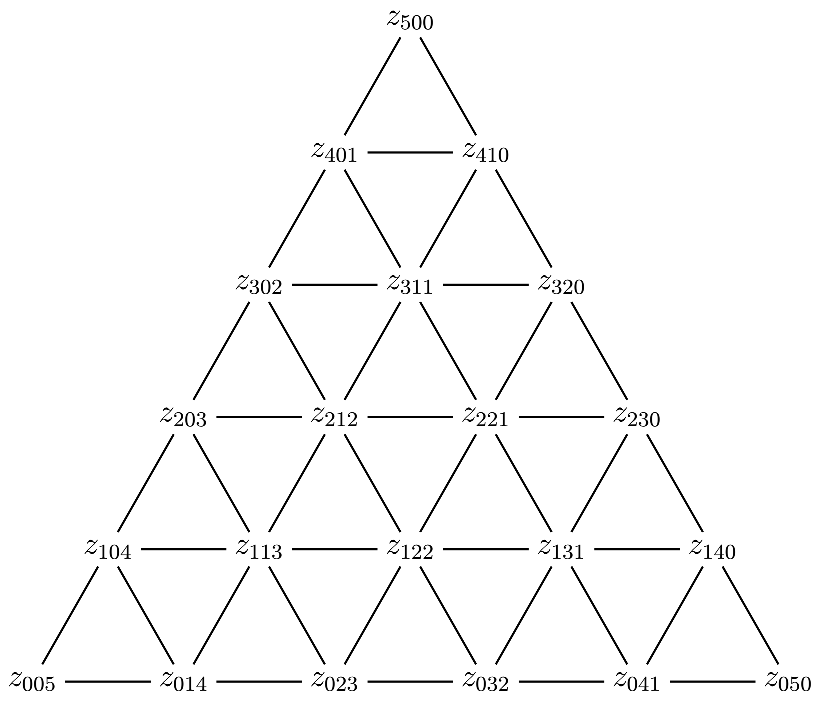

z_{500}
z_{401} z_{410}
z_{302} z_{311} z_{320}
z_{203} z_{212} z_{221} z_{230}
z_{104} z_{113} z_{122} z_{131} z_{140}
z_{005} z_{014} z_{023} z_{032} z_{041} z_{050}

Figure 6.8: The polar value labeling of quintic Bézier control points

Specifically, given $v_1(v_{1x}, v_{1y}, v_{1z})$, $v_2(v_{2x}, v_{2y}, v_{2z})$, $v_3(v_{3x}, v_{3y}, v_{3z})$, as the three vertices and their corresponding coordinates and assume that v_1 corresponds to the control point with polar values $(m, 0, 0)$, v_2 corresponds to the control point with corner values $(0, m, 0)$ and v_3 the polar values $(0, 0, m)$, the coordinates of any control point z_{ijk} in the patch with the polar value $(i, j, k),\ i+j+k = m$ are given as follows:

$$z_{ijk_x} = \frac{i}{m}v_{1x} + \frac{j}{m}v_{2x} + \frac{k}{m}v_{3x},$$

$$z_{ijk_y} = \frac{i}{m}v_{1y} + \frac{j}{m}v_{2y} + \frac{k}{m}v_{3y},$$

$$z_{ijk_z} = \frac{i}{m}v_{1z} + \frac{j}{m}v_{2z} + \frac{k}{m}v_{3z},$$

As we mentioned before, only a certain set of control points need to be assigned by these formulae. The purpose of this first approximation is so that we have something to start with when working on our reparameterization maps. Thus, only control points who need to have prescribed values as called for by the reparameterization maps should be assigned. This includes all the star points, inner points, and half of the flip points.

All control points generated this way lie directly on the surface of the triangulated polyhedron. To improve surface shape and prevent undulations, the control points need to be moved off the surface of the triangulated polyhedron. The star points are moved against the direction of the average of the normals of the two faces adjacent to the edge it sits on. This makes the final plane-placed star smaller, and thereby causes less dipping. Inner points are moved in the direction of the average of the normals of the two faces adjacent to its residing face. Flip points are moved in the direction of the normal of its residing face.

We move the star points proportional to the length of their residing edges, the inner and flip points proportional to the area of their residing faces.

6.2.5 Pseudo-AG^1 surfaces

Using the approaches explaining in the last few subsections, we fitted pseudo-AG^1 surfaces on triangulated polyhedra. The polyhedra used in implementation and testing are built from scratch by hand, and are completely arbitrary.

