2007-04-17

T-Spline Simplification

David L. Cardon
Brigham Young University - Provo

T-SPLINE SIMPLIFICATION

by

David L. Cardon

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

David L. Cardon

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Thomas W. Sederberg, Chair
Date	Bryan S. Morse
Date	Scott N. Woodfield

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of David L. Cardon in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Thomas W. Sederberg Chair, Graduate Committee
Accepted for the Department	Parris K. Egbert Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

T-SPLINE SIMPLIFICATION

David L. Cardon

Department of Computer Science

Master of Science

This work focuses on generating approximations of complex T-spline surfaces with similar but less complex T-splines. Two approaches to simplifying T-splines are proposed: a bottom-up approach that iteratively refines an over-simple T-spline to approximate a complex one, and a top-down approach that evaluates existing control points for removal in producing an approximations. This thesis develops and compares the two simplification methods, determining the simplification tasks to which each is best suited. In addition, this thesis documents supporting developments made to T-spline research as simplification was developed.

ACKNOWLEDGMENTS

Roughly half of the material in this thesis is drawn from the publication, "T-Spline Local Refinement and Simplification" [1], in which I was closely involved during the beginning of my program at BYU. I am extremely grateful for all of the theoretical discussions with Dr. Sederberg, which allowed us to develop the code and the theory for that paper concurrently. All of the coauthors also contributed significantly to the production of such a quality publication—many thanks to Tom Finnigan and Nick North for their inestimable help with figures and last minute coding, and to Jianmin Zheng and Tom Lyche for their invaluable knowledge of the literature and related methods.

I would also like to thank Dr. Sederberg for his patience with my progress—it has been a long road with more than a few diversions. Fortunately, I think they have provided useful lessons and insights, which have rounded out my understanding of the subject. In the end, I can say with certainty that I gained an education out of all of this.

In addition to those directly connected to my research, I would like to acknowledge the useful contributions of other students in the lab—particularly, David Cline, Josh Jenny and Doug Kennard come to mind. Their willingness to listen to my complex descriptions (of whatever algorithm I happened to be working on) may not have helped them, but it certainly helped me keep moving ahead.

Finally, I would like to thank my parents for never making me to go to school and for valuing knowledge themselves—their examples taught me more than coercion ever could.

Contents

List of Figures

Chapter 1

Introduction

Taking the design of a geometric object from conception to instantiation is a lengthy process for artisan and craftsman alike. The quality of the design and the speed with which it is produced is in part a function of the tools available to the designers. The usefulness of a design tool depends on the expertise required to wield it and on the tool's effectiveness in affecting the shape of the object being designed. By providing more useful design tools and a more forgiving design environment, computers have become essential assets for designers.

The field of *Computer-Aided Geometric Design (CAGD)* encompasses all problems in which a computer algorithm may be used to compute or store the geometry of an object [2]. The primary goal of CAGD is to assist the designer in creating a geometric description of a target object's shape. CAGD systems are the tools of the modern designer and artist. For example, modern automobiles are now designed, formed and evaluated (via simulation) entirely using a computer. In the past, precise paper diagrams and costly physical prototypes were required in order to produce the same quality of design [3].

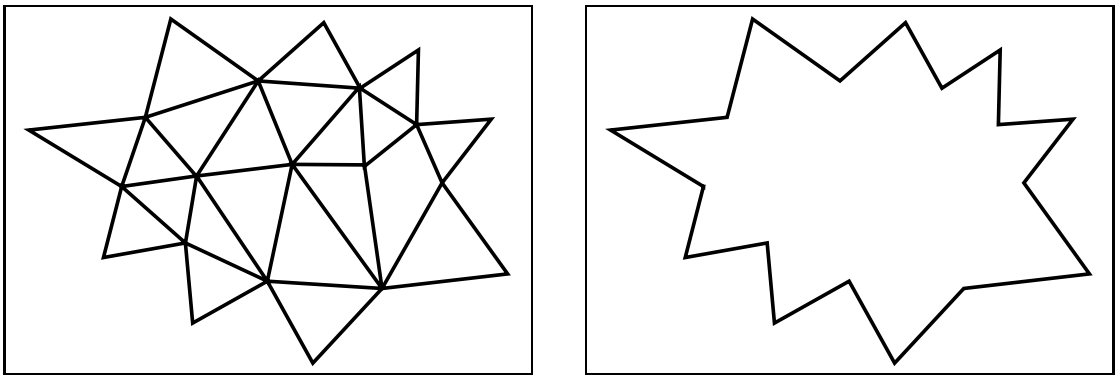
The geometric description of an object in a computer is typically constructed using simple, atomic shapes called *primitives*. Computers can describe objects in two and three spatial dimensions, and primitives differ depending on the dimension. *Lines* and *curves* are the primitive shapes from two-dimensional geometric design. *Planes* and *surfaces* are the equivalent primitives in three dimensions.

In order to build objects from graphics primitives in a CAGD system, an artist manipulates the primitives to form them into a desired shape. The process of forming primitives into a desired object is called *modeling* and the geometric object that is formed is called a *model*. There are many different kinds of models and modeling systems and for this thesis we focus on free-form curve and surface models, whose primitives have adjustable components called *control points*. In these systems, the control point is the most basic element with which the designer interacts while modeling—by altering the geometric position of a control point within the system, the designer alters a portion of the corresponding primitive's shape.

For curve primitives the curve control points form what is called a *control polygon*. For surface primitives the control points form a *control grid* or *control mesh*, which consists of sets of control points connected to one another via a set of edges. To add more detail to a primitive, an artist performs an operation that produces additional control points for adjusting the model. As more control points are added to a model, more detail is permitted, but the increased number of control points can make the model more complicated.

Simplification is the process of modifying an existing geometric description of an object to an equivalent or approximately equivalent form that is less complex—i.e., that has fewer control points. Since the main focus of this thesis is the simplification of geometric surfaces, the task of surface simplification is to reduce and modify the sets of control points used to define a geometric surface. In the approximation case, this modification should attempt to represent the original shape as best as possible within the desired approximation.

Sometimes complexity is necessary in order to describe a desired shape. We can think of each control point of an object as carrying some geometrical information about a portion of the object's description. From this perspective some control points may carry redundant information. Useful simplification of the object will eliminate



(a) Complex planar polygon mesh.

(b) Simplified planar polygon mesh. The inner control points contain no useful geometric information and thus may be removed losslessly.

Figure 1.1: A comparison of two planar polygon meshes. All polygons in the depicted meshes are co-planar.

redundant control points, reducing the object's control points to the necessary set. For example, we might describe a single planar polygon with a set of several coplanar triangles as in Figure 1.1(a). However, a simpler description of this same shape would be a single 18-sided planar polygon as in Figure 1.1(b). In Figure 1.1(a) the additional control points of the shape's component triangles carry no information needed to describe the shape. The identification of unnecessary control points for simplification depends entirely on the properties of the surface used to create the object—if Figure 1.1(a)'s triangles were not all coplanar, the described simplification may not be possible.

One of the most common types of free-form surfaces in three-dimensional CAGD are founded on the definition of a two-dimensional curve called a NURBS (Non-Uniform Rational B-Spline) curve. The NURBS curve is a *parametric curve*, meaning that the points on the curve are swept out as the curve's parameter[1] changes. Figure 1.2 illustrates a NURBS curve with its control points and a few parameter val-

[1]Mathematically, the parameters of a curve or surface are the independent variables in the expression for the curve/surface

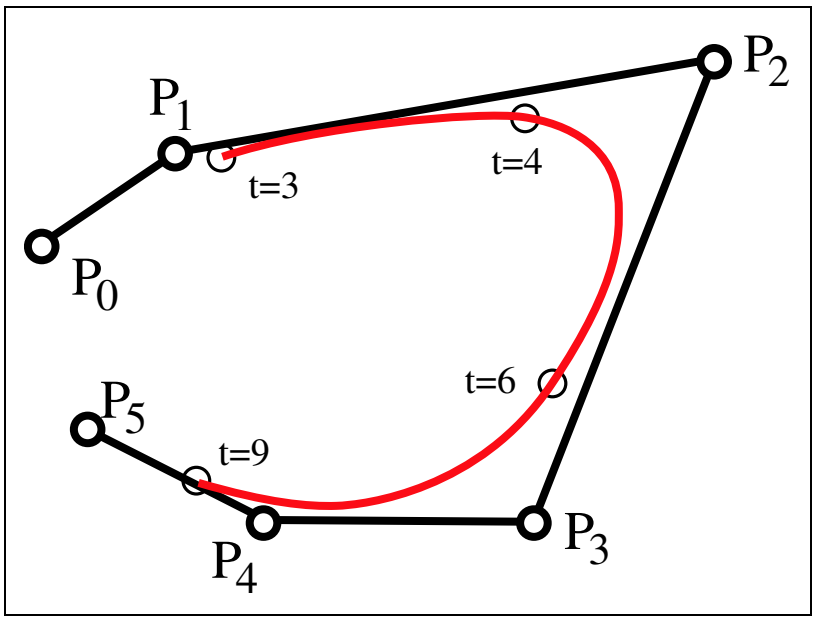
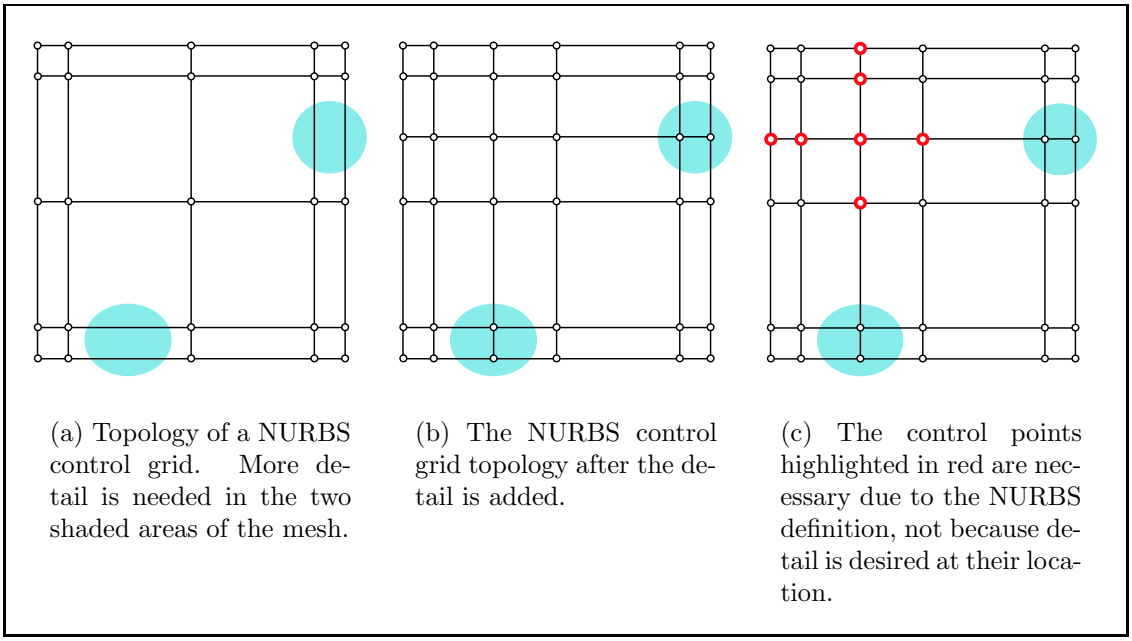


Figure 1.2: A NURBS curve with a few labeled parameter values. The curve is a function of the control points and the t parameter value.

ues along the curve. The NURBS curve is formed by interpolating the positions of the curve's control points as the parameter changes. One can visualize how this works using the concept of center-of-mass from physics. Imagine that each control point has a mass that changes smoothly over time. If we plot the center of the masses as time passes, the time-varying centers will sweep out a smooth curve in space.

The main difference between various curve definitions lies in the definition of the mass functions (called *blending* or *basis functions*) associated with the curve's control points. The main reason that NURBS curves and surfaces are particularly useful to designers lies in the mathematical properties of their blending functions, which include local control, the variation diminishing property[2] and constraints on the continuity of the curve. Due to their usefulness in design, NURBS curves and their blending functions have been studied at length by the research community [4][5][6][7].

[2]The *variation diminishing* property states that no straight line intersects the curve more times than it does it's control polygon.



(a) Topology of a NURBS control grid. More detail is needed in the two shaded areas of the mesh.

(b) The NURBS control grid topology after the detail is added.

(c) The control points highlighted in red are necessary due to the NURBS definition, not because detail is desired at their location.

Figure 1.3: The superfluous control points that occur in NURBS surfaces. The highlighted control points in (c) are not desired by the artist, but must exist in the mesh due to the NURBS definition.

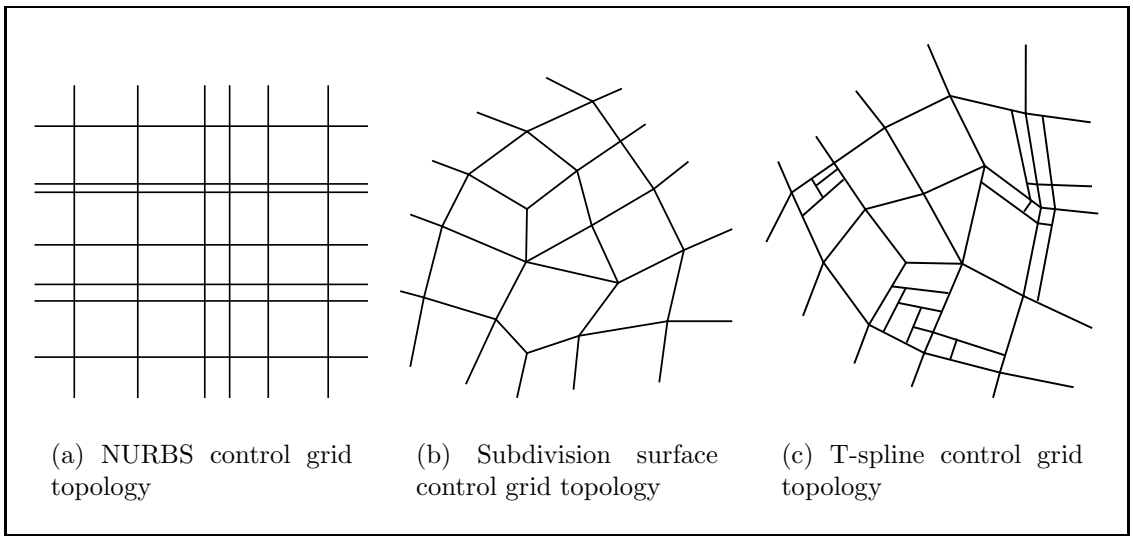
The extension of NURBS curves in two dimensions to surfaces in three dimensions may be trivially achieved via a tensor product [8]. The result of the tensor product produces control points arranged in an array that defines the NURBS surface in terms of two parameters instead of the single one used in the curve case. While this derivation is straightforward mathematically, it has two distinct disadvantages:

1. The rigid array topology of the NURBS control grid. While simple to describe mathematically, the array topology can lead to a large number of unnecessary control points. For example, Figure 1.3(a) illustrates the topology of a NURBS surface, into which a designer wishes to add more detail. In order to do so, new control points are added (Figure 1.3(b)), which results in several unnecessary control points (Figure 1.3(c)). These control points are only present in the control grid due to the topological constraints of the NURBS surface.

2. The strict parameterization of the surface does not permit a lot of common shapes. NURBS surfaces may only be homeomorphic to a plane, to an open cylinder or to a torus. Many objects in the real world do not fit into these criteria. For example, a NURBS surface representation of a sphere is not actually a closed surface and is usually a cylinder with collapsed ends.

In an attempt to overcome the latter limitation of NURBS surfaces, researchers derived from it another surface called a *subdivision surface* to allow for more topological flexibility [9][10][11][12]. With subdivision surfaces, control meshes may include faces with an arbitrary number of sides (NURBS meshes are limited to four-sided faces) and vertices of arbitrary valence. While an improvement over NURBS surfaces, subdivision surfaces have limitations of their own. First, many of these surfaces are defined only in terms of their subdivision rules, which must be applied to the entire mesh, making them slow to evaluate. Secondly, subdivision surfaces are difficult to modify at different levels of detail. In order to modify in detail a small portion of the mesh, a global refinement must be performed on the entire mesh, making coarse control difficult where it is still desired. While hierarchical methods have been developed to overcome this problem [13], the hierarchy makes user interfaces clunky and they are therefore seldom used.

Recently, a new surface definition called a T-spline was introduced [14][15]. The T-spline surface is founded on the same blending functions as a NURBS surface, but the surface definition overcomes the topological limitations of the NURBS control grid. In a T-spline control grid, control points are not required to be arranged in an array of control points, reducing dramatically the complexity required to represent a smooth surface. Figure 1.4 illustrates the topological difference between T-spline, NURBS and subdivision surface control grids. Founded on the NURBS surface, T-spline surfaces are backward compatible with both NURBS and popular subdivision surfaces.



(a) NURBS control grid topology

(b) Subdivision surface control grid topology

(c) T-spline control grid topology

Figure 1.4: A comparison of different surface type topologies. Subdivision control grids generalize NURBS to arbitrary topology. T-splines allow T-junctions, which further generalize the topology permitted.