An arbitrary polyhedron

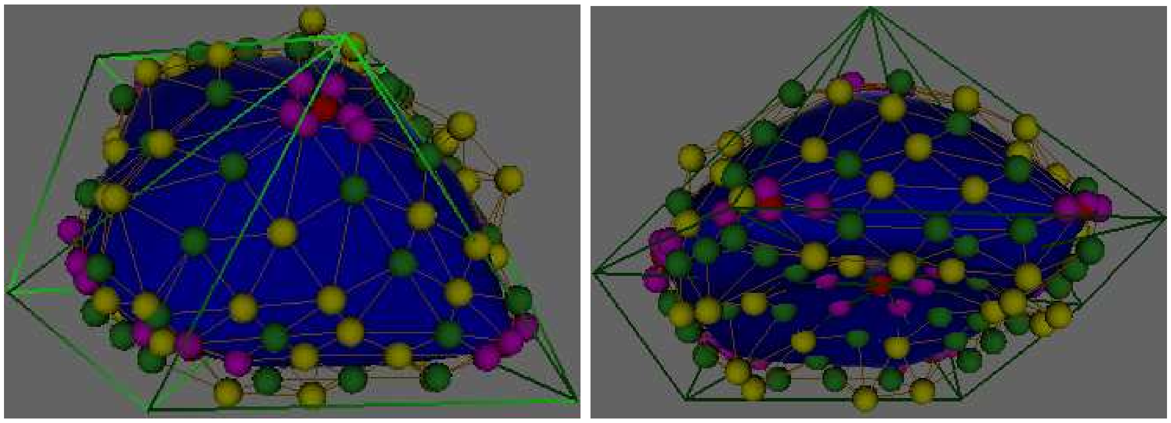


Figure 6.9: A smooth polyhedron with its control net

In the above figure, an arbitrary polyhedron is fitted with a pseudo-AG^1 surface. we can see the control net with all the control points whose placements determine the final surface shape. The corner points are shown red, the star points shown magenta, the control points that we have freedom to prescribe are green and the propagated ones yellow.

We will show a few more figures of the same surface.

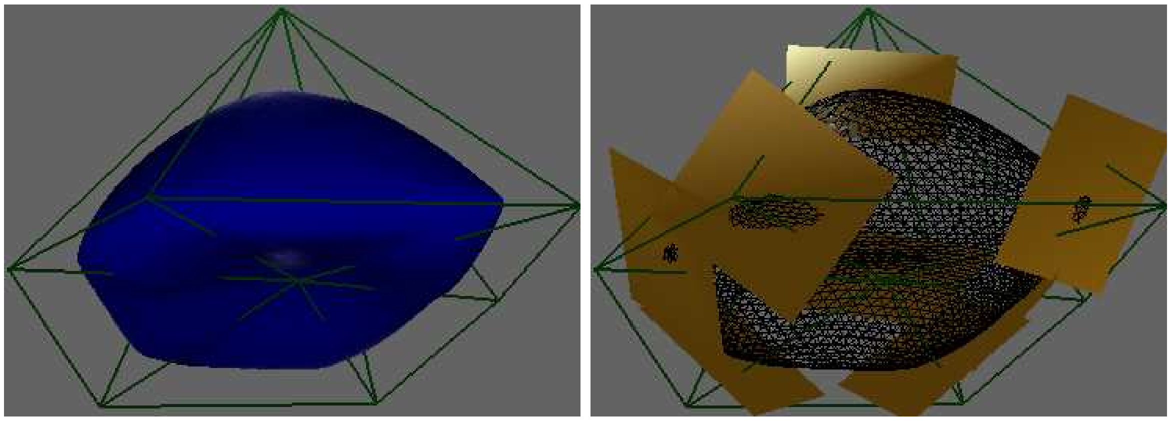


Figure 6.10: A smooth polyhedron with its corner planes

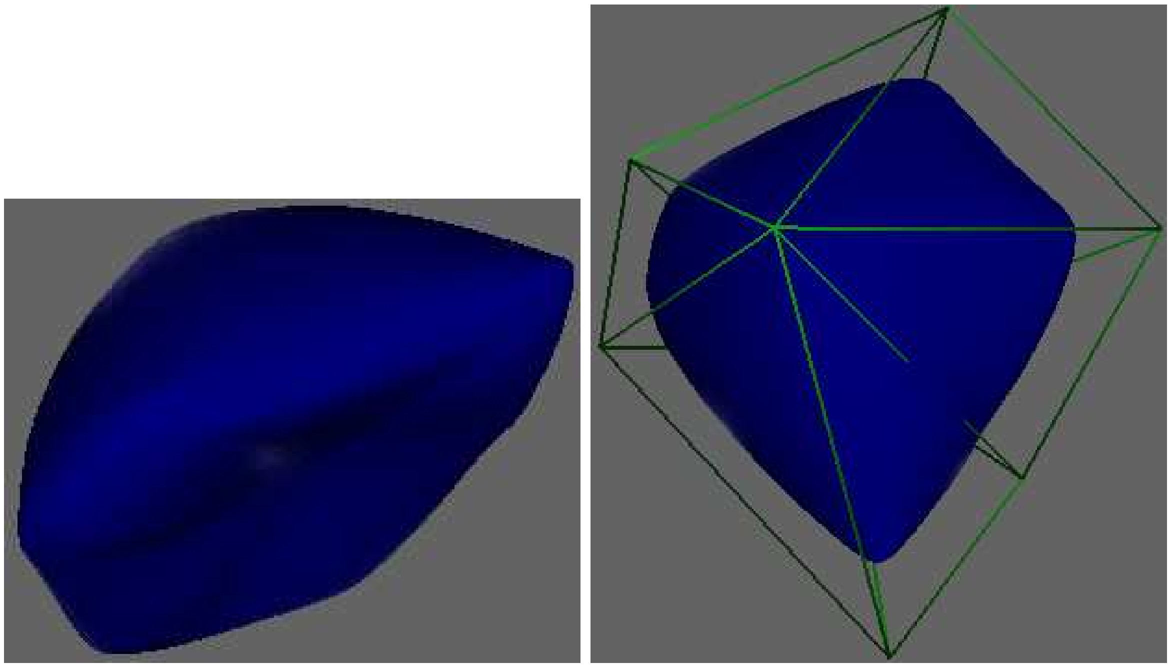


Figure 6.11: A smooth polyhedron, ctd.

In Figure 6.10, the same surface is shown with the corner planes and corner plane positions the program picked according to our algorithm. To keep the picture from being too busy, we did not show the control net again. However, if the net were shown, all the corner points and its star points will lie on their common corner plane, respectively.

A bigger example

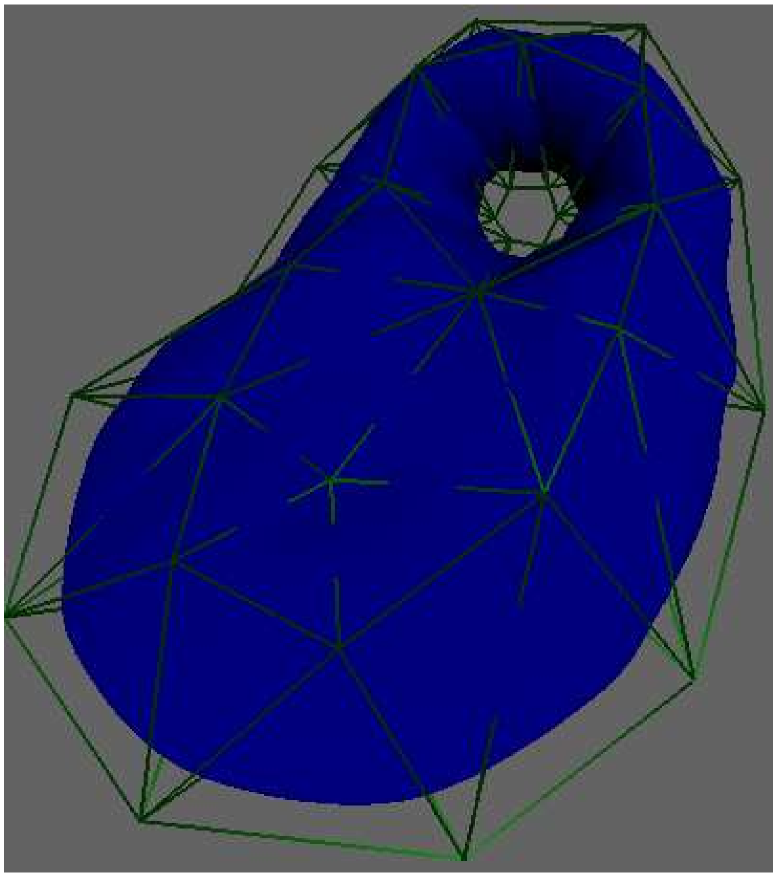


Figure 6.12: Another smooth polyhedron

This example fits a pseudo-AG^1 surface onto a polyhedron, which has 43 vertices and 87 faces. The next figure shows the same surface with its control net.