The new flexibility of T-splines over NURBS presents new opportunities for research in simplification. Previous work for simplifying NURBS models [16] was limited by the topological constraints of the NURBS surface—an entire row of control points has to be removed to simplify a NURBS model. With T-splines the removal of only a partial row is permitted in simplification. In addition, while some work has been done on simplifying subdivision surfaces [17], T-splines allow for greater freedom in the arrangement of control points and in the parameterization of the surface. Finally, T-spline surfaces permit formerly infeasible surface operations such as merging [18] and intersection of separate surfaces. The results of these new operations can also benefit from automated T-spline simplification methods.

1.1 Overview

Motivated by the new possibilities available in T-splines, this work develops specifically two methods for simplifying T-spline surfaces. Since conversion of existing

NURBS and subdivision surface models to T-spline models is lossless, these methods may be applied to a large quantity of existing models.

This document proceeds as follows: Chapter 2 describes the T-spline surface and fundamental operations that are the foundation for the algorithms used in T-spline and related simplification. Chapter 3 presents previous research related to the simplification of free-form surface models and identifies the simplification problem addressed in this thesis. Chapters 4 and 5 comprise the two developed approaches to T-spline simplification—iterative refinement and iterative simplification, respectively. Chapter 6 presents results and compares the two approaches to each other and to T-spline derivatives of spline decomposition. Chapter 7 briefly summarizes and concludes the thesis.

Chapter 2

Foundations

This chapter presents the salient features of T-splines and related surfaces, including the two fundamental atomic operations of T-spline simplification: blending function refinement and least-squares fitting.

2.1 T-Splines

The T-spline surface was developed and introduced to the community at large in [14]. This chapter focuses specifically on the aspects of the T-spline definition that apply to the simplification problem, beginning with a brief review of NURBS curves and surfaces (on which T-splines are founded). We limit our discussion to T-spline and NURBS surfaces of degree three although the principles extend to any degree T-spline. [19] explains how to construct T-splines of any degree.

2.1.1 NURBS Curves

Figure 2.1 shows a cubic NURBS curve. The $\mathbf{P_i}$ are geometric positions (in $\mathbb{R}^2$ or $\mathbb{R}^3$) and are called control points. The line segments connecting the control points are called the curve's control polygon. Each edge of the control polygon (except for the first and last) corresponds to a Bézier curve. The shape of the curve can be changed by either moving the control points or by altering a scalar value assigned to each $\mathbf{P_i}$ called a weight, w_i.

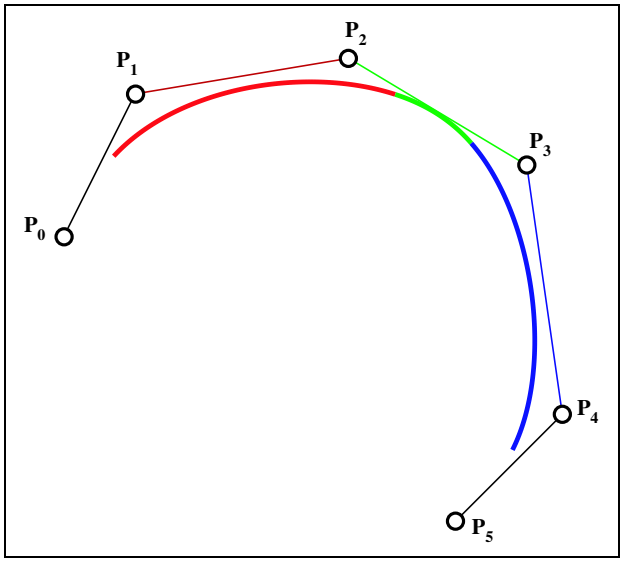


Figure 2.1: A cubic NURBS curve shown with its control points and control polygon.

A cubic NURBS curve can be decomposed into multiple connected cubic Bézier curves, each of which is C^2 continuous with its neighbors. The parameter values at which two adjacent Bézier curves meet are called knot values or, simply, knots. Thus, a NURBS curve consists of a set of control points and a sequence of knot values called the curve's *knot vector*.

The equation for a degree three NURBS curve is given by

$$\mathbf{P}(t) = \frac{\sum_{i=0}^{n} \mathbf{P_i} w_i B_i^3(t)}{\sum_{i=0}^{n} w_i B_i^3(t)}, \tag{2.1}$$

where the $B_i^3(t)$ are called B-spline basis functions. Each B-spline basis function is a piecewise polynomial comprised of four C^2 segments as shown in Figure 2.2. $B_i^3(t)$ can be defined by the recurrence relation:

$$B_i^0(t) = \begin{cases} 1 & \text{if } t_i \le t \le t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

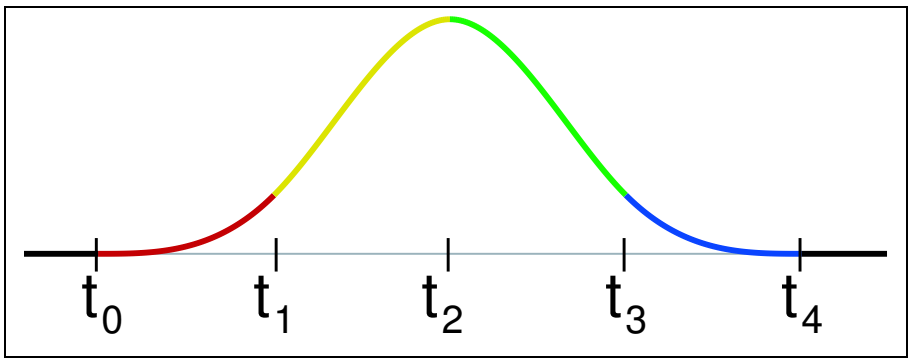


Figure 2.2: A B-spline basis function shown as four piecewise polynomial segments and labeled with its knot values. The function value outside of the colored polynomial segments is zero.

$$B_i^n(t) \quad = \quad \omega_i^n(t) B_i^{n-1}(t) + \left(1 - \omega_{i+1}^n(t)\right) B_{i+1}^{n-1}(t)$$

where

$$\omega_i^n(t) \quad = \quad \begin{cases} \frac{t-t_i}{t_{i+n-1}-t_i} & \text{if } t_i \neq t_{i+n-1} \\ 0 & \text{otherwise} \end{cases} .$$

As shown in Figure 2.2, $B_i^3(t)$ is non-zero over a span of five knot values. Because of this, NURBS curves are said to exhibit *local control*; i.e., any one control point in a cubic NURBS may influence at most four Bézier curves. Referring to Figure 2.3, the blending function of the control point labeled t_3 is non-zero over the five knots centered about that point: t_1, t_2, t_3, t_4 and t_5.

While knot values are traditionally used to define B-spline basis functions, an alternative notation called *knot intervals* was introduced in [12]. A knot interval is the difference between two adjacent knot values and is the parameter length of the Bézier curve that lies between the knots. Figure 2.3 illustrates a NURBS curve with both knot values and knot intervals labeled. Take note that degree three NURBS curves have an extra knot on each end of the curve that is not associated with a

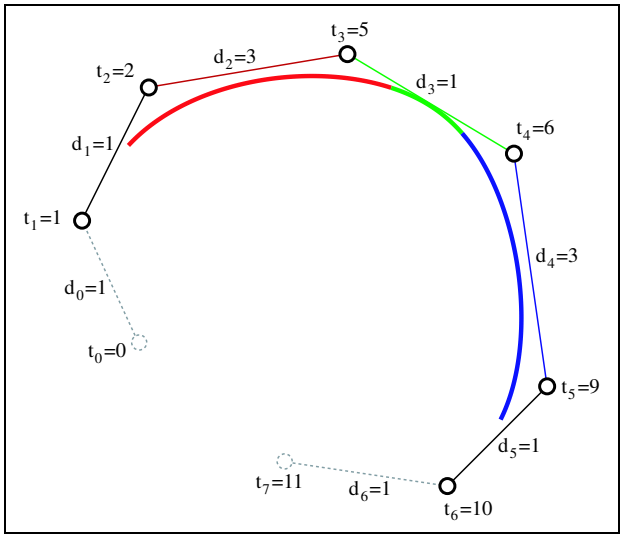


Figure 2.3: A NURBS curve labeled with knot values and knot intervals. The knot vector for the curve is $(t_0, \ldots, t_7)$, which is expressed as $[d_0, \ldots, d_6]$ in knot interval notation.

control point. In interval notation these knots are represented with an extra interval off the end-points of the curve.

From Figure 2.3 it is clear that knot values may be computed from the knot intervals—assign a value to one of the knot boundaries of an interval and then add or subtract intervals accordingly to attain the remaining knot values. Since the appearance of a NURBS curve is invariant under linear transformation of the knot vector, the knot value selected as the initial reference may be entirely arbitrary.

Although knot values are sufficient for use with NURBS curves and surfaces, knot interval notation is necessary for T-splines, which have fewer restrictions on topology. In addition, knot interval notation is often more intuitive than knot values, since the intervals associate parameter distances directly with the edges of the control polygon. For these reasons, we will use knot interval notation from this point on in the thesis.

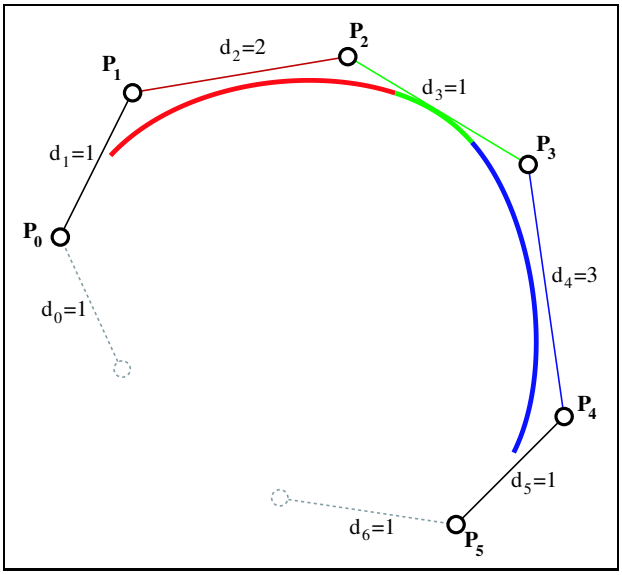


Figure 2.4: A NURBS curve before the knot insertion process with knot intervals labeled d_i.

Knot Insertion

We now present an algorithm for knot insertion[1] using the knot interval notation. Since we are dealing with knot intervals rather than knot values, this procedure might more properly be called interval splitting. We illustrate the procedure using the curve in Figure 2.4, for which we will split interval $d_2 = 3$ into intervals 2 and 1. First, annotate each edge of a degree three NURBS curve with its own interval and that of its neighbors as in Figure 2.5. Note that the intervals marked on the edges are drawn in proportion to their interval values. Notice also that intervals associated with a single edge appear in three of the marked up edges (the associated edge itself and its two neighbors).

Second, select a parameter distance down one of the control polygon's edges, at which insertion is desired (the insertion edge) as in Figure 2.6. Then, identify that parameter position on the three annotated edges that include the insertion parameter's associated interval (see Figure 2.7). Finally, compute the positions of the control points at the identified parameter positions—the two control points incident

[1]For more background information on knot insertion, see Section 3.4.

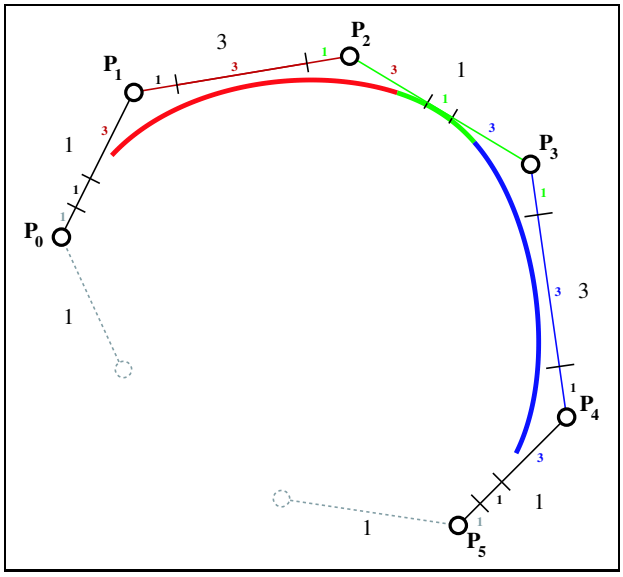


Figure 2.5: A NURBS curve with edges annotated in preparation for insertion. Each of the edges of the control polygon is annotated according to nearby knot intervals. The annotation knot intervals are color-coded to match their interval's associated edge.

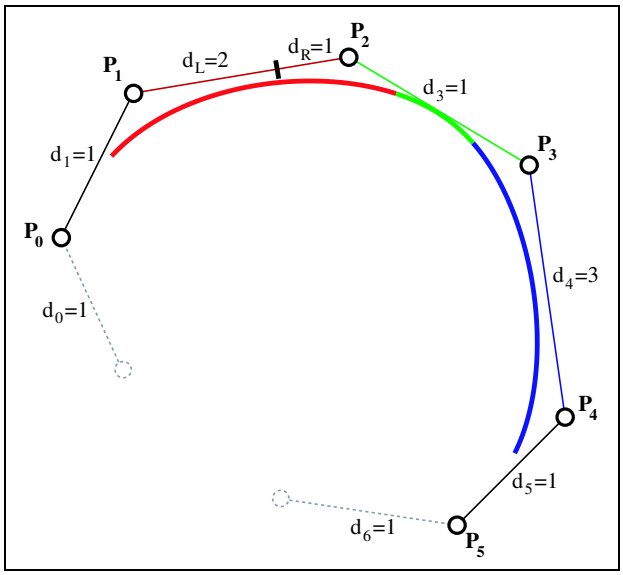


Figure 2.6: Selecting an insertion parameter position on a NURBS curve. The selected position is two-thirds the distance down the red edge with interval d_2, dividing it into d_L and d_R.

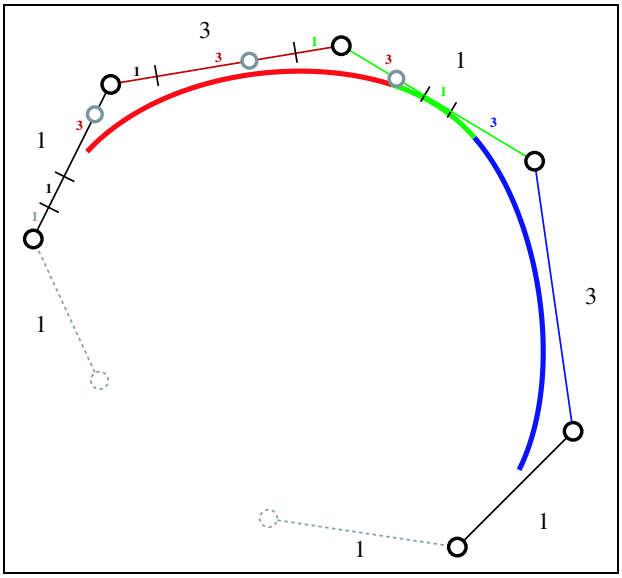


Figure 2.7: Marking the insertion parameter on the annotated edges. The insertion parameter is now marked on each of the three annotated edges, the three markers indicate the locations for the new NURBS control points after insertion.

to the insertion edge are moved to the positions on their respective neighbor edges and the new control point is placed at the position identified on the insertion edge, as illustrated in Figure 2.8.