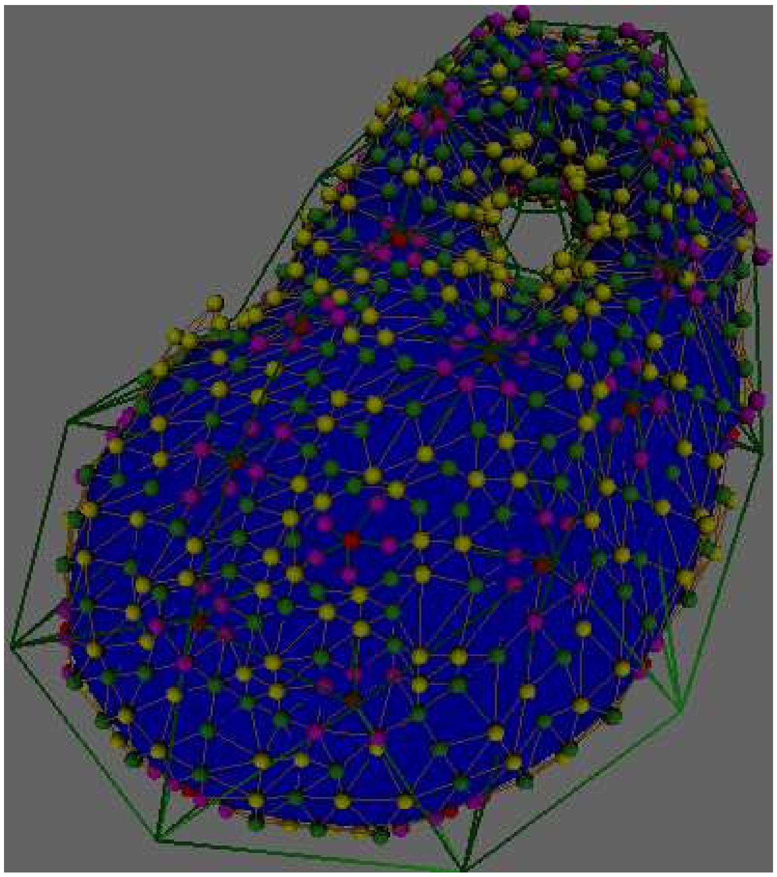


Figure 6.13: Another smooth polyhedron, with control net

Chapter 7

Implementation

An implementation of this algorithm has been carried out using C, OpenGL and the xforms library on an O2 SGI. The xforms library is used mainly to bring up a GUI that displays the template triangulation, and allows the user to prescribe the control points according to the algorithm. The propagation of the control points and subdivision evaluation of the surface thereafter are carried out by C routines. The final display of the resulting surface is done in OpenGL.

Some work has gone into implementing the algorithm and the display controls, therefore we will also discuss the implementation in some detail. The implementation itself is algorithmically straight-forward, but somewhat of a challenge in data structure and book-keeping. The main problem is that although the algorithm is easily understood while drawn on paper, where the relative positions of all control points in different control nets are in clear view and are geometrically intuitive, we do not have the advantage of the automatic knowledge (at the glance of an eye) of such geometric positional information regarding the control points when it comes to actual programming. We only have a set of scattered points with given coordinates or polar values. Thus, the necessary geometric information must be preprocessed and encoded into the data structure ahead of time.

7.1 Program outline

All of the work is done on triangulations. The input to the program is the template triangulation, given in the form of a list of two dimensional vertices in its x, y coordinates, and a list of faces (triangles), each consisting of three vertices in the vertex-list. The program then reads in the input, creates the data structures to store the vertex, edge and face information given in the input triangulation. Then the program proceeds to generate control points (polar values) associated with each triangular face, and store the control points and their geometric information necessary for the algorithm in another data structure. The xforms library is then used to provide a canvas with the input triangulation and its associated control points drawn. At the next step, either the user directs the prescription of the set of control points specified according the algorithm, or, in the case where the template triangulation is too large, the control points are prescribed automatically. We will look at this automatic method of prescribing control points in a bit more detail later. Notice that although some control points need not be prescribed in the sense that they do not affect continuity constraints, e.g. the ear points, in the reality of generating a surface, those control points need to have values. Therefore they need to be prescribed as well. Propagation is then carried out to compute all remaining control points. Once all control points have values, the program proceeds to subdivide to an appropriate “smooth” level. Then the resulting surface is displayed in OpenGL.

7.2 Data structures

There are five main components of the data structure:

1. the vertex list
2. the edge list
3. the face list
4. the control points (polar values) list
5. the control nets tree

The first three are set up when the input template triangulation is read. The control points list is built up a bit later, and the control nets tree is built up for the purpose of recursive subdivision.

7.2.1 The vertex list

This is a doubly-linked list that is used to store all information about the vertices of the template triangulation. Each `struct` in the link list contains a double array of size 2, in which the two-dimensional coordinates of that particular vertex are stored as they are read in by the program. Also, a double array of size 3, in which the three-dimensional coordinates of the control point associated with that particular vertex is also present, as well as many flags for debugging and other purposes. A wealth of other geometric information regarding that particular control point is also stored, as its adjacent edges, its star, if a corner point, its two incidental edges, if an inner point, the number of adjacent edges, and the number of adjacent faces. ID `vnum` is also asigned to each vertex, for debugging identification.

```
struct tVertexStructure {
        double  v[2];         /* 2d coords of template vertex */
        double  v3d[3];       /* 3d coords of corresponding
                                    control point in space */
        int     vnum;         /* vertex ID */
        bool    mark;         /* T iff point processed. */
        int     degfree;      /* color index for drawing */
        bool    onboundary;
        bool    onborder;
        bool    isstar;
        bool    isinner;
        bool    isflip;
        tEdge   edges[2];     /* incident edges if inner */
        tEdge   staredges[6];/* star edges if corner */
        tVertex star[6];      /* star vertices if corner */
        tVertex starv;        /* corner vertex if star vertex */
        double  angle;
        int     adjenum;      /* number of adjacent edges */
        int     adjfnum;     /* number of adjacent faces */
        tVertex next, prev;
};
```

7.2.2 The edge list

The edge list stores information about the edges in much the same way as the vertex list does, with a doubly-linked list. Each edge is pointed to by its two end points, which are both pointers to vertex `structs`. Adjacent faces are computed and stored with pointers to face `structs` in the face list, which we will talk about later. Edges are stored only one time, and are single directional, which means, $\overrightarrow{ab}$ is considered the same as $\overrightarrow{ba}$, and an edge is consider to go from `endpts[0]` to `endpts[1]`. 0 is used to reference the left adjacent face and 1 for the right. The 0 left/1 right convention is used throughout the program. The edge list is reorganized so that the 0th adjacent face will always be on the left side of the edge. The edge points associated with that particular edge are also stored.