This geometric description provides a graphical medium for deriving an analytical solution to knot insertion. Directly from Figure 2.7, we can infer the linear combinations of the original control points that compose the positions of the new ones, using the labels of Figure 2.6:

$$\begin{aligned} \mathbf{P'_1} &= \tfrac{d_R}{d_0+d_1+d_2}\mathbf{P_0} + \tfrac{d_0+d_1+d_L}{d_0+d_1+d_2}\mathbf{P_1} \\ \mathbf{P'_k} &= \tfrac{d_R+d_3}{d_1+d_2+d_3}\mathbf{P_1} + \tfrac{d_1+d_L}{d_1+d_2+d_3}\mathbf{P_2} \\ \mathbf{P'_2} &= \tfrac{d_R+d_3+d_4}{d_2+d_3+d_4}\mathbf{P_2} + \tfrac{d_L}{d_2+d_3+d_4}\mathbf{P_3}. \end{aligned} \tag{2.2}$$

2.1.2 NURBS Surfaces

A NURBS surface is the bivariate extension of the NURBS curve and is expressed as

$$\mathbf{P}(s,t) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\mathbf{P_{ij}}w_{ij}B^3_{s,i}(s)B^3_{t,j}(t)}{\sum_{i=0}^{n}\sum_{j=0}^{m}w_{ij}B^3_{s,i}(s)B^3_{t,j}(t)} \tag{2.3}$$

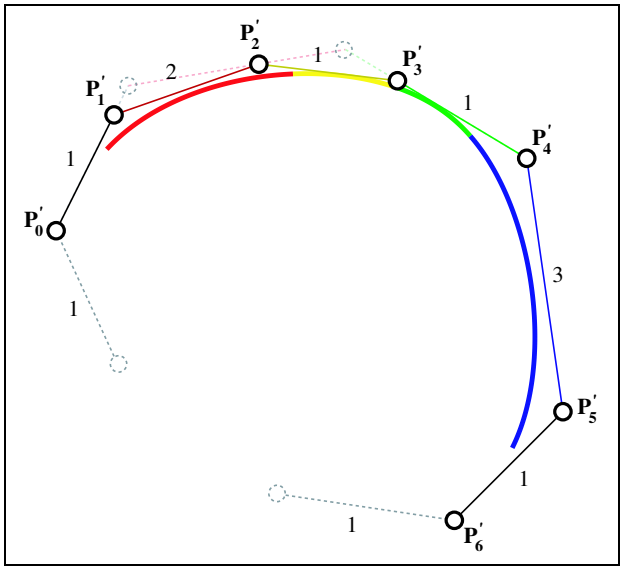


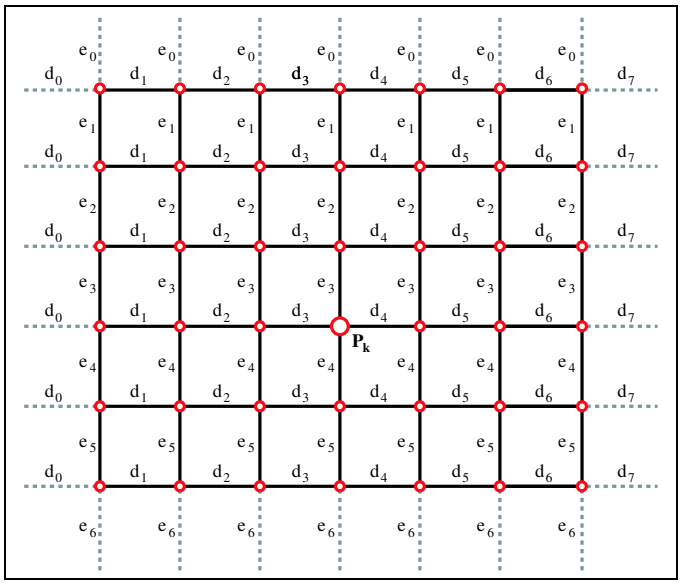
Figure 2.8: The final NURBS curve after the insertion at the selected parameter position.

where $B^3_{s,i}(s)$ and $B^3_{t,j}(t)$ are B-spline basis functions associated with the s and t parameters (and therefore their corresponding knot intervals), respectively. This NURBS surface has $(n+1)\times(m+1)$ control points, which are arranged in a rectangular array. Figure 2.9(a) illustrates the control grid for a NURBS surface with edges labeled by knot intervals. As seen in this figure, knot intervals for a NURBS surface must be repeated for each row and column of edges.

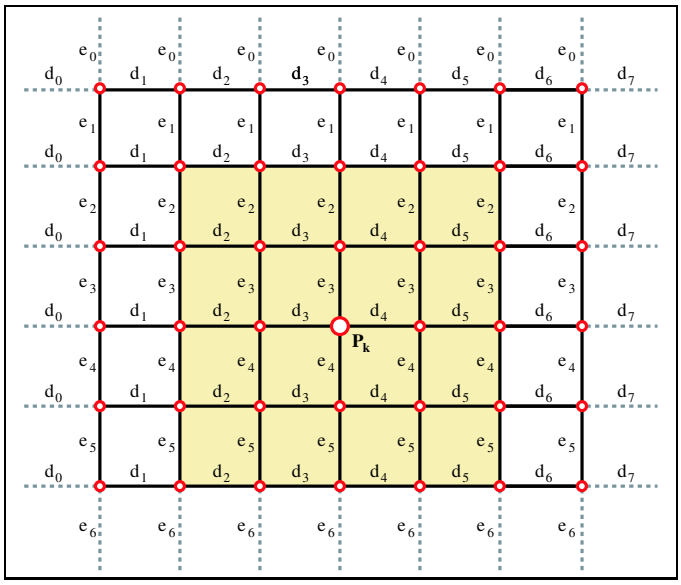
Just as with NURBS curves, NURBS surface control points feature local control. For example, in Figure 2.9(a) the control point labeled $\mathbf{P_k}$ influences the surface shape for only a finite region of the grid as highlighted in Figure 2.9(b). The blending functions for $\mathbf{P_k}$, $B^3_{s,k}$ and $B^3_{t,k}$, depend only on the knot intervals surrounding $\mathbf{P_k}$: $B^3_{s,k}$ depends on intervals (e_2, e_3, e_4, e_5) and $B^3_{t,k}$ depends on intervals (d_1, d_2, d_3, d_4). Note the correspondence between these intervals and the region that $\mathbf{P_k}$ influences.

2.1.3 T-Spline Definition

Like a NURBS surface, a *T-spline* is a tensor product B-spline surface, meaning that control points of a T-spline surface contribute to the shape of the surface according



(a) A degree three NURBS surface control grid. The knot intervals for the s parameter are labeled with $\{d_0, \ldots, d_7\}$ and those for the t parameter are labeled with $\{e_0, \ldots, e_6\}$.



(b) The parameter region of the NURBS surface that $\mathbf{P_k}$ influences.

Figure 2.9: Topology and Local Control of NURBS Surfaces.

to a tensor product B-spline blending function[2]. While T-splines are based on the same blending functions as a NURBS surface, a T-spline equation is slightly different from that of a NURBS surface:

$$\mathbf{P}(s,t) = \frac{\sum_{i=0}^{n} \mathbf{P_i} w_i B_i(s,t)}{\sum_{i=0}^{n} w_i B_i(s,t)} \tag{2.4}$$

where $B_i(s,t) = B_{s,i}(s) \cdot B_{t,i}(t)$.

The univariate portions[3] (i.e., $B_{s,i}(s)$ and $B_{t,i}(t)$) of the T-spline blending functions can be unique to each control point, whereas in a NURBS surface $B_{s,i}(s)$ is the same for all control points in a given column, and $B_{t,i}(t)$ is the same for all control points in a given row. In a T-spline, the T-grid conveys the knot intervals for each control point's blending functions, as follows.

A T-grid is defined by a set of edges $E = \{e_i\}$ and control points $P = \{\mathbf{P_i}\}$. Each edge connects two control points and has associated with it a scalar knot interval value, d_i. By connecting control points together, sets of edges form a grid with four-sided faces in the topology[4]. A single *side* of a face may contain multiple edges, connected via T-junctions on that face. For example, in Figure 2.10 face F_1 contains three edges on its left side. The *knot interval length* of a side is the sum of the edge knot intervals along that face. So, the knot interval length of the left side of F_1 in Figure 2.10 is $d_3 + d_4 + d_5$. The T-spline definition states that the knot interval lengths of opposite sides of a face must be equal. For Figure 2.10 this implies that $d_0 + d_1 = d_3 + d_4 + d_5$.

With edges and knot intervals defined, the local knot intervals for any control point's blending function may be easily determined as follows: Draw a line through

[2]We will see later on that this statement is not true for *all* control points in a T-spline—there are some exceptions.

[3]Since we will only be discussing degree three T-splines, we dispense with the degree-specifying superscript here to avoid clutter.

[4]In this thesis we limit our discussion to orientable surfaces (2-manifolds) with or without boundaries having four-sided faces only. While T-splines do support n-sided faces and non-orientable surfaces, these topological constructions are needlessly complex and beyond the scope of this work.

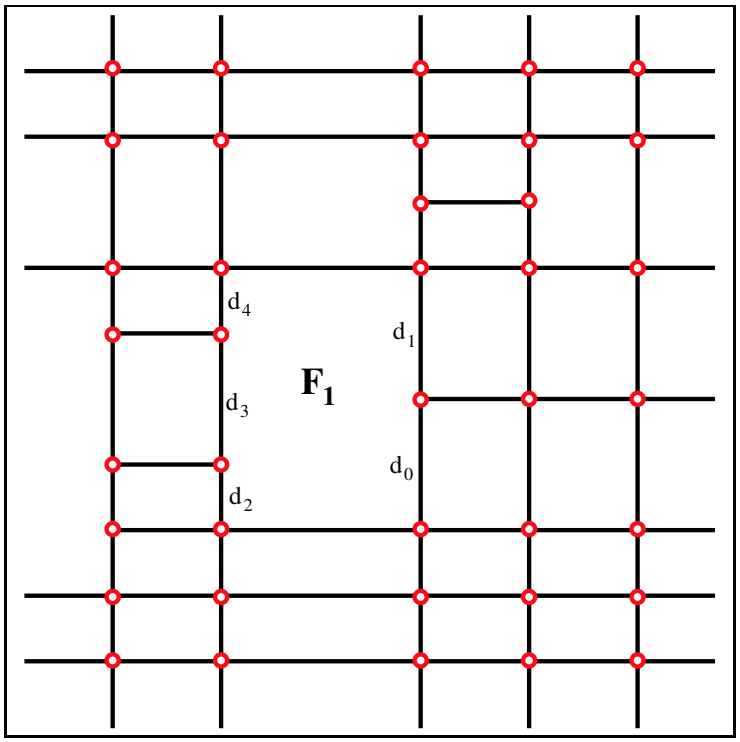
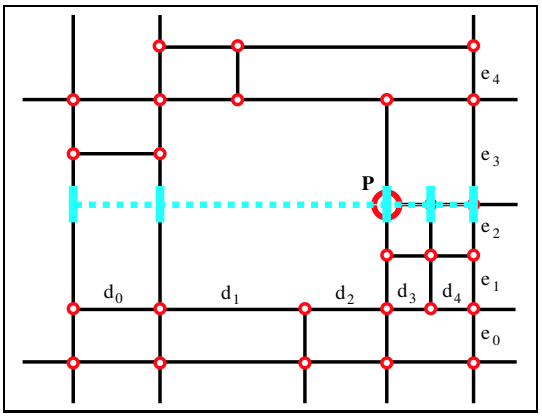
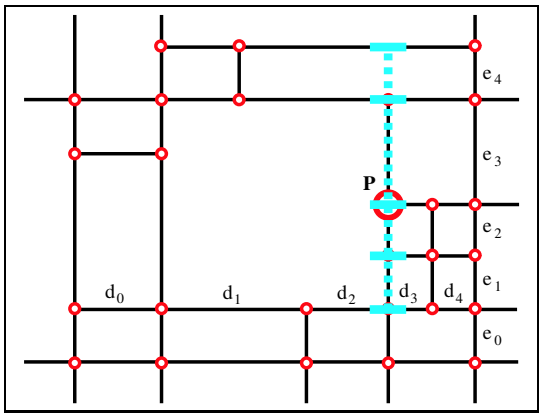


Figure 2.10: A portion of a T-spline control grid. Only select knot intervals are labeled in the figure.



(a) The s intervals for control point **P** are $[d_0, d_1 + d_2, d_3, d_4]$.



(b) The t intervals for control point **P** are $[e_1, e_2, e_3, e_4]$.

Figure 2.11: Identifying local knot intervals in a T-spline.

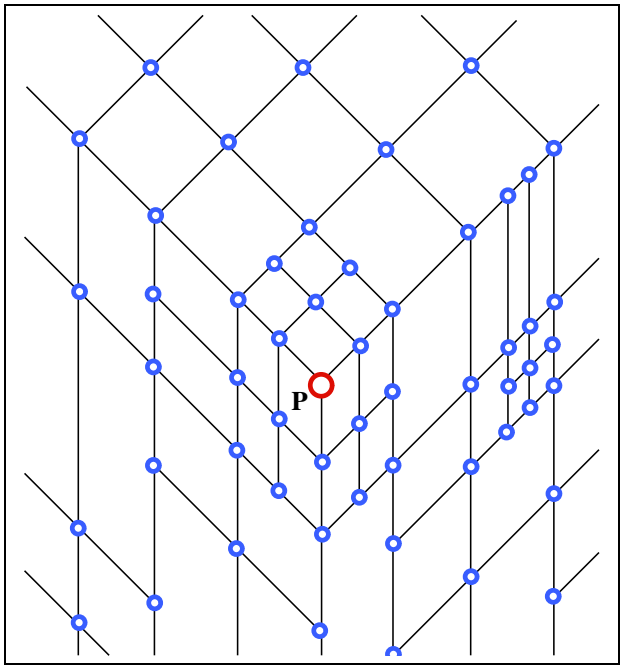


Figure 2.12: A portion of a T-spline with arbitrary topology.

the control point along each of the parameter directions (s and t). Then, each control point or edge that the line crosses divides the line into segments, each of which has an interval length. The two line segment intervals closest to each side of the control point comprise the four local knot intervals for the point in the parameter direction of the line. For example, Figure 2.11(a) identifies the local knot intervals for point **P**'s s parameter and Figure 2.11(b) identifies the local knot intervals for point **P**'s t parameter using the described method.

We can now determine blending functions for most of the control points in a T-spline grid. However, there are still some blending functions that cannot be determined using this method. So far, all of the T-spline topologies we have seen have been laid out such that all edges are either horizontal or vertical, but T-splines support other topologies as well. For example, Figure 2.12 represents a valid T-spline topology[5]. Note that the vertex labeled **P** is of valence three, but is not a

[5]The original T-splines paper [14] makes a distinction between two classes of topologies: T-splines and T-NURCCs. In [14] a *T-spline*'s only non-T-junction control points must be of valence four, whereas a *T-NURCC* may contain control points of any valence (T-junction or not). In this thesis,

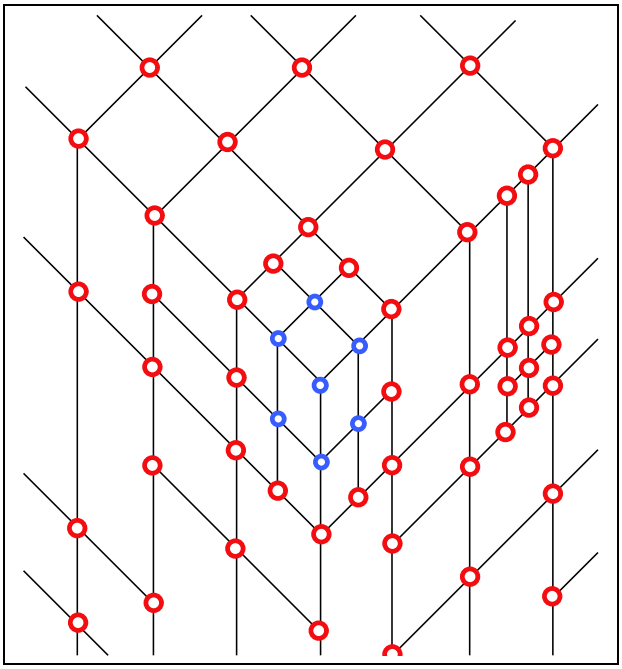


Figure 2.13: Control points with tensor product blending functions in a T-spline with an extraordinary point. The control points with tensor product blending functions are shown in red.

T-junction. As a result, the parameterization of the surface at this point does not map parametrically to a coordinate system of two parameters—we cannot use our method for determining local knot intervals here. Vertex **P** is called an *extraordinary point* and all non-T-junction vertices not of valence four will produce an unusual parameterization and are therefore extraordinary points.

In spite of this change in parameterization caused by extraordinary points, our method for determining tensor product blending functions applies to many of the control points in the T-grid. Figure 2.13 emphasizes the control points whose blending functions may still be derived with this method. Note that this case demonstrates the usefulness of knot interval notation—a global knot vector would not work in this case, but it is handled naturally by knot intervals. For the remaining points (those not highlighted in Figure 2.13), we will use subdivision surface rules to determine the

we do not adopt this nomenclature, but group both classes under the term T-spline. This is mostly for ease of discussion, since under [14]'s terminology T-splines are proper subsets of T-NURCCs.

blending functions of the non-tensor product control points. Specifically, the rules derived in [12] will be used.

Because of the its origins, the T-spline is exactly compatible with two popular surface types: In the uniform knot interval case with no T-junctions in the mesh, a T-spline is equivalent to a Catmull-Clark subdivision surface. In the case where there there are no extraordinary points, the T-spline is equivalent to a NURBS surface (although the NURBS surface may have considerably more control points).

With all of the blending functions of the T-spline defined, it is then possible to compute the surface using (2.4). We stress here that, by definition, a T-spline's blending functions are tightly coupled with the T-grid, because the blending functions are directly derived from it (via the grid's knot intervals). This observation may seem unnecessary, but as we develop our simplification methods later on, we will see examples of blending functions that are temporarily decoupled from the T-grid.

2.2 Blending Function Refinement

In developing methods for T-spline simplification, one of the atomic operations used is *blending function refinement*. This operation originates directly from work on knot insertion (see [5] or [2]). In this section we focus specifically on the refinement of the tensor product B-spline blending functions of a T-spline, which differs only slightly from general knot insertion. Note that we do not cover the refinement of extraordinary point blending functions here; that topic is covered in Appendix A.

The refinement of tensor product B-spline blending functions is a special case of knot insertion. Refinements may be performed on each of the parameter directions separately. Therefore, the refinement operation may be simplified to a knot insertion into a univariate B-spline blending function.

A B-spline basis function can be treated as a special case of a B-spline curve. The ordinates of the curve are given such that only the central value is equal to

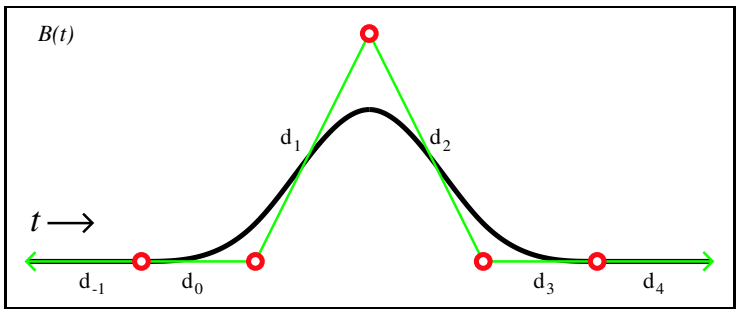


Figure 2.14: A B-spline basis function with arbitrary knot intervals. The shape of the curve is not affected by the values of the intervals d_{-1} and d_4. The function's control point ordinate values are (0,0,1,0,0) from left to right.

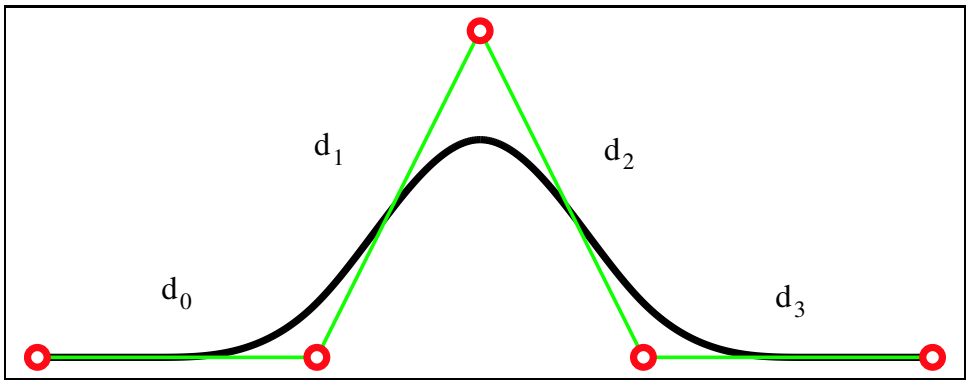
one with all other ordinates equal to zero. Since the ordinates are given, only the knot intervals alter the shape of the curve. A B-spline basis function with arbitrarily labeled knot intervals is shown in Figure 2.14.

A single insertion into a B-spline basis function produces two scaled B-spline basis functions, whose sum is exactly equal to the original. The knot intervals of the scaled basis functions match those of the original, but one of the intervals is split—the interval at which the refinement occurred. Analytically, we express this idea as

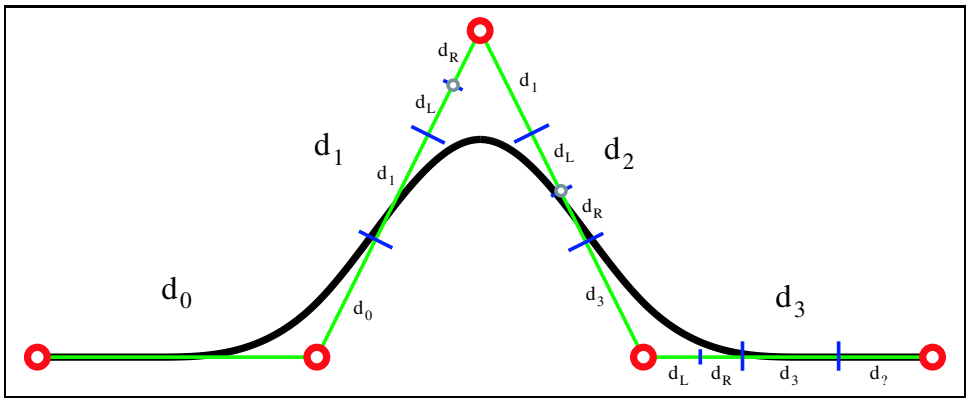
$$B_i(u) \quad = \quad c_1 B_{i_1}(u) + c_2 B_{i_2}(u). \tag{2.5}$$

The values of the scaling coefficients depend solely on the knot intervals of the original blending function and the parameter value at which an interval in that function is split—an example will illustrate.

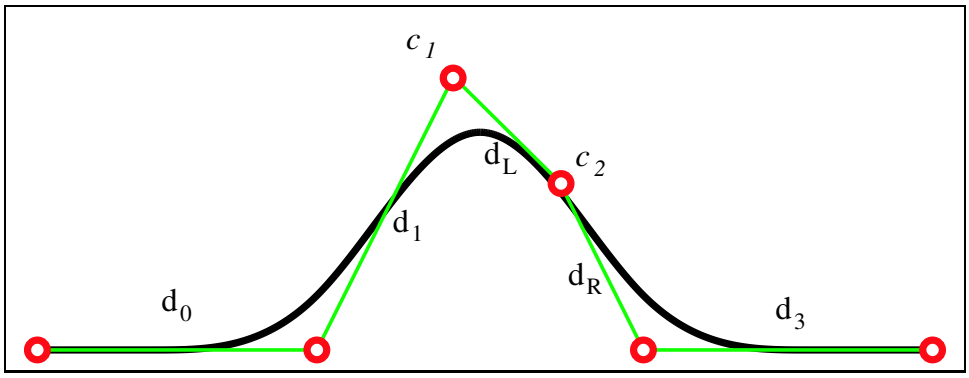
The control polygon for a degree three B-spline basis function is given in Figure 2.15(a) with knot intervals labeled (d_0, d_1, d_2, d_3). Suppose we wish to split d_2 into intervals, d_L and d_R. We can perform the split geometrically, by mapping the new intervals and each of the interval neighbors onto the associated edges of the control polygon as illustrated in Figure 2.15(b). Then, to compute the refinement, we derive the ordinate values of the new control polygon's vertices, using our annotated original control polygon. The positions of the new ordinates are illustrated in Figure 2.15(c).



(a) A B-spline basis function, before refinement.



(b) The B-spline basis function labeled in preparation for refinement. The unknown interval $d_?$'s value is not needed to compute the refinement, since the ordinate value will equal zero in every case.



(c) The refined B-spline basis function.

Figure 2.15: Basis Function Refinement using Knot Insertion.

From this figure, we can derive expressions for the two non-zero ordinates resulting from the refinement:

$$c_1 = \frac{d_0 + d_1 + d_L}{d_0 + d_1 + d_2} \tag{2.6}$$

$$c_2 = \frac{d_R + d_3}{d_1 + d_2 + d_3}. \tag{2.7}$$

Note that after refinement the blending functions $B_{i_1}(u)$ and $B_{i_2}(u)$ have knot intervals (d_0, d_1, d_L, d_R) and (d_1, d_L, d_R, d_3), respectively. Observe, also, that although d_2 maps onto the edge with interval d_3, the ordinate value resulting from the insertion on this edge is always zero. Hence, the unknown interval $d_?$ value is not needed to compute the refinement.

In our example, we derived expressions for c_1 and c_2 when the interval d_2 was split. We can similarly derive expressions for splits of the remaining intervals. For ease we summarize all of the expressions in Table 2.1. In each case, the ordinate values and the blending functions are given in increasing parametric order (left-to-right).

Split Interval	c_1 *value*	c_2 *value*	*Blending Function Intervals*
d_0	$\frac{d_L}{d_0+d_1+d_2}$	1	$B_{i_1} : (d_L, d_R, d_1, d_2)$ $B_{i_2} : (d_R, d_1, d_2, d_3)$
d_1	$\frac{d_1+d_L}{d_0+d_1+d_2}$	$\frac{d_R+d_2+d_3}{d_1+d_2+d_3}$	$B_{i_1} : (d_0, d_L, d_R, d_2)$ $B_{i_2} : (d_L, d_R, d_2, d_3)$
d_2	$\frac{d_0+d_1+d_L}{d_0+d_1+d_2}$	$\frac{d_R+d_3}{d_1+d_2+d_3}$	$B_{i_1} : (d_0, d_1, d_L, d_R)$ $B_{i_2} : (d_L, d_R, d_2, d_3)$
d_3	1	$\frac{d_R}{d_1+d_2+d_3}$	$B_{i_1} : (d_0, d_1, d_2, d_L)$ $B_{i_2} : (d_1, d_2, d_L, d_R)$

Table 2.1: Ordinate Values Resulting from Blending Function Refinement

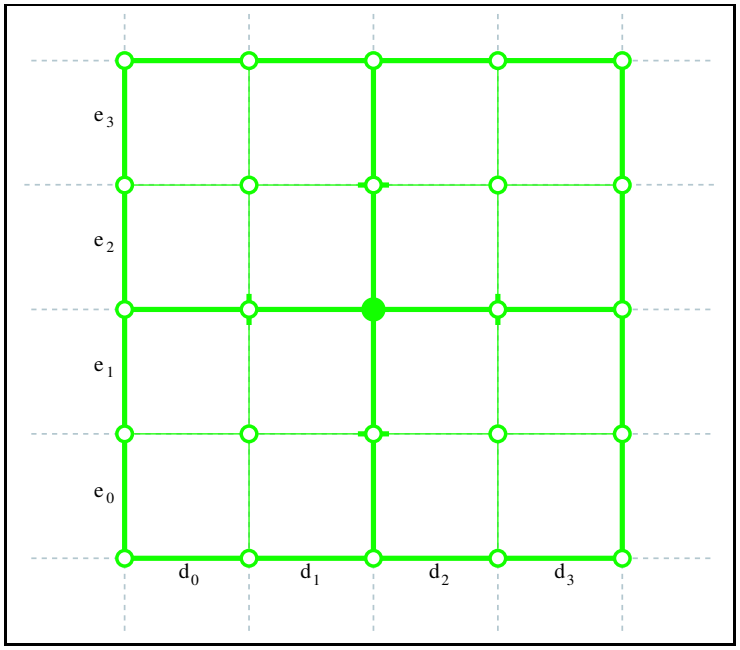


Figure 2.16: The central control point has an ordinate value of one, while all other ordinates are zero (in further diagrams, only the central ordinate will be shown). All of the knot intervals shown here are equal to two.

The principle idea of this refinement operation, is expressed in Equation 2.5—that a single B-spline blending function may be split into the sum of two scaled blending functions with slightly different knot intervals. We can, of course, apply the knot insertion operation multiple times to the blending function, breaking it into the sum of several scaled blending functions.

We end this section by illustrating refinement on a bivariate blending function with uniform knot intervals. This exercise should help solidify the refinement operation and process, which we will employ extensively in the simplification methods of Chapters 4 and 5. The initial B-spline basis function on which we will operate is depicted in Figure 2.16. Note that both the s and t knot intervals are initially all equal to two. The central ordinate of the blending function is equal to one with all remaining ordinates at zero.

We will begin refining $B(s,t)$ halfway through interval d_1. To do so, we apply the corresponding rule from Table 2.1 (where $d_L = d_R = 1$) to produce two new

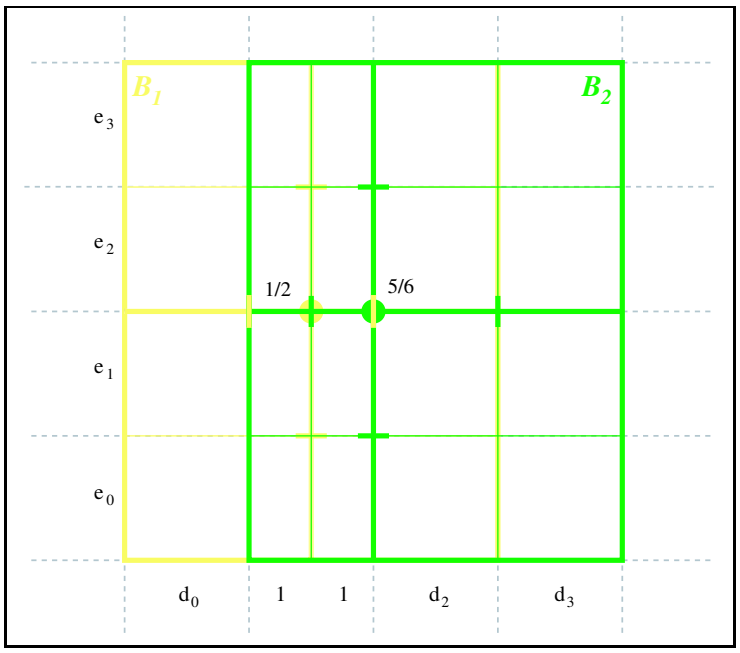


Figure 2.17: Result of refining a bivariate blending function with central ordinates labeled.

blending functions: $B_1(s,t)$ and $B_2(s,t)$. This works out to

$$B(s,t) \;=\; \frac{1}{2}B_1(s,t) + \frac{5}{6}B_2(s,t), \tag{2.8}$$

where B_1 and B_2 have t intervals $(2,1,1,2)$ and $(1,1,2,2)$, respectively. The result of this operation is illustrated graphically in Figure 2.17.

Now that B_1 and B_2 have been generated, we may operate on them separately. For instance, refining B_1 halfway through interval e_0 produces $B_{1,1}$ and $B_{1,2}$:

$$B_1(s,t) \;=\; 1 \cdot B_{1,1}(s,t) + \frac{1}{6}B_{1,2}(s,t), \tag{2.9}$$

where the s knot intervals for $B_{1,1}$ and $B_{1,2}$ are $(1,1,2,2)$ and $(1,2,2,2)$, respectively. Similarly, we may refine B_2 by dividing e_2 in half, where the resulting expression is

$$B_2(s,t) \;=\; \frac{5}{6}B_{2,1}(s,t) + \frac{1}{2}B_{2,2}(s,t), \tag{2.10}$$

where the s knot intervals for $B_{2,1}$ and $B_{2,2}$ are $(2,2,1,1)$ and $(2,1,1,2)$, respectively. Notice that the s knot intervals for $B_{2,1}$ and $B_{2,2}$ are quite different from those for $B_{1,1}$ and $B_{1,2}$, even though they originated from the same initial blending function.

By combining (2.8), (2.9) and (2.10), we can produce a single equation for the original blending function:

$$\begin{aligned} B(s,t) &= \frac{1}{2}B_1(s,t) + \frac{5}{6}B_2(s,t) \\ &= \frac{1}{2}\left(B_{1,1}(s,t) + \frac{1}{6}B_{1,2}(s,t)\right) + \frac{5}{6}\left(\frac{5}{6}B_{2,1}(s,t) + \frac{1}{2}B_{2,2}(s,t)\right) \\ &= \frac{1}{2}B_{1,1}(s,t) + \frac{1}{12}B_{1,2}(s,t) + \frac{25}{36}B_{2,1}(s,t) + \frac{5}{12}B_{2,2}(s,t). \end{aligned}$$

The scaled blending functions are shown together graphically in Figure 2.18.

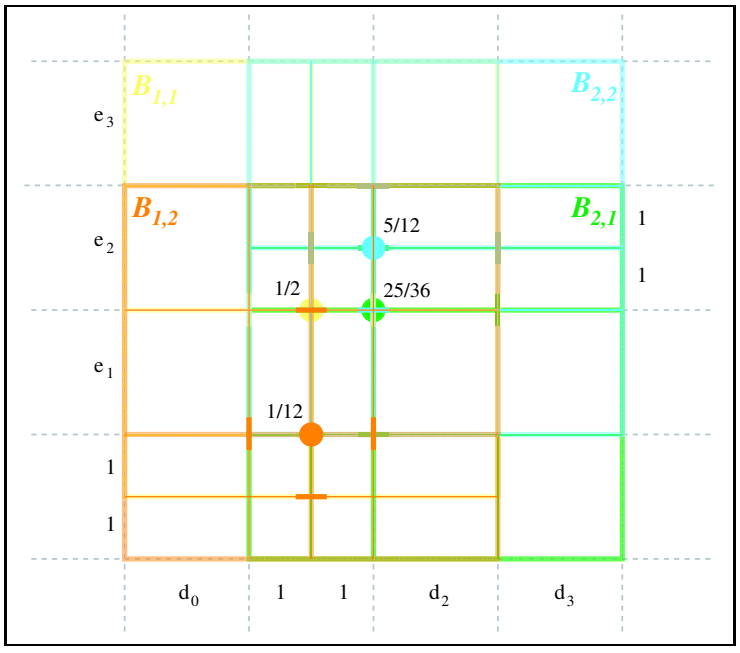


Figure 2.18: The original blending function refined to four smaller blending functions. Each blending function's central ordinate is labeled with its value.

2.3 Least-Squares Fitting of Simplified T-Splines

Since removing control points from a T-spline surface is not exact in general, our simplified surface will be an approximation of the original. To produce the approximation, we will use least-squares fitting as presented in [16].

Linear least-squares fitting is a well-researched topic, which is covered in most introductory linear algebra textbooks[20]. In addition, several methods for performing matrix operations on large (sparse and dense) systems have also been developed ([21] is a good resource, but there are many others). This section reviews linear least-squares fitting and its application to T-splines.

In general, least-squares is an optimization problem. That is, given a function $f(x_0, \ldots, x_n; \beta_0, \ldots, \beta_m)$ adjust the parameters of the function $(\beta_0, \ldots, \beta_m)$, such that the squared-distance from the function to some set of input data (either continuous or discrete values of $x_0, \ldots, x_n$) is minimized. We can express the function to be minimized with

$$S(\beta_0, \ldots, \beta_m) = \int [f(x_0, \ldots, x_n; \beta_i) - g(x_0, \ldots, x_n)]^2 \, dx_0 \ldots dx_n \qquad (2.11)$$

in the continuous case (where g is the known function) and

$$S(\beta_0, \ldots, \beta_m) = \sum_{i=0}^{k} [f(X_{1,i}, \ldots, X_{m,i}; \beta_i) - G_i]^2 \qquad (2.12)$$

in the discrete case—where G_i are the k discrete samples corresponding to $f(X_{1,i}, \ldots, X_{n,i})$. As taught in most introductory calculus classes, this type of problem is solved by finding the zeros of the function's first derivative ($\nabla S = 0$), where the partial derivatives are taken with respect to the function parameters $(\beta_0, \ldots, \beta_m)$. In the case where f is linear in these parameters, the problem reduces to a system of linear equations and is faster and easier to solve.