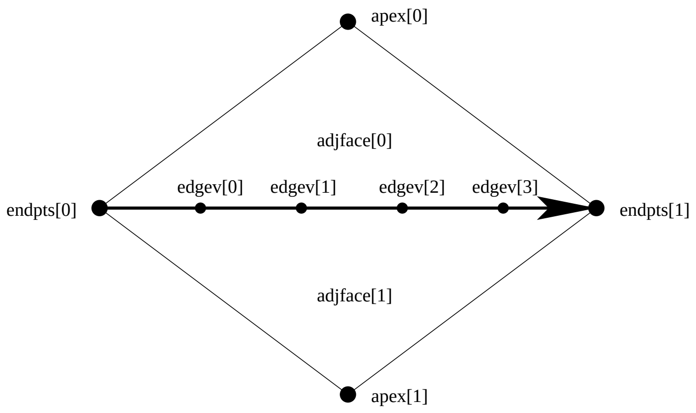


Figure 7.1: Structure of an edge

```
struct tEdgeStructure {
        tFace   adjface[2];     /* two adjacent faces*/
        tVertex endpts[2];      /* two end points */
        bool    onborder;
        bool    falseonb;       /* flag no longer used */
        double  length;         /* length of the edge. */
        tVertex apex[2];        /* the opposite vertices
                           of adjacent faces */
        tVertex edgev[DEG-1];   /* the edge points associated
           with this edge */
        int     id;             /* edge ID */
        bool    dir;
        int     adjfacenum;     /* number of adjacent faces */
        tEdge   next, prev;
};
```

7.2.3 The face list

The face list is also a doubly-linked list. As all faces are triangles, each face is described by three pointers to its three vertices, and three pointers to its three edges. The datastructure is so organized that all vertices are stored in counterclockwise order, and the 0th edge is the edge that goes from the 0th vertex to the 1st. The three angles of the face are also computed and stored. `angle[i]` is the angle on the ith vertex (Figure 7.2). Each face is again assigned an unique ID `id`. The inner points associated with that particular face are stored, so are all control points associated with the patch.

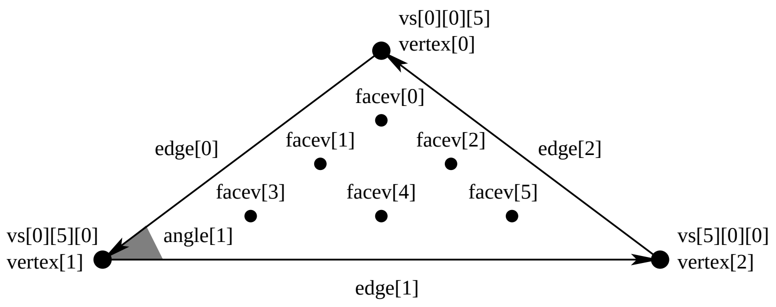


Figure 7.2: Structure of a Face.

```
struct tFaceStructure {
     tEdge   edge[3];                  /* adjacent edges */
     tVertex vertex[3];                /* adjacent vertices */
     tVertex facev[(DEG-2)*(DEG-1)/2]; /* inner points */
     tVertex vs[DEG][DEG][DEG];        /* all control
                                          points on the face */
     double  angle[3];                 /* angles formed by the
                          edges of the face */
     int     id;                       /* face ID */
     tFace   next, prev;
};
```

7.2.4 The control point list

The structure of this list is identical to that of a vertex list, using the same `struct`, except that this keeps track of all the control points the program generates based on the given degrees of the patches. Since the corner points are already stored in the vertex list, they are not stored in the control points list again. Therefore the control points list stores all edge points and inner points.

7.2.5 The control net tree

This is the data structure used to facilitate the recursive subdivision version of the de Casteljau algorithm for evaluation of a polynomial surface. It is basically a tree of doubly linked lists. Each linked list represents a control net. A typical `struct` in this tree is as follows:

```
struct tPointStructure{
  double coords[3];     /* 3D coords of the control point */
  double weight;        /* weight for rational */
  int id;
  tPoint net;           /* pointer to a linked list of
                           a sub control net */
  tPoint next, prev;

};
```

The subdivision method is the de Casteljau algorithm used to recursively subdive a triangular surface patch into subpatches, to obtain a triangulation of a surface with is a good approximation of the actual surface. Four calls to the standard de Casteljau are used on each patch at each level. It is basically a translation from the Mathematica implementation given by Gallier in [Gal00b]. For more information on the subdivision routines, please refer to Chapter 8 of [Gal00b].

7.3 Automatic prescription method

On large surfaces with many patches, it gets rather tedious to manually prescribe all control points required for proper propagation and subdivision. Given a template triangulation, it is often the case that there exists a three-dimensional mesh based on that triangulation. The mesh can come from scanned data or sampling, etc. In the case that no such mesh exists, it is also easy to simply "raise" the template triangulation by providing a z coordinate for each vertex in the template. Such a mesh then provides us with the necessary coordinates for all of the corner points. What remains is a way to set the rest of the control points via software.

We have developed an automatic method to prescribe control points for each patch based on the values (coordinates) of the three corner points. We employ the same scheme that assigns a first approximation of control points based on vertices of each triangular face on a triangulated mesh, see Subsection 6.2.4. We blend using the three given corner-point coordinates. This method generates surfaces that are fair approximations to the original mesh given. The prescribed non-corner control points are a weighted average of the corner points, and therefore the resulting surface after propagation is an approximation of the original mesh given. Any desired changes to the shape can then be carried out through local modifications and repropagation in the local vicinity.