Applying this general framework to the least-squares fitting of T-spline surfaces, we rewrite (2.11) substituting (2.4) for f:

$$S(\mathbf{P_0},\ldots,\mathbf{P_n},w_0,\ldots,w_n) \;=\; \iint\left[\frac{\sum_{j=0}^{n}\mathbf{P_j}w_jB_j(s,t)}{\sum_{j=0}^{n}w_jB_j(s,t)}-\mathbf{G}(s,t)\right]^2 ds\,dt \quad (2.13)$$

where $\mathbf{G}(s,t)$ is the known surface being approximated by $\mathbf{P}(s,t)$. As should be immediately evident, $\mathbf{P}(s,t)$ is linear in its control point parameters, but non-linear in the control point weights. Thus, linear least-squares may be applied to a T-spline surface only if the weights are known or may somehow be determined. In order to guarantee this condition for simplification, we will derive an important relationship between $\mathbf{G}(s,t)$ and $\mathbf{P}(s,t)$.

2.3.1 T-Spline Spaces

Let us define a *T-spline space*[6] as the set of all T-splines that have the same mesh topology, knot intervals and knot coordinate system—essentially, members of a T-spline space may differ only in their geometries and weights. Clearly, if $\mathbf{P}$ and $\mathbf{G}$ belong to the same T-spline space an exact solution to (2.13) exists.

A T-spline space $\mathcal{S}_1$ is called a subspace of $\mathcal{S}_2$ (denoted $\mathcal{S}_1 \subset \mathcal{S}_2$) if refining a T-spline in $\mathcal{S}_1$ will produce a T-spline in $\mathcal{S}_2$[7]. In other words, the topology and knot intervals of $\mathcal{S}_2$ may be attained by refining any T-spline in $\mathcal{S}_1$. A *nested sequence* $(\mathcal{S}_1 \ldots \mathcal{S}_n)$ of T-spline satisfies $\mathcal{S}_1 \subset \mathcal{S}_2 \subset \cdots \subset \mathcal{S}_{n-1} \subset \mathcal{S}_n$. Figure 2.19 illustrates the pre-images of such a nested sequence.

In the context of least-squares fitting, it would be helpful if we could guarantee that we can apply linear least-squares fitting to any pair of T-splines $T_1 \in \mathcal{S}_1$ and $T_2 \in \mathcal{S}_2$ where $\mathcal{S}_1 \subset \mathcal{S}_2$. However, in a T-spline space, the weights in (2.13) remain free parameters and we do not yet have a method of determining them in a linear

[6] This definition relates closely to the definition of a spline space in [16].
[7] T-spline refinement operations are discussed in detail in Chapters 4 and 5

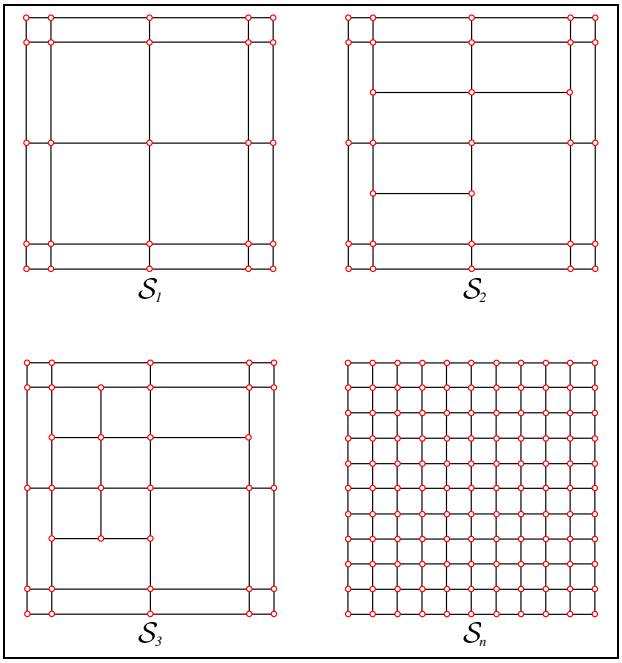


Figure 2.19: A nested sequence of T-spline spaces. $\mathcal{S}_i$ may be generated by refining $\mathcal{S}_{i-1}$ or any of its subsets.

least-squares fit. Because the weights prevent a linear fit, we define a *weighted T-spline space* to include mesh topology, knot intervals, knot coordinate system and weights (denoted $\mathcal{S}^w$). Only the control point positions differ among members of a weighted T-spline space.

The advantage of weighted T-spline spaces is that linear least-squares fitting of a T-spline may be performed in a weighted T-spline space to any T-spline in its superspace *even if the weights of the T-spline being fitted are not known!* To see why this is the case, we restate (2.13):

$$S(\mathbf{P_0},\ldots,\mathbf{P_n},w_0,\ldots,w_n) \;=\; \iint\left[\frac{\sum_{j=0}^{n}\mathbf{P_j}w_jB_j(s,t)}{\sum_{j=0}^{n}w_jB_j(s,t)}-\mathbf{G}(s,t)\right]^2 ds\,dt$$

In this equation, we assume that $\mathbf{P}(s,t) \in \mathcal{S}_a^w$, that $\mathbf{G}(s,t) \in \mathcal{S}_b^w$ and that $\mathcal{S}_a^w \subset \mathcal{S}_b^w$. This implies that $\mathbf{G}(s,t)$ is a T-spline also and therefore expands to

$$\mathbf{G}(s,t) = \frac{\sum_{i=0}^{\tilde{n}} \tilde{\mathbf{P}}_\mathbf{i} \tilde{w}_i \tilde{B}_i(s,t)}{\sum_{i=0}^{\tilde{n}} \tilde{w}_i \tilde{B}_i(s,t)},$$

where $\mathbf{G}(s,t)$'s control points, weights and blending functions are denoted with a overscript tilde. Performing this substitution, we can now rewrite 2.13 as

$$\begin{aligned} & S(\mathbf{P_0}, \ldots, \mathbf{P_n}, w_0, \ldots, w_n) \\ &= \iint \left[\frac{\sum \mathbf{P_j} w_j B_j(s,t)}{\sum w_j B_j(s,t)} - \frac{\sum \tilde{\mathbf{P}}_\mathbf{i} \tilde{w}_i \tilde{B}_i(s,t)}{\sum \tilde{w}_i \tilde{B}_i(s,t)} \right]^2 ds\, dt \end{aligned} \tag{2.14}$$

Because of the space relationship between $\mathbf{G}(s,t)$ and $\mathbf{P}(s,t)$, we know that the denominators of their expanded expressions are equal:

$$\sum_{j=0}^{n} w_j B_j(s,t) = \sum_{i=0}^{\tilde{n}} \tilde{w}_i \tilde{B}_i(s,t). \tag{2.15}$$

Therefore, we can expand (2.15) to a system of linear equations and solve for w_i exactly[8] After doing so, we may similarly derive a linear system and use linear least-squares fitting to solve for the geometries of $\mathbf{P}(s,t)$ as well.

2.3.2 Linear Least-Squares

This section explores linear least-squares T-spline fits in more detail. T-spline refinement creates a set of new blending functions[9] and each blending function before

[8]Note that this reasoning rests on an unproven conjecture on the linear independence of the T-spline's blending functions. So far, no counterexample has been produced and the conjecture is believed to be true.

[9]Precisely how the new blending functions are computed via refinement is the topic of the next chapter.

refinement may be written as a linear combination of the new functions:

$$B_i(s,t) \;=\; \sum_{j=0}^{\tilde{n}} c_{i,j}\tilde{B}_j(s,t), \tag{2.16}$$

where the B_i are the blending functions of the original T-spline ($T_1 \in \mathcal{S}_1^w$) and the $\tilde{B}_j$ are the blending functions of the refined T-spline ($T_2 \in \mathcal{S}_2^w$). Each of the refined blending functions is assigned a non-negative scalar weight $c_{j,i}$. Since $\mathcal{S}_1^w \subset \mathcal{S}_2^w$ must hold for a linear fit, we require that (2.15) holds and substitute for B_i using (2.16):

$$\sum_{i=0}^{n} w_i \left[\sum_{j=0}^{\tilde{n}} c_{i,j}\tilde{B}_j(s,t)\right] \;=\; \sum_{j=0}^{\tilde{n}} \tilde{w}_j\tilde{B}_j(s,t). \tag{2.17}$$

Expanding this equation and combining blending function terms yields a linear system:

$$\mathbf{M}_{2\leftarrow 1}\mathbf{w} \;=\; \tilde{\mathbf{w}} \tag{2.18}$$

where $\mathbf{w} = [w_0,\ldots,w_n]^T$, $\tilde{\mathbf{w}} = [\tilde{w}_0,\ldots,\tilde{w}_{\tilde{n}}]^T$ and $\mathbf{M}_{2\leftarrow 1}$ is the linear transformation matrix, whose entry at row i and column j is $c_{i,j}$. We call such a matrix a *T-spline refinement matrix*, where $\mathbf{M}_{j\leftarrow i}$ computes the refinement of T_i to T_j.

Since T_2 has more blending functions than T_1, $\mathbf{M}_{2\leftarrow 1}$ has more rows than columns. This implies that the system is over-constrained. However, for the weights, we require that an exact solution exists. Thus, we may solve the system in the least-squares sense to produce the exact solution for T_1's weights.

We next proceed to calculate T_1's control points. Fortunately, this derivation falls out very cleanly. The denominators in (2.14) are equal and known, so we will rewrite them simply as a weight function, $w(s,t)$:

$$S(\mathbf{P_0},\ldots,\mathbf{P_n},w_0,\ldots,w_n) \;=\; \iint \left[\frac{\sum \mathbf{P_j} w_j B_j(s,t) - \sum \tilde{\mathbf{P}}_\mathbf{i}\tilde{w}_i\tilde{B}_i(s,t)}{w(s,t)}\right]^2 ds\,dt.$$

Then, by coupling weights and geometries (denoted $\mathbf{P_i^w} = w_i\mathbf{P_i}$), we can simplify this expression even further:

$$S(\mathbf{P_0^w}, \ldots, \mathbf{P_n^w}) \quad = \quad \iint \left[\frac{\sum \mathbf{P_j^w} B_j(s,t) - \sum \tilde{\mathbf{P}}_\mathbf{i}^\mathbf{w} \tilde{B}_i(s,t)}{w(s,t)} \right]^2 ds\, dt.$$

From this point, solving for the weighted geometries is derived in the same way as solving for the weights—the known weights function ($w(s,t)$) factors out of both sides of the equation just as $\tilde{B}_j(s,t)$ does in (2.17). So, solving for $\mathbf{P_i^w}$ to minimize $S(\mathbf{P_0^w}, \ldots, \mathbf{P_n^w})$ means solving

$$\mathbf{M}_{2\leftarrow 1}\mathbf{P^w} \quad = \quad \tilde{\mathbf{P}}^\mathbf{w}$$

in the least-squares sense. Essentially, this means that we can combine the control point geometries ($\mathbf{P_i} = (x_i, y_i, z_i)$) and their weights into a four-tuple ($w_i{\cdot}x_i, w_i{\cdot}y_i, w_i{\cdot}z_i, w_i$) and then use the same method to solve for each component.

Before concluding this chapter, it is important to note that methods do exist for performing non-linear least squares fits of NURBS surfaces to discrete data points. See [22], [23] and [24] for some approaches. Unfortunately, these methods are complex, slow and do not easily extend to the continuous case. In addition, the added degrees of freedom of T-spline weights actually increase the complexity of the simplification problem. Therefore, we limit this thesis to the linear least-squares cases. Application of these non-linear fitting methods to T-spline simplification may be a fruitful direction for future research.

Chapter 3

Related Work

Although T-spline surfaces are new, techniques to simplify NURBS and other surface types have been given some attention. Many of the principles and methods in such techniques relate to T-spline simplification, so we will review them here. First, in Section 3.1 we define specifically the bounds of the simplification problem that this thesis addresses and then review related simplification methods. NURBS surface simplification is reviewed in Section 3.2 and subdivision surface simplification is covered in Section 3.3. Also, since T-spline knot insertion is a significant contribution to this thesis, Section 3.4 reviews research related to that topic.

3.1 Simplification

As overviewed in the introduction, simplification is the generation of an exact or approximate representation of a free-form surface model that has fewer control points than the original. While concise, this definition differs slightly from the simplification problem addressed in this thesis and its related work, which has a few additional constraints:

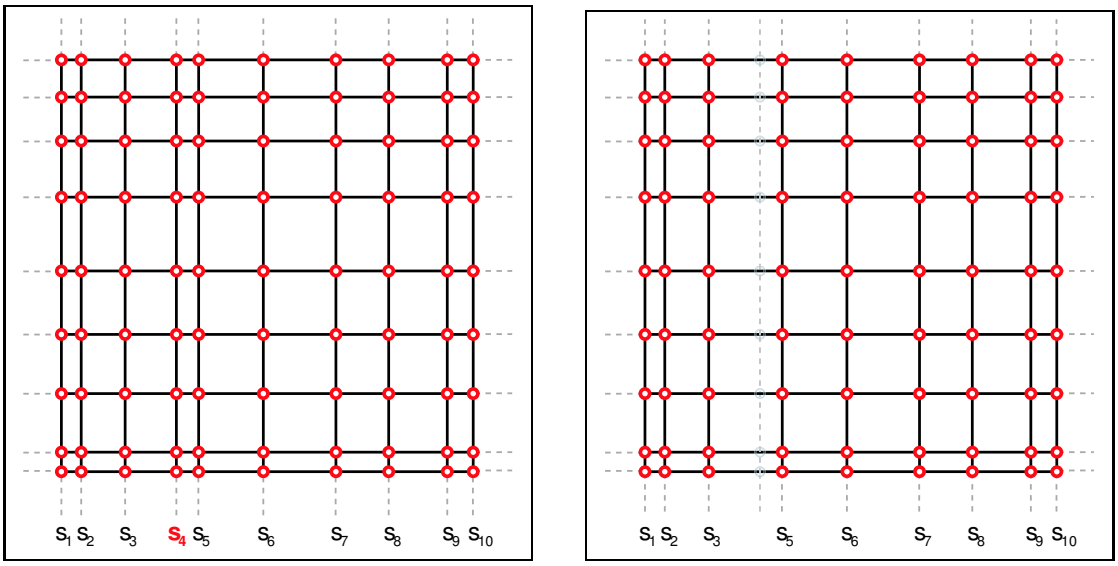
- The parameterization of the surface is not altered by the simplification. For many models there may exist simpler representations that alter the parameterization of the surface, but identifying such models focuses more on re-parameterization than on the reduction of the input model.

- The degree of the surface is not reduced through the simplification. In certain cases, a reduction of control points may be attained for a free-form surface by reducing the degree of the polynomials that represent the surface. We consider specifically the degree three case and do not alter the degree of the surface to reduce control points.
- The parametric position of control points in the control mesh is not altered in the simplification. Each control point in a NURBS, subdivision or T-spline surface exists at a specific parameter position. Some simplifications for a surface could be generated by removing existing control points from a mesh, inserting a new (not formerly removed) control point into the mesh and then calculating an approximation. Since we view each control point as storing some geometric information, we constrain our simplification to eliminating redundant control points already in a surface's control mesh.
- As mentioned in the last chapter, we consider only the simplifications where a linear least-squares fit is possible. The additional difficulties inherent in the non-linear fit are not addressed as part of this work and have yet to be addressed with NURBS surface simplification.

To summarize these constraints, we approach the simplification problem as the removal of control points from an existing control mesh while keeping the other properties of the mesh (parameterization, degree and the weight surface) fixed.

3.2 NURBS Surface Simplification

For the most part, simplification methods for NURBS surfaces are direct extensions of methods developed for simplifying NURBS curves. The majority of NURBS simplification methods treat first the curve case and then apply the same method to the surface case. Due to the rigid topological restrictions of NURBS surfaces, one is really



(a) The knot s_4 is selected to be removed.

(b) Removing s_4 requires that all control points in its column be removed, because the NURBS surface definition requires all control points to be arranged in an array.

Figure 3.1: Removing a single knot from a NURBS surface requires the removal of all control points residing on that knot.

forced to take this approach, since no directly local simplification is permitted. Thus, most of the NURBS surface literature related to T-spline simplification treats mainly curve simplification.

The goal of simplification is the removal of control points from a curve or surface. In a NURBS curve, control points correspond to a non-decreasing set of parameter values called *knots*. These knots are typically presented in an ordered list called a *knot vector* [2]—e.g., $[t_0, \ldots, t_n]$. The number of control points present in a NURBS curve is a function of the number of knots. Because of this functional relationship, NURBS control point removal is often referred to as *knot removal*. Since NURBS surfaces are bivariate, one knot vector is associated with each parameter variable and NURBS surface control points correspond in number to the product of the number of knots in the two knot vectors. Accordingly, while a single knot removal

from a NURBS curve results in the removal of a single control point, knot removal from a NURBS surface results in the removal of an entire row or column of control points as shown in Figure 3.1. Notably, this poses a significant limitation on knot removal for NURBS surfaces—a limitation which T-splines overcomes.