7.4 Mouse control

To facilitate easy shape modification, we have also implemented mouse-controlled picking/dragging of a control point. Mouse picking is implemented via standard OpenGL built-in selection. Once the new location of the control point being modified is determined, necessary propagations will be carried out based on the changes, and the resulting surface will be displayed.

7.5 Linux platform

The entire implementation has been ported from SGI O2s running IRIX 6.5 to PCs running Redhat Linux. The SGI openGL library has been replaced with the open-source MESA GL library. The xforms library has support for the Linux platform. Since the original implementation was written in C, portability is of no issue there. No major change is made during porting, only those necessary to accommodate the minor differences between the operating systems and their respective implementation of the libraries.

7.6 Test results

The largest test case so far is a Delaunay triangulation with one hole and consists of 1000 triangles. All algorithmic computations have no visible delay. They are done in real time with no preprocessing. The bulk of the processing time is spent on subdivision (varies depending on depth) and final display of the surface (varies depending on hardware).

Chapter 8

Conclusions and Future Work

We have developed new incremental algorithms with local control for the design of triangular spline surfaces. Continuity of C^1 (possibly C^2) can be guaranteed for open surfaces, and our adaptation to G^1/AG^1 handles closed surfaces based on a triangulated polyhedron.

8.1 C^1 on open triangular mesh with parameterization

We have given a detailed summary on our C^1 method in Section 4.4, therefore we will only give a brief review here.

We have investigated the problem of polarizing triangular spline surfaces with built-in continuity constraints and proposed a new way of approaching the problem. Our algorithm's strength lies in its simplicity and local control ability. Possible directions for extending this method is C^2 continuity and interpolation. Preliminary work on both have been carried out.

8.2 AG^k on closed triangulated polyhedra

We have formally defined (triangulated) polyhedron-based spline surfaces. From there, we have also defined G^k continuity for spline surfaces on a polyhedron, which we call AG^k-continuous. Based on these definitions and our previous work, we have proposed a new way of fitting G^1 closed surfaces of arbitrary topology. It is an extension of our work on C^1 open surfaces over planar domain. A majority of the C^1 characteristics will carry over, for example, local control and ease of adaptation to interpolation.

8.2.1 Pseudo-AG^1 surfaces

In our attempt to adapt our C^1 algorithm to generate an AG^1 surface, we discovered that the constraints of AG^1 do not allow enough freedom to guarantee a local method. In favor of local control, we relax the continuity constraints further to pseudo-AG^1 surfaces. Pseudo-AG^1 surfaces are AG^1 surfaces that are not edge-consistent. That is, we have allowed the edges to differ from the vertex regions in terms of continuity conditions.

This way, we were able to achieve a local algorithm very much like our C^1 algorithm, even with the same propagation directions. The high local flexibility of our C^1 algorithm also carries over, as well as the fast and easy computations of control-point propagations.

Due to the freedom in the choice of the reparameterization maps both for the corner-point regions and across the edge diamonds, we have problems with undulations, which result in creases and folds in the final surface. Local and global fairing methods are still being researched.

8.2.2 Interpolation

Similar to what we have seen in the C^1 case, our method also leaves ample freedom for adaptation to interpolation. However, again, undulations caused by the possible "tilting" of affine frames are also valid worries here. In this case there might be more severe "tilting" as we start from non-planar domain, i.e., a polyhedron in space. It has become clear in the course of our research that a good fairing method is needed, and future efforts should be well spent there.

8.3 Fairing methods

The problem of unwanted "undulations" has appeared in both our C^1 and AG^1 work, with it being much more noticeable in the latter case. Faring methods, global, local or most likely both, needed to be introduced. However, it is not clearly which approach will be most effective, or if any would bring notable improvement in the final surface shape quality. Further research into areas such as surface optimization and discrete geometry might bring insights.

Bibliography

[Aki78] H. Akima. A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points. *ACM Transaction on Mathematical Software*, 4:148–159, 1978.

[AS60] Lars V. Ahlfors and Leo Sario. *Riemann Surfaces.* Princeton Math. Series, No. 2. Princeton University Press, 1960.

[BD88] A. A. Ball and Storry D.J.T. Conditions for tangent plane continuity over recursively generated B-spline surfaces. *ACM Transactions on Graphics*, 10(6):113–119, 1988.

[Ber90] Marcel Berger. *Géométrie 1.* Nathan, 1990. English edition: Geometry 1, Universitext, Springer Verlag.

[BLZ00] H. Biermann, A. Levin, and D. Zorin. Piecewise smooth subdivision surfaces with normal control. In *SIGGRAPH '00 Conference Proceedings, Annual Conference Series*, pages 113–119. ACM SIGGRAPH, 2000.

[Bre93] Glen E Bredon. *Topology and Geometry.* GTM No. 139. Springer Verlag, first edition, 1993.

[CDM91] A.S. Cavaretta, W. Dahmen, and C.A. Micchelli. Stationary subdivision. *Memoirs American Mathematical Society*, 93:1–186, 1991.