One of the few global approaches to NURBS curve and surface knot removal is presented by Lyche and Mørken in [25]. This approach develops a process for selecting knots to be removed from a curve or surface. The process divides the removal task into three distinct operations: *Rank*, *Remove* and *Approximate*. In the *Rank* operation, each knot is assigned a ranking number related to the maximum perturbation produced by fitting the surface with the removed knot to that of the original surface. Ranking numbers are used instead of the perturbation metric directly in order to promote uniform knot removal. Once knots have been ranked, the *Remove* strategically removes knots until the maximum number of removals (within a certain perturbation tolerance) is reached. *Remove* uses the *Approximate* operation to determine perturbation errors, by fitting a simplified NURBS to the original. By performing a binary search on the number of possible removals, the *Remove* operation finds a reasonable set of knots to remove from a NURBS curve or surface.

An alternative approach to [25]'s global knot removal is that of spline decomposition [16] by Dæhlen and Lyche. This research presents a generalized method for fitting any type of spline to an input data set. The complex input splines are fitted with an over-simplified approximation, which is then successively refined until a fit within the desired tolerance is achieved. For NURBS curves and surfaces the refinement steps are demonstrated using knot insertion. When a parametric region of the approximation has an error above tolerance, that region is refined at a knot value in the input model that divides the knots in the region in half. Dæhlen and Lyche's work also presents the idea of nested sequences of spline spaces, which defines a relationship between a simple mesh and refinements of that mesh. Essentially, the

nested sequence concept states that two meshes are members of the same sequence if the topology of one mesh may be attained via refinement of the other. The extension of this concept to T-splines is an important one that was described in the previous chapter.

Related closely to NURBS decomposition, hierarchical NURBS [13] presents a method for editing a NURBS model at multiple resolutions. The hierarchical NURBS scheme permits local refinement of a model, but does so at the cost of maintaining a hierarchy. Each level of the NURBS hierarchy stores geometry offsets from the previous (coarser) level. This framework allows surface offsets to be localized to a desired region of the mesh, permitting local refinement. Forsey and Wong [26] use the hierarchical NURBS framework to automatically generate a simpler, locally refined surface. The algorithm presented in [26] generates each level of the hierarchy at half the resolution of the previous level with added smoothness constraints and curvature penalties. Since this method closely resembles T-spline simplification (it analogously approximates a more complex NURBS with a locally refined surface representation), this proves a useful comparison to the results demonstrated in this thesis.

Some researchers approach knot removal as simply the inverse of knot insertion. For example, Tiller [27] derives an algorithm for exact removal of a knot from a NURBS curve. Tiller's main contribution is more practical than theoretical: his work is largely pseudocode and centers on presenting a packaged algorithm for removing unnecessary knots without modifying the curve's parameterization or geometry. [27] also includes a proof showing that maximum NURBS control point deviation is an upper bound on curve deviation—a property that is put to good use in methods developed here.

Like Tiller, Eck and Hadenfeld focus on the removal of a single knot from a curve [28]. In their work, they improve somewhat on Tiller, by focusing on deriving formulas that produce good approximations for the discrete ℓ_2 and discrete and con-

tinuous ℓ_∞ norms. One useful contribution in [28] is a proof showing that fitting may be performed locally on a NURBS curve/surface after removal, which may be used in T-spline simplification to expedite the least-squares fitting process.

3.3 Subdivision Surface Simplification

While substantial work has been done on simplifying NURBS surfaces, subdivision surface simplification has been given little significant attention. Most likely due to the complexity that arbitrary topology introduces to the problem, most of the related research attempts to impose a new parameterization onto existing models instead of trying to retain topology during simplification.

One of the few direct subdivision surface simplification approaches is presented in [17] by Jeong *et al.* This work develops a method for simplifying Loop subdivision surfaces, by combining a quadric error metric with simple topological operations on a Loop surface's triangle mesh. Because T-spline surfaces generate rectangular (and not triangular) domains, it is difficult to extend Jeong *et al.*'s approach to T-splines.

There are several re-parameterization-based approaches to the simplification problem. Typically, work in this area takes as input some form of dense data (e.g., point clouds or dense triangle meshes), derives a parameterization (either automatically or semi-automatically) and then fits the resulting subdivision surface to the dense input data. Eck and Hoppe [29], for example, construct a parameterization, to which a network of NURBS patches are fitted using a two-step linear least-squares and parameter correction technique. After initial fitting is performed, the patches are refined to generate a better fit with the input data. Ma and Zhao [30] use a similar approach but reduce the time cost of fitting significantly over Eck and Hoppe.

In [31] He *et al.* generate a parameterization for an existing model and then fit a T-spline model using the generated parameterization. After initial fitting, the

T-spline mesh is locally refined in locations where the fitting error exceeds the user-specified tolerance.

Tangentially, there has been a lot of work done on triangular mesh simplification for multiresolution / level-of-detail representation of models (see for example, [32], [33] and [34]), but techniques in this research area are specialized to triangle meshes and cannot reasonably be extended to subdivision surface (or ultimately T-spline) meshes.

3.4 NURBS Knot Insertion

Refinement is one of the important operations for a NURBS curve or surface and, as such, much research has been devoted to it. Probably the earliest related method is the de Boor algorithm [4] for B-spline curve evaluation. While the de Boor algorithm is specifically for evaluating a NURBS at a given parameter value, most of the subsequent developments relate closely to his method in some way.

As we have already seen in Section 2.1.1, the Boehm algorithm computes the result of inserting a single knot into a NURBS curve [5], and Boehm uses his method to present a supporting proof of the de Boor evaluation algorithm. Concurrent with Boehm, Cohen *et al.*developed the *Oslo Algorithm* [6]. The Oslo Algorithm is presented in iterative and recursive forms and computes the discrete control point positions (the so-called *discrete B-splines*) for a refined NURBS curve as linear combinations of the original curve's control points. Both Boehm's and Cohen *et al.*'s work supply stable and practical means for refining a NURBS curve or surface, making NURBS knot insertion a simple and reliable procedure.

Although knot insertion had been fully developed, new methods for understanding and thinking about NURBS helped lay the foundation for T-spline local refinement. Ramshaw's *blossoming* idea [7] provides an elegant framework for understanding NURBS knot insertion. In *blossoming* each NURBS control point is assigned

a *polar label*, which identifies both the curve's degree and the knot values local to that control point. Ramshaw's useful insight imparts a simple geometric interpretation to NURBS knot insertion and other related operations.

Another perspective that can be useful is the alternative notation of *knot intervals* [12][35]. This alternative notation assigns non-negative knot interval values to the spans between knots in a NURBS curve instead of assigning an absolute knot value to the knots. As we have seen in the previous chapter, knot intervals offer a very intuitive method for computing the insertion of a knot into a NURBS curve.

Chapter 4

T-Spline Simplification using Iterative Refinement

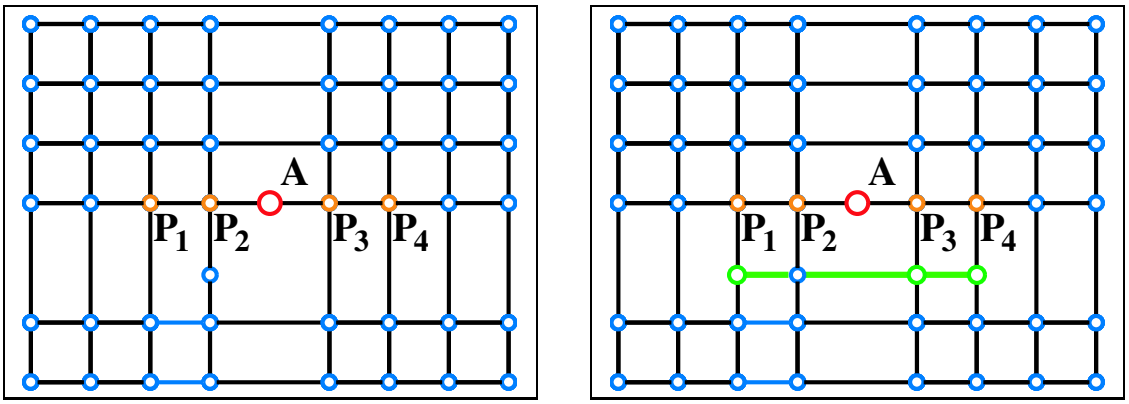
This chapter reviews in detail a method for T-spline simplification based on iterative refinement, published in [1]. Chapter 5 presents a new T-spline simplification method based on iterative simplification.

Implicit in its name, this chapter's approach produces a simplification by applying successive refinements to a model. While refining is not intuitively associated with simplification, the key initial step is to generate an over-simplified version of the model to be simplified and then to refine it until a desired approximation to the original is achieved. In a sense, this approach is a "top-down" approach to the simplification problem, resolving large differences between the simplified model and the original first and then moving on to higher resolution with each step. The next chapter examines the "bottom-up" counterpart to iterative refinement.

Section 4.1 examines T-spline local refinement—an essential operation of this simplification method. Section 4.2 applies local refinement, T-spline spaces and concepts from spline decomposition to develop a method for T-spline simplification.

4.1 T-Spline Local Refinement

To simplify the presentation this section focuses specifically on the refinement of T-splines without extraordinary points. The extension of this algorithm to handle extraordinary points is presented in Appendix A.



(a) Control point **A** is a desired insertion into this T-spline.

(b) The additional control point insertions (shown in green) necessary to permit the insertion of **A**.

Figure 4.1: According to the original T-spline paper [14], **A** may only be inserted if the t knot intervals for the control points $\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}$ and $\mathbf{P_4}$ are equal.

The problem of T-spline local refinement is this: given an input T-spline and a set of control point insertions on the edges of its T-grid, compute the control point positions of the new mesh that will produce a surface equivalent to the original.

The original T-spline paper[14], presented a solution to this problem in which a new control point could only be inserted on an edge if the control points with blending functions affected by the insertion had exactly matching local knot vectors in the parameter direction perpendicular to that of the insertion edge, as illustrated in Figure 4.1. If this condition is not met, some additional control points are first inserted so as to satisfy the condition, as shown in Figure 4.1(b). A problem with this algorithm is that the sequence of additional control point insertions could be long, and it is not clear that the sequence would always terminate. The improved local refinement algorithm overcomes these hurdles.

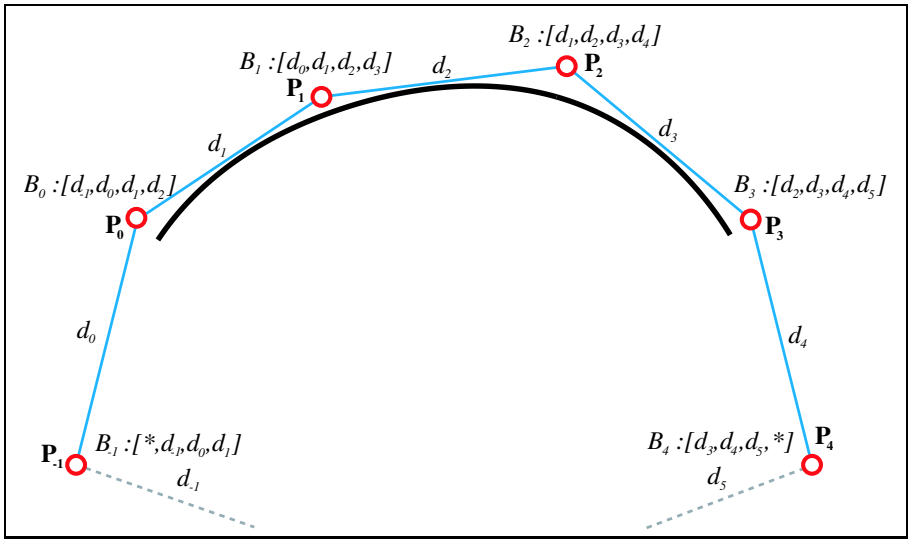


Figure 4.2: Basis functions and their local knot intervals in a simple NURBS curve.

4.1.1 NURBS Knot Insertion Revisited

We begin by reviewing the mathematical foundations of the refinement of a NURBS curve in a way that parallels our development of the T-spline refinement algorithm.

Recall that each control point in a NURBS curve has an associated B-spline basis function that is defined using the neighboring knot intervals on the control polygon. Figure 4.2 shows the knot intervals for the basis functions of each of a curve's control points. If the knot intervals for a basis function match those on the control polygon, we say that the function *agrees with* the control polygon or that the basis function is *coupled* to the control polygon.

If we decouple the B-spline basis functions of the curve from the control polygon (i.e., the knot intervals for each basis function need not agree with the control polygon), then we can add control points to the curve's control polygon without changing the blending functions. If the added control points do not have blending functions associated with them, they will not alter the shape of the curve. This is illustrated in Figure 4.3, in which $\mathbf{P_k}$ is inserted but no blending function is assigned to it, so the curve is unchanged. Note that this is not a NURBS curve, because the blending functions of $\mathbf{P_0}, \mathbf{P_1}, \mathbf{P_2}$ and $\mathbf{P_3}$ do not agree with the control polygon, and

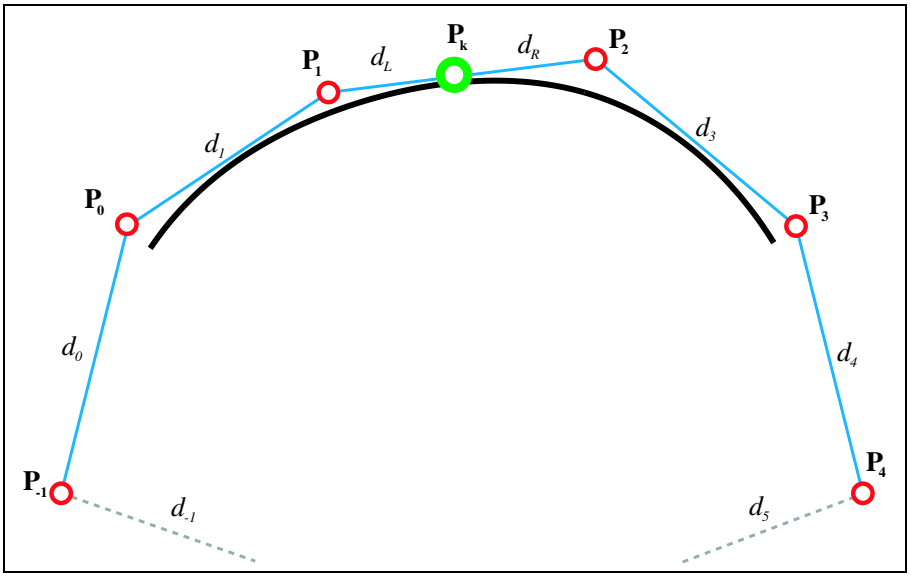


Figure 4.3: Modifying the control polygon. The new control point $\mathbf{P_k}$ does not yet have a geometric position assigned, but is simply placed topologically into the control polygon. When the process completes, $\mathbf{P_k}$ will have both a blending function and a geometric location. Note also that $d_2 = d_L + d_R$.

because $\mathbf{P_k}$ has no blending function. In order to create a valid NURBS, we will need to bring the decoupled blending functions into agreement with the modified control polygon.

To bring each decoupled blending function into agreement with the control polygon, we must modify each offending blending function. The insertion of $\mathbf{P_k}$ has introduced a new knot into the control polygon, so refining each offending blending function to include the new knot will bring it into agreement with the control polygon:

$$
\begin{aligned}
B_0(t) &= B^0_{[d_{-1},d_0,d_1,d_L]}(t) + \frac{d_R}{d_0+d_1+d_2} B^1_{[d_0,d_1,d_R,d_L]}(t) \\
B_1(t) &= \frac{d_0+d_1+d_L}{d_0+d_1+d_2} B^1_{[d_0,d_1,d_L,d_R]}(t) + \frac{d_R+d_3}{d_1+d_2+d_3} B^k_{[d_1,d_R,d_L,d_3]}(t) \\
B_2(t) &= \frac{d_1+d_L}{d_1+d_2+d_3} B^k_{[d_1,d_L,d_R,d_3]}(t) + \frac{d_R+d_3+d_4}{d_2+d_3+d_4} B^2_{[d_L,d_R,d_3,d_4]}(t) \\
B_3(t) &= \frac{d_L}{d_2+d_3+d_4} B^2_{[d_L,d_R,d_3,d_4]}(t) + B^3_{[d_R,d_3,d_4,d_5]}(t)
\end{aligned}
$$

where $B^i_{[a,b,c,d]}(t)$ denotes a B-spline basis function defined by knot intervals $[a, b, c, d]$ and associated with control point $\mathbf{P_i}$.