[CJ78] E. Catmull and Clark J. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[dC86] Paul de Faget de Casteljau. *Shape Mathematics and CAD.* Hermes, first edition, 1986.

[DeR85] T. DeRose. *Geometric Continuity: A Parametrization Independent Measure of Continuity for CAGD.* PhD thesis, University of California at Berkeley, Berkeley, CA, 1985. Dissertation.

[Doo78] D. Doo. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design*, pages 157–165, 1978.

[DRKT98] T. De Rose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *SIGGRAPH '98 Conference Proceedings, Annual Conference Series*, pages 85–94. ACM SIGGRAPH, 1998.

[DS78] D. Doo and M. Sabin. Analysis of the behavior of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10(6):356–360, 1978.

[DS90] W.H. Du and F. Schmitt. On the G^1 continuity of piecewise Bézier surfaces: A review with new results. *Computer-aided Design*, 22:556–573, 1990.

[Far82] G. Farin. A construction for the visual C^1 continuity of polynomial surface patches. *Computer Graphics and Image Processing*, 20:272–282, 1982.

[Far83] G. Farin. Smooth interpolation to scattered 3D data. In R.E. Barnhill and W. Boehm, editors, *Surfaces in CAGD*, pages 43–63. North-Holland, 1983.

[Far92] Gerald Farin. *Curves and Surfaces for CAGD.* Academic Press, third edition, 1992.

[Far95] Gerald Farin. *NURB Curves and Surfaces, from Projective Geometry to practical use.* AK Peters, first edition, 1995.

[FN83] R. Franke and G. Nielson. Surface approximation with imposed conditions. In R.E. Barnhill and W. Boehm, editors, *Surfaces in CAGD*, pages 135–146. North-Holland, 1983.

[FN91] R. Franke and G. Nielson. Scattered data interpolation and applications: A tutorial and survey. In H. Hagen and D. Roller, editors, *Geometric Modeling*, pages 131–160. Springer Verlag, 1991.

[Fra82] R. Franke. Scattered data interpolation: Tests of some methods. *Mathematics of Computation*, 38:181–200, 1982.

[Ful95] William Fulton. *Algebraic Topology, A first course.* GTM No. 153. Springer Verlag, first edition, 1995.

[Gal97] J. Gallier. Notes on surface topology: The classification theorem for compact surfaces and a detour on fractals. Course notes, 1997.

[Gal00a] Jean H. Gallier. *Curves and Surfaces in Geometric Modeling.* Morgan Kaufman, first edition, 2000.

[Gal00b] Jean H. Gallier. *Geometric Methods and Applications.* Springer, first edition, 2000.

[GCR77] C.M. Gold, J.D. Charters, and J. Ramsden. Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain. *Computer Graphics*, 11:170–175, 1977.

[GGH02] X. Gu, S. Gortler, and H. Hoppe. Geometry images. In *SIGGRAPH '02 Conference Proceedings, Annual Conference Series.* ACM SIGGRAPH, 2002. To appear in SIGGRAPH '02.

[GH95] C. Grimm and J. Hughes. Modeling surfaces of arbitrary topology using manifolds. In *SIGGRAPH '95 Conference Proceedings, Annual Conference Series*, pages 359–368. ACM SIGGRAPH, 1995.

[GR74a] W.J. Gordon and R.F. Riesenfeld. B-splines curves and surfaces. In R.E. Barnhill and R.F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 95–126. Academic Press, New York, 1974.

[GR74b] W.J. Gordon and R.F Riesenfeld. Bernstein-Bézier methods for the computer aided design of free form curves and surfaces. *ACM*, 21:293–310, 1974.

[GS93] Günther Greiner and Hans-Peter Seidel. Modeling with triangular *b*-splines. In *2nd ACM Symposium on Solid Modeling*, pages 211–220, 1993.

[Hag86] H. Hagen. Geometric surface patches without twist constraints. *Computer Aided Geometric Design*, 3:179–184, 1986.

[HBT00] S. Hahmann, G.P. Bonneau, and R. Taleb. Localizing the 4-split method for G^1 free-form surface fitting. Technical report, University of Grenoble, 2000. To appear in Computing, 2001.

[HBT01] S. Hahmann, G.P. Bonneau, and R. Taleb. Modélisation de surfaces de topologie arbitraire. In *SIAM Conference on Geometric Design and Computing*, page 37, 2001. Abstract only.

[HDD$^+$94] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Piecewise smooth surface reconstruction. In *SIGGRAPH '94 Conference Proceedings, Annual Conference Series*, pages 295–302. ACM SIGGRAPH, 1994.

[Her85] G. Herron. Smooth closed surfaces with discrete triangular interpolants. *Computer Aided Geometric Design*, 2:297–306, 1985.

[Her87] G. Herron. Techniques for visual continuity. In G. Farin, editor, *Geometric Modeling*, pages 163–174. SIAM, 1987.

[HH89] Hagen H. and Pottmann H. Curvature continuous triangular interpolants. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 373–384. Academic Press, 1989.

[HL93] J. Hoschek and D. Lasser. *Computer Aided Geometric Design*. AK Peters, first edition, 1993.