After refining the affected blending functions to include the newly inserted knot, note that each of the resulting blending functions corresponds those implied by the control polygon. Recalling the equation of the curve ($\frac{\sum \mathbf{P_i} w_i B_i(t)}{\sum w_i B_i(t)}$), we multiply each refined blending function by its coefficients—for convenience we will denote $\mathbf{Q_i} = (x_i \cdot w_i, y_i \cdot w_i, z_i \cdot w_i, w_i)$ and the blending functions of the control polygon[1] with $\tilde{B}_i(t)$:

$$\begin{aligned}
\mathbf{Q_0} B_0(t) &= \mathbf{Q_0} \tilde{B}_0(t) + \frac{d_R}{d_0 + d_1 + d_2} \mathbf{Q_0} \tilde{B}_1(t) \\
\mathbf{Q_1} B_1(t) &= \frac{d_0 + d_1 + d_L}{d_0 + d_1 + d_2} \mathbf{Q_1} \tilde{B}_1(t) + \frac{d_R + d_3}{d_1 + d_2 + d_3} \mathbf{Q_1} \tilde{B}_k(t) \\
\mathbf{Q_2} B_2(t) &= \frac{d_1 + d_L}{d_1 + d_2 + d_3} \mathbf{Q_2} \tilde{B}_k(t) + \frac{d_R + d_3 + d_4}{d_2 + d_3 + d_4} \mathbf{Q_2} \tilde{B}_2(t) \\
\mathbf{Q_3} B_3(t) &= \frac{d_L}{d_2 + d_3 + d_4} \mathbf{Q_3} \tilde{B}_2(t) + \mathbf{Q_3} \tilde{B}_3(t).
\end{aligned}$$

Since these terms are all summed together in the curve equation, we may use the commutativity and associativity of addition to group the expressions in terms of like blending functions. This grouping produces the geometries and weights of the new control points ($\tilde{\mathbf{Q}}_i$) from the original ones:

$$\begin{aligned}
\tilde{\mathbf{Q}}_0 &= \mathbf{Q_0} \\
\tilde{\mathbf{Q}}_1 &= \frac{d_R}{d_0 + d_1 + d_2} \mathbf{Q_0} + \frac{d_0 + d_1 + d_L}{d_0 + d_1 + d_2} \mathbf{Q_1} \\
\tilde{\mathbf{Q}}_k &= \frac{d_R + d_3}{d_1 + d_2 + d_3} \mathbf{Q_1} + \frac{d_1 + d_L}{d_1 + d_2 + d_3} \mathbf{Q_2}
\end{aligned}$$

[1]The equivalence of the control polygon's implied blending functions (of the form $\tilde{B}_i$) with those of the refined, decoupled blending functions (of the form $B^i_{[a,b,c,d]}$) may be quickly verified from Figure 4.3.

$$\tilde{\mathbf{Q}}_2 = \frac{d_R + d_3 + d_4}{d_2 + d_3 + d_4}\mathbf{Q_2} + \frac{d_L}{d_2 + d_3 + d_4}\mathbf{Q_3}$$

$$\tilde{\mathbf{Q}}_3 = \mathbf{Q_3}.$$

This result is equivalent to that produced using the conventional method for NURBS curve refinement.

4.1.2 The T-Spline Local Refinement Algorithm

We now describe an algorithm for T-spline refinement, based on the curve refinement algorithm in Section 4.1.1. The steps of that algorithm can be summarized:

1. **Decouple** the blending functions from the control polygon.
2. **Modify** the control polygon as desired.
3. **Resolve** disagreement by refining offending blending functions.

Step 3 will need some slight modification when applied to T-spline local refinement.

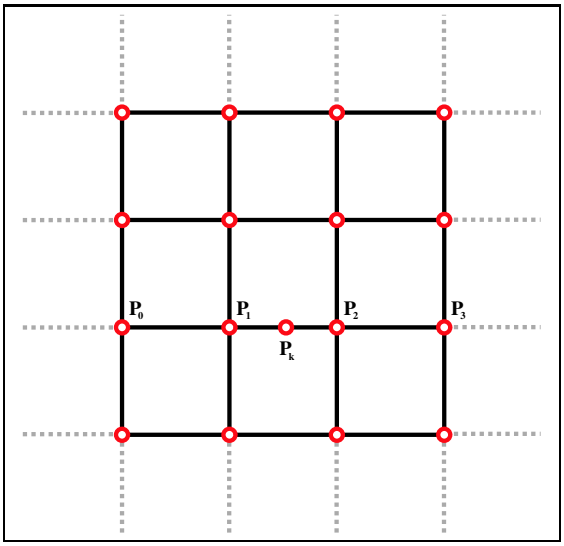
Figure 4.4(a) shows a decoupled T-grid and a newly inserted control point $\mathbf{P_k}$. Figure 4.4(b) labels those blending functions that now disagree with the T-grid. We now move on to the resolution step. By inserting into the intervals of the blending functions, we have:

$$B_0(t) = B^0_{[*,2,2,1][2,2,2,2]}(s,t) + \frac{1}{6}B^1_{[2,2,1,1][2,2,2,2]}(s,t)$$

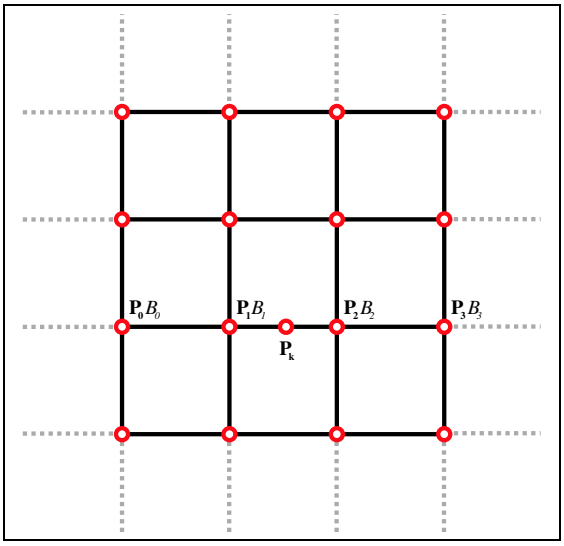
$$B_1(t) = \frac{5}{6}B^1_{[2,2,1,1][2,2,2,2]}(s,t) + \frac{1}{2}B^k_{[2,1,1,2][2,2,2,2]}(s,t)$$

$$B_2(t) = \frac{1}{2}B^k_{[2,1,1,2][2,2,2,2]}(s,t) + \frac{5}{6}B^2_{[1,1,2,2][2,2,2,2]}(s,t)$$

$$B_3(t) = \frac{1}{6}B^2_{[1,1,2,2][2,2,2,2]}(s,t) + B^3_{[1,2,2,*][2,2,2,2]}(s,t).$$



(a) T-Spline control grid before refinement. Only certain control points are labeled for use in this example. The control point $\mathbf{P_k}$ is not yet inserted in the grid. All intervals in the grid have a value of 2 and $\mathbf{P_k}$ lies halfway (parametrically) down its associated edge.



(b) Blending functions decoupled from a T-spline control grid. The local s knot intervals for each blending function are: B_0:$[*, 2, 2, 2]$, B_1:$[2, 2, 2, 2]$, B_2:$[2, 2, 2, 2]$, and B_3:$[2, 2, 2, *]$. The t knot intervals are $[2, 2, 2, 2]$ for each blending function and are not affected by the insertion of $\mathbf{P_k}$.

Figure 4.4: A T-spline control grid before refinement at $\mathbf{P_k}$.

Here, the refined B-spline basis functions for control point $\mathbf{P_i}$ are denoted $B^i_{[a,b,c,d][e,f,g,h]}$, where $[a,b,c,d]$ are the knot intervals for the s parameter and $[e,f,g,h]$ are the knot intervals for the t parameter[2].

Since all blending functions now agree with the T-grid, the process terminates. The resulting T-spline control point geometries are given by

$$\begin{aligned}
\tilde{\mathbf{P}}_0 &= \mathbf{P_0} \\
\tilde{\mathbf{P}}_1 &= \tfrac{1}{6}\mathbf{P_0} + \tfrac{5}{6}\mathbf{P_1} \\
\mathbf{P}_k &= \tfrac{1}{2}\mathbf{P_1} + \tfrac{1}{2}\mathbf{P_2} \\
\tilde{\mathbf{P}}_2 &= \tfrac{5}{6}\mathbf{P_2} + \tfrac{1}{6}\mathbf{P_3} \\
\tilde{\mathbf{P}}_3 &= \mathbf{P_3},
\end{aligned} \tag{4.1}$$

as illustrated in Figure 4.5.

We now refine the mesh in Figure 4.5 one step further, by inserting a control point at $\mathbf{P_{k+1}}$ as indicated in Figure 4.6. Now, the decoupled blending functions associated with points $\mathbf{P_1}$, $\mathbf{P_4}$, $\mathbf{P_5}$ and $\mathbf{P_6}$ disagree with the T-grid. So, we refine those blending functions:

$$\begin{aligned}
B_4(t) &= B^4_{[2,2,2,2][*,2,2,1]}(s,t) + \tfrac{1}{6}B^1_{[2,2,2,2][2,2,1,1]}(s,t) \\
B_1(t) &= \tfrac{5}{6}B^1_{[2,2,1,1][2,2,1,1]}(s,t) + \tfrac{1}{2}B^{k+1}_{[2,2,1,1][2,1,1,2]}(s,t) \\
B_5(t) &= \tfrac{1}{2}B^{k+1}_{[2,2,2,2][2,1,1,2]}(s,t) + \tfrac{5}{6}B^5_{[2,2,2,2][1,1,2,2]}(s,t) \\
B_6(t) &= \tfrac{1}{6}B^5_{[2,2,2,2][1,1,2,2]}(s,t) + B^6_{[2,2,2,2][1,2,2,2]}(s,t).
\end{aligned} \tag{4.2}$$

[2]To handle the arbitrary topology case, each control point is assigned it's own orientation in the mesh, so the knot intervals are given in the point's local parameterization—i.e., in general s and t are local to each control point.

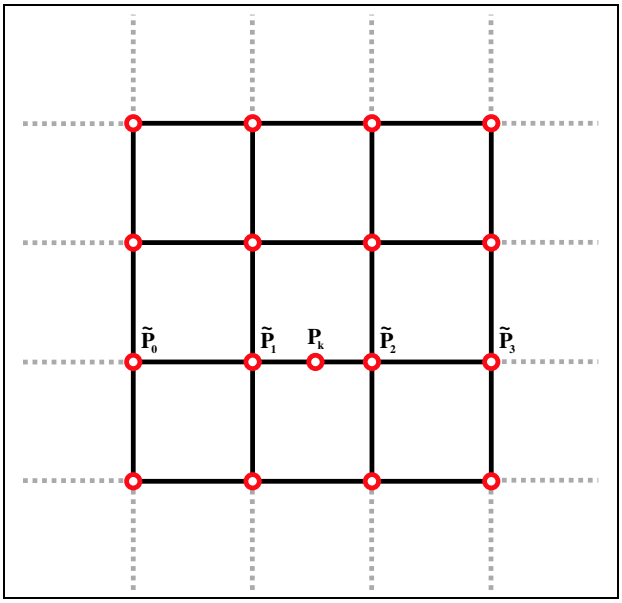


Figure 4.5: T-spline control polygon after refinement. Each of the labeled control points is computed as a linear combination of the original control points as given by (4.1).

Notice that, unlike the previous example, two blending functions still do not agree with the T-grid, specifically, $B^1_{[2,2,2,2][2,2,1,1]}(s,t)$ and $B^{k+1}_{[2,2,1,1][2,1,1,2]}(s,t)$ (originating from $B_4(s,t)$ and $B_1(s,t)$, respectively) do not agree, as emphasized in Figure 4.7.

It is clear that further blending function refinement alone will not result in agreement. For example, refining $B^1_{[2,2,2,2][2,2,1,1]}(s,t)$, we have

$$B^1_{[2,2,2,2][2,2,1,1]}(s,t) \;=\; \frac{5}{6}B^1_{[2,2,1,1][2,2,1,1]}(s,t) + \frac{1}{2}B^k_{[2,1,1,2][2,2,1,1]}(s,t), \qquad (4.3)$$

which may then be substituted into the equation for $B_4(s,t)$ in (4.2). Unfortunately, this additional refinement produces another blending function that does not agree with the mesh: $B^k_{[2,1,1,2][2,2,1,1]}(s,t)$.

The solution to this problem is to alter the mesh by placing a new control point $\mathbf{P_{k+2}}$ into the location demanded by $B^k_{[2,1,1,2][2,2,1,1]}(s,t)$ and $B^{k+1}_{[2,2,1,1][2,1,1,2]}(s,t)$ (as depicted in Figure 4.9), bringing those blending functions into agreement. However, $B^{k+1}_{[2,2,2,2][2,1,1,2]}(s,t)$ and $B^k_k(s,t)$ (the blending function associated with $\mathbf{P_k}$ before

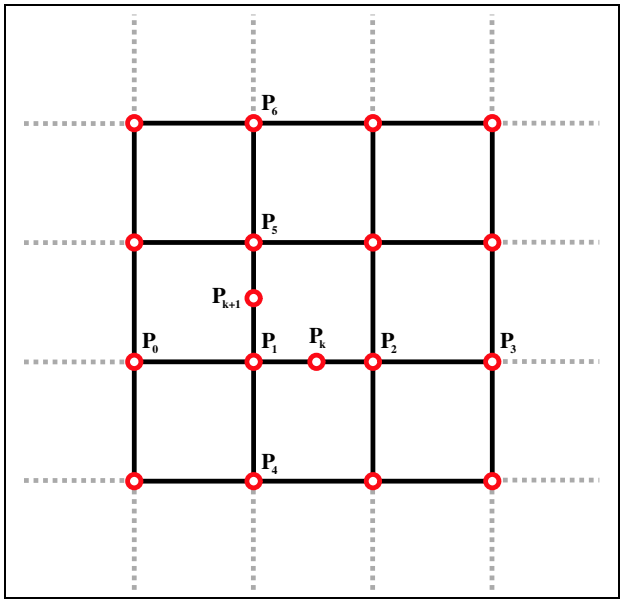
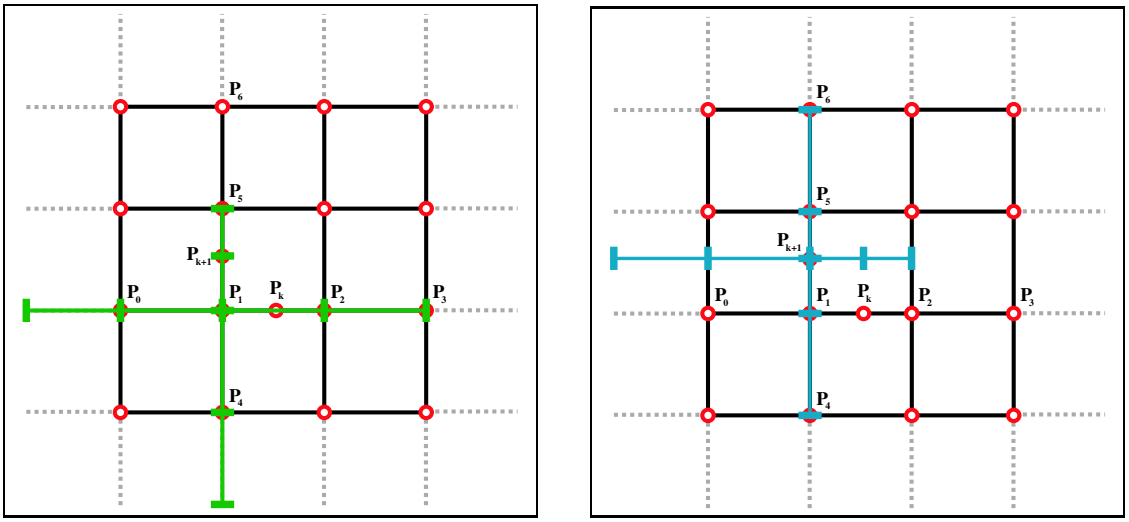


Figure 4.6: T-spline control polygon before refining at $\mathbf{P_{k+1}}$. To match other control points $\tilde{\mathbf{P}}_\mathbf{0}, ..., \tilde{\mathbf{P}}_\mathbf{3}$ have been rewritten with the tilde removed (while retaining their positions from Figure 4.5).



(a) (b)

Figure 4.7: Blending functions out-of-agreement with the T-spline control grid. Above, (a) and (b) show diagrams of $B^1_{[2,2,2,2][2,2,1,1]}(s,t)$ and $B^{k+1}_{[2,2,1,1][2,1,1,2]}(s,t)$, respectively. The blending function diagrams emphasize the locations of each function and its knots.

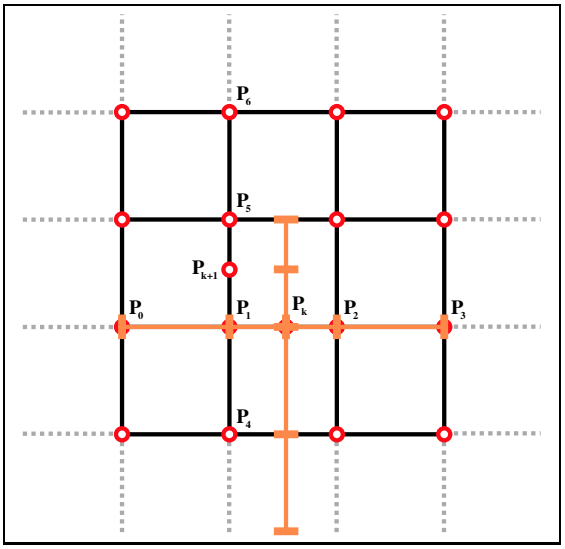


Figure 4.8: Diagram of $B^k_{[2,1,1,2][2,2,1,1]}(s,t)$ whose knots do not completely correspond to the knots implied by the underlying mesh.