[Jen87] T. Jensen. Assembling triangular and rectangular patches and multivariate splines. In G. Farin, editor, *Geometric Modeling: Algorithm and New Trends*, pages 203–222. SIAM, 1987.

[Kob01] L. Kobbelt. $\sqrt{3}$-subdivision. In *SIGGRAPH '01 Conference Proceedings, Annual Conference Series*, pages 103–112. ACM SIGGRAPH, 2001.

[Law77] C.L. Lawson. Software for C^1 surface interpolation. In J.R. Rice, editor, *Software III*, pages 159–192. Academic Press, 1977.

[LMH00] A. Lee, H Moreton, and H Hoppe. Displaced subdivision surfaces. In *SIGGRAPH '00 Conference Proceedings, Annual Conference Series*, pages 85–94. ACM SIGGRAPH, 2000.

[Loo87] Charles Loop. *Smooth Subdivision Surfaces based on triangles.* PhD thesis, University of Utah, Department of Mathematics, Salt Lake City, Utah, 1987. Masters's thesis.

[Loo94] Charles Loop. A G^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11:303–330, 1994.

[Mas87] William S. Massey. *Algebraic Topology: An Introduction.* GTM No. 56. Springer Verlag, second edition, 1987.

[Mas91] William S. Massey. *A Basic Course in Algebraic Topology.* GTM No. 127. Springer Verlag, first edition, 1991.

[MLM$^+$92] S. Mann, C. Loop, Lounsberry M., D. Meyers, Painter J., DeRose T., and K. Sloan. A survey of parametric fitted data: Fitting using triangular interpolants. In H. Hagen, editor, *Curve and Surface Design*, pages 145–172. SIAM, 1992.

[MS92] D. Micchelli and H.-P. Seidel. Blossoming begets B-spline bases built better by B-patches. *Mathematics of Computation*, 59:97–115, 1992.

[Pet91] J. Peters. Smooth interpolation of a mesh of curves. *Constructive Approximation*, 7:221–246, 1991.

[Pet95] J. Peters. Biquartic C^1-surface splines over irregular meshes. *Computer Aided Geometric Design*, 27:895–903, 1995.

[Pet02] J. Peters. Geometric continuity. In *Handbook of Computer Aided Geometric Design*. Elsevier, 2002.

[Pip87] B. Piper. Visually smooth interpolation with triangular Bezier patches. In G. Farin, editor, *Geometric Modeling: Algorithm and New Trends*, pages 221–234. SIAM, 1987.

[PS95] P Pfeifle and H.-P. Seidel. Fitting triangular B-splines to functinal scattered data. In *Graphics Interface '95*, pages 80–88, 1995.

[Ram87] L. Ramshaw. Blossoming: A connect-the-dots approach to splines. Technical report, Digital SRC, Palo Alto, CA 94301, 1987. Report No. 19.

[Ram88] L. Ramshaw. Bézier and B-splines as multiaffine maps. In RA Earnshaw, editor, *Theorectical Foundations of Computer Graphics and CAGD*, pages 757–776. Springer, 1988.

[Ram89] L. Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 13:38–46, 1989.

[Ris92] J.-J. Risler. *Mathematical Methods for CAD.* Masson, first edition, 1992.

[Sab83] M. Sabin. Non-rectangular surface patches suitable for inclusion in a b-spline surface. In *Proceedings of Eurographics '83*, pages 57–69, 1983.

[Sar00] R. F. Sarraga. A variational method to model G^1 surfaces over triangular meshes of arbitrary topology in R^3. *ACM Transactions on Graphics*, 19:279–301, 2000.

[Sat99] Hajime Sato. *Algebraic Topology: An Intuitive Approach.* American Mathematical Society, first edition, 1999.

[Sei88] H.-P. Seidel. Knot insertion from a blossoming point of view. *Computer Aided Geometric Design*, 5:81–86, 1988.

[Sei89] H.-P. Seidel. A new multiaffine approach to B-splines. *Computer Aided Geometric Design*, 6:23–32, 1989.

[Sei93] H.-P. Seidel. Polar forms for geometrically continuous spline curves of arbitrary degree. *ACM Transactions on Graphics*, 12(1):1–34, 1993.

[She68] D. Shepard. A two-dimentional interpolation function for irregularly spaced data. In *Proc. 23rd Nat. Conf. ACM*, pages 517–523, 1968.

[SS88] L. Shirman and C. Sequin. Local surface interpolation with Bezier patches. *Computer Aided Geometric Design*, 4:279–296, 1988.

[Vel92] R. C. Veltkamp. Survey of continuities of curves and surfaces. *Computer Graphics Forum*, 11(2):93–112, 1992.

[Wey46] Hermann Weyl. *The Classical Groups. Their Invariants and Representations.* Princeton Mathematical Series, No. 1. Princeton University Press, second edition, 1946.

[Xu99] D. Xu. Classification theorem for compact surfaces. Technical report, University of Pennsylvania, 1999. MS-CIS-02-14.

[Zor96] D. Zorin. *C^k Continuity of Subdivision Surfaces.* PhD thesis, California Institution of Technology, 1996. Ph.D. thesis.