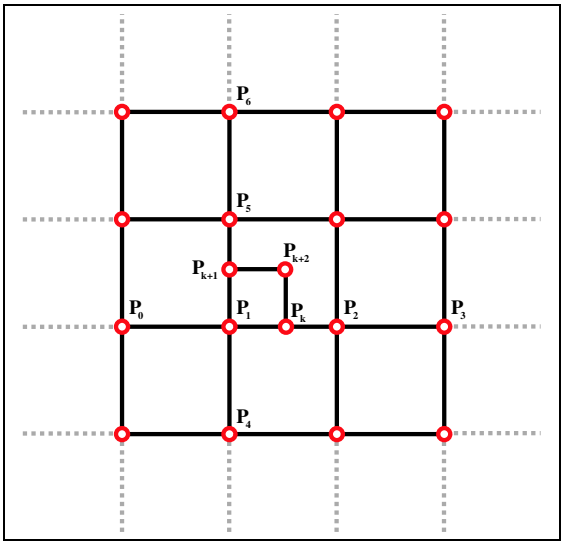


Figure 4.9: A mesh modification prompted by blending functions in disagreement with the mesh. $\mathbf{P_{k+2}}$ and its incident edges were added to the mesh, to match the knots implied by refined blending functions (see Figure 4.7(b) and Figure 4.8).

modifying the mesh) now disagree with the mesh. Refinement of these functions yields

$$\begin{aligned} B_k(s,t) &= \tfrac{5}{6}B^k_{[2,1,1,2][2,2,1,1]}(s,t) + \tfrac{1}{2}B^{k+2}_{[2,1,1,2][2,1,1,2]}(s,t) \\ B^{k+1}_{[2,2,2,2][2,1,1,2]}(s,t) &= \tfrac{5}{6}B^{k+1}_{[2,2,1,1][2,2,1,1]}(s,t) + \tfrac{1}{2}B^{k+2}_{[2,1,1,2][2,1,1,2]}(s,t). \end{aligned} \tag{4.4}$$

Substituting (4.4) and (4.3) into (4.2) and expanding the expressions, we have:

$$\begin{aligned} B_4(s,t) &= B^4_{[2,2,2,2][2,2,2,1]}(s,t) + \frac{5}{36}B^1_{[2,2,1,1][2,2,1,1]}(s,t) + \frac{1}{12}B^k_{[2,1,1,2][2,2,1,1]}(s,t) \\ B_1(s,t) &= \frac{5}{6}B^1_{[2,2,1,1][2,2,1,1]}(s,t) + \frac{1}{2}B^{k+1}_{[2,2,1,1][2,1,1,2]}(s,t) \\ B_5(s,t) &= \frac{5}{12}B^{k+1}_{[2,2,1,1][2,2,1,1]}(s,t) + \frac{1}{4}B^{k+2}_{[2,1,1,2][2,1,1,2]}(s,t) + \frac{5}{6}B^5_{[2,2,2,2][1,1,2,2]}(s,t) \\ B_6(s,t) &= \frac{1}{6}B^5_{[2,2,2,2][1,1,2,2]}(s,t) + B^6_{[2,2,2,2][1,2,2,2]}(s,t) \\ B_k(s,t) &= \frac{5}{6}B^k_{[2,1,1,2][2,2,1,1]}(s,t) + \frac{1}{2}B^{k+2}_{[2,1,1,2][2,1,1,2]}(s,t). \end{aligned}$$

Each of these refined blending functions now agrees with the T-grid, so the resolution process terminates.

Grouping by blending functions as before, we compute the weights and control points for the modified T-spline:

$$\begin{aligned} \tilde{\mathbf{P}}_1 &= \frac{5}{6}\mathbf{P_1} + \frac{5}{36}\mathbf{P_4} \\ \tilde{\mathbf{P}}_4 &= \mathbf{P_4} \\ \tilde{\mathbf{P}}_5 &= \frac{5}{6}\mathbf{P_5} + \frac{1}{6}\mathbf{P_6} \\ \tilde{\mathbf{P}}_6 &= \mathbf{P_6} \\ \tilde{\mathbf{P}}_k &= \frac{5}{6}\mathbf{P_k} + \frac{1}{12}\mathbf{P_4} \end{aligned}$$

$$\mathbf{P}_{k+1} = \frac{5}{12}\mathbf{P_5} + \frac{1}{2}\mathbf{P_1}$$

$$\mathbf{P}_{k+2} = \frac{1}{2}\mathbf{P_k} + \frac{1}{4}\mathbf{P_5}.$$

This example introduced a final element of the T-spline refinement algorithm—a vertex insertion operation to resolve disagreements of blending functions with the T-grid. Adding this step to our process, we produce our final T-spline local refinement algorithm.

Algorithm 4.1 T-Spline Local Refinement

Inputs: A T-spline T with control points $\mathbf{P_i}$, weights w_i and blending functions B_i. A set of topology modifications M to perform on T.

Output: The T-spline T includes the modifications M and is equivalent to the input.

$\tilde{\mathbf{P}} \Leftarrow \{\mathbf{P_i}\}$
$B \Leftarrow \{(w_i, B_i^T)\}$
Apply M to T—add control points and/or edges
while $\{B_j^i\} \neq \{B_j^T\}$ **do**
  for all $(w_j^i, B_j^i) \in B$ **do**
    if B_j^T is more refined than B_j^i **then**
      add a knot from B_i^T to B_j^i using refinement: $w_j^i B_j^i = w_j^i(c_1 \tilde{B}_j^i + c_2 \tilde{B}_k^i)$
      $B \Leftarrow B - (w_j^i, B_j^i)$
      $B \Leftarrow B \bigcup (w_j^i \cdot c_2, \tilde{B}_k^i)$
      $B \Leftarrow B \bigcup (w_j^i \cdot c_1, \tilde{B}_j^i)$
    else if B_i has a knot that is not in B_i^T **then**
      insert a vertex at B_j^i's knot value into the T-grid
    end if
  end for
end while
for all $\mathbf{P_j}$ **do**
  $\mathbf{P_j} \Leftarrow \sum_i w_j^i \tilde{\mathbf{P}}_\mathbf{i}$ where $w_j^i \in (w_j^i, B_j^i) \in B$ and $\tilde{\mathbf{P}}_\mathbf{i} \in \tilde{\mathbf{P}}$
  $w_j \Leftarrow \sum_i w_j^i$ where $w_j^i \in (w_j^i, B_j^i) \in B$
end for

For the presentation of Algorithm 4.1, we adopt an indexing scheme that distinguishes between the new T-spline's control points and those of the original. A superscript indicates the index of the control point with which a variable was associated before refinement and a subscript references the control point at which

the blending function's central knot currently resides[3]—e.g., B_j^i is centered at point $\mathbf{P_j}$ and is a portion of the blending function, B_i^i that was originally centered at $\mathbf{P_i}$. We will overload this notation by referring to the blending functions implied by the current T-grid as B_j^T, which denotes the blending function implied by the current T-grid at point $\mathbf{P_j}$.

We will now discuss each line of the algorithm:

Line 1 Copy the original control points (coordinates and weights) before any modification is done. The new control points are given in terms of these.

Line 2 Decouple the blending functions from the mesh by storing them in a separate data structure. Pair each blending function with its associated weight, which will be scaled during refinement. The scale values of the new blending functions resulting from refinement are multiplied by and stored in the weight portion of the pair. The blending functions are stored as a reference to the control point with which they are associated, and their s and t knot intervals.

Line 3 We restrict topology modifications to splitting edges in the T-grid, enforcing that new edges be created between control points as required for a T-spline[4]. Any new edges may also be split in the modification, as long as the T-spline rules hold. The modifications are the main input to the algorithm and are supplied by the user.

Lines 4–15 The main resolution loop. This loop terminates when all of the refined blending functions match those implied by the T-grid. This means that the

[3]This alternate notation has its weaknesses. Namely, it is ambiguous with regard to the knot intervals of a blending function. For example, a refinement operation on B_j^i results in two scaled blending functions: B_j^i and B_k^i. In the notation, the latter B_j^i is indistinguishable from the former, even though it is more refined. Since this ambiguity only occurs within a single refinement step, we only have need to make a distinction there.

[4]The T-spline definition in [14] requires a T-mesh to follow two rules: **Rule 1** The sum of knot intervals on opposing sides of any face in the T-mesh must be equal. **Rule 2** If a T-junction on one side of a face can be connected to a T-junction on an opposite side of the face (thereby splitting the face into two faces) without violating Rule 1, that edge must be included in the T-mesh.

blending functions residing at a control point in the grid have the same local knot intervals as indicated by the grid at that point.

Lines 5–13 Loop over all of the blending functions, either refining the blending functions to match the mesh or refining the mesh to match the blending functions.

Line 6 Compare the knot intervals implied by the mesh to those in the current blending function—if the mesh is more refined than the current blending function, modify the blending function (lines 7–10).

Line 7 Refine the current blending function (B_j^i), as indicated in Table 2.1—the current blending function is split into two scaled, refined blending functions.

Line 8–10 Remove the current blending function from the set and add the result of refinement into the set. The scale values are stored by multiplying them by the weight of the current blending function.

Lines 11–13 If a knot in the current blending function does not appear in the mesh, place a vertex at the knot location. It might be necessary to create new edges connecting the new vertex to other vertices that align with it parametrically.

Lines 16–19 Compute the new weights (w_j) and control points ($\mathbf{P_j}$) as linear combinations of old control points ($\tilde{\mathbf{P}}_\mathbf{i}$).

Observe that, aside from the initial modification, no new knots are introduced into the T-grid. This follows directly from the central while loop of the algorithm—any blending function or mesh modifications within the loop have to occur at parameter positions already introduced into the T-grid before the loop. This observation leads to the following:

Theorem 4.1 *Let T denote a finite T-spline surface and M denote a finite set of topological modifications to T subject to the T-mesh rules in [14]. If M is applied to T using Algorithm 4.1, then the algorithm will terminate in a finite number of steps*

and the resulting T-spline, T', produces a surface exactly equivalent to the surface of T.

Proof The proof of Theorem 4.1 has two parts:

First, we show that the process terminates in a finite number of steps. Because the central loop of Algorithm 4.1 adds no new knots to the mesh, the most refined mesh extends all knot lines fully across the mesh—terminating in a NURBS equivalent mesh.

Second, we show that $\mathbf{P}'(s,t) \equiv \mathbf{P}(s,t)$. From (2.4), this means that

$$\frac{\sum_i \mathbf{P'_i} w'_i B'_i(s,t)}{\sum_i w'_i B'_i(s,t)} \equiv \frac{\sum_i \mathbf{P_i} w_i B_i(s,t)}{\sum_i w_i B_i(s,t)}.$$

Since our process uses refinement to derive each of the new blending functions, after running Algorithm 4.1 we have:

$$w_i B_i(s,t) = \sum_j w^i_j B^i_j(s,t).$$

The termination condition also implies $\forall j \left[B^i_j(s,t) \equiv B^T_j(s,t)\right]$, where the B^T_j are the blending function of the original T-spline. This equivalence allows us to group terms according to new blending functions and compute new weight and control point positions (as is done in lines 16-19). Since neither the refinement operation nor our algebraic manipulations alter the blending functions, the new surface must be equivalent to the original.

□

We conclude with two observations. First, the denominator of the surface equation does not change during the execution of the local refinement algorithm—this property is important for the next section. Second, although the proof of Theorem 4.1 depends on the full refinement of the T-grid, this refinement is rarely realized

in practice; most refinement operations remain local to the region in which the mesh modifications occur. Practical experience shows that in most cases the local refinement algorithm lives up to its title: it remains *local*.

4.2 T-Spline Simplification using Iterative Refinement

With a procedure for T-spline local refinement, we now develop the iterative refinement method for simplification. Algorithm 4.1 computes the control points of the refined mesh as linear functions of the initial control points:

$$\mathbf{Q_j} = \sum_i w_j^i \tilde{\mathbf{Q}}_\mathbf{i}, \tag{4.5}$$

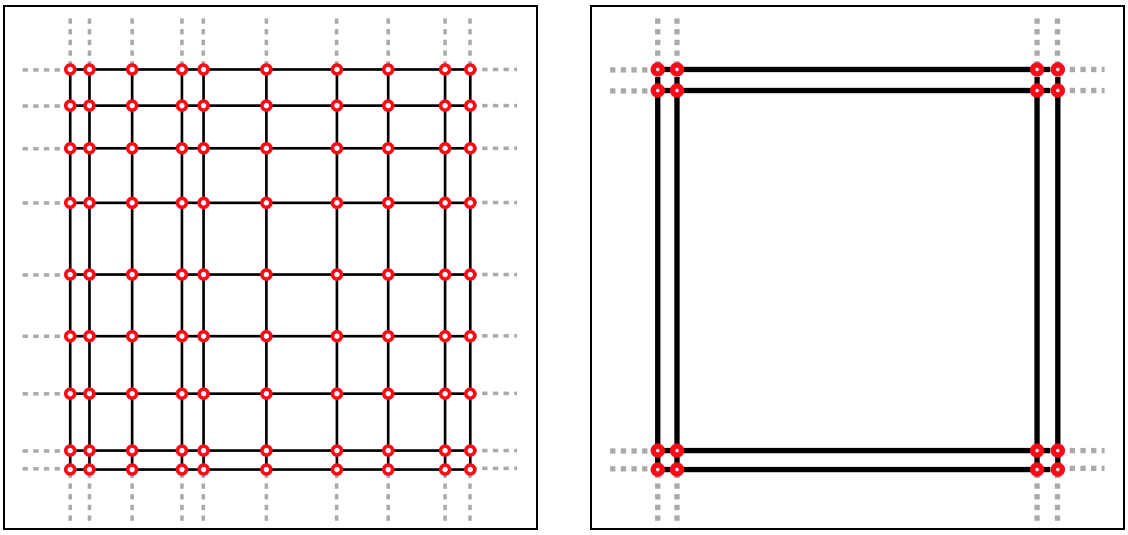
where $\mathbf{Q_j}$ are the control points of the refined mesh, w_j^i are the weights computed during local refinement and $\tilde{\mathbf{Q}}_\mathbf{i}$ are the control points of the original mesh. Converting (4.5) to matrix-vector notation, we produce this system of equations:

$$\mathbf{Q} = \mathbf{M}\tilde{\mathbf{Q}}$$

where the matrix $\mathbf{M}$ computes the refinement of the original T-spline (denoted T_i) to the refined T-spline (denoted T_j).

In the terminology of Section 2.3, $\mathbf{M}$ is a *T-spline refinement matrix* and could be denoted $\mathbf{M}_{j \leftarrow i}$ and may be used in the linear least-squares fitting of T_i to T_j: First, run the local refinement algorithm on T_i to convert it to T_j. Second, use the scale values collected during local refinement to construct $\mathbf{M}_{j \leftarrow i}$. Finally, apply the linear least-squares fitting of Section 2.3 to compute the fit.

We now develop a framework for T-spline simplification. Suppose we have a NURBS mesh as depicted topologically in Figure 4.10(a). From this NURBS,



(a) A complex unsimplified NURBS surface.

(b) The topology of a single Bézier surface.

Figure 4.10: A complex NURBS topology and a single Bézier used to initially approximate it.

we wish to produce a simplified T-spline that approximates the NURBS to some tolerance, ε. We start with an extremely simplified approximation of the surface—like a single Bézier patch, whose mesh topology is shown in Figure 4.10(b). Then, the approximation is *iteratively refined* until the desired tolerance is met. Generalizing this process to simplify any T-spline, we produce Algorithm 4.2.

Algorithm 4.2 T-Spline Simplification : Iterative Refinement

Inputs: A T-spline T with control points, weights and edges. A tolerance value, ε, at which to generate the simplified T-spline.

Output: A simplified T-spline T' that is perturbed from T no more than ε.

```
1: T' ⇐ OverSimplify(T)
2: ε' ⇐ LeastSquaresFit(T',T)
3: while ε < ε' do
4:    for all F_i ∈ T' do
5:       if GetError(F_i) > ε then
6:          Subdivide(F_i)
7:       end if
8:    end for
9:    ε' ⇐ LeastSquaresFit(T', T)
10: end while
```

Details of Algorithm 4.2

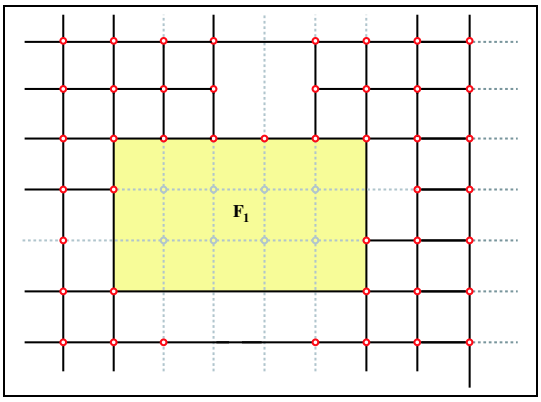
Line 1 Store an over-simplified T-spline as the first approximation of the input T-spline. OverSimplify defines a T-spline over the domain of the input T-spline, but with all interior knots removed. While OverSimplify is straightforward for T-splines with rectangular domains, this procedure is complex for arbitrary topologies. The potential complexity of this procedure is one of the major motivations for developing alternative T-spline simplification methods.

Line 2 LeastSquaresFit performs the least-squares fit of T' to T and returns the error between the two. The least-squares fit is calculated via linear least-squares as described in Section 2.3, where the T-spline refinement matrix is computed from local refinement of T' to T. The error metric is the ℓ^2 norm of the difference between T' and T. For efficiency, a discrete ℓ^2 norm is calculated on the difference between the control points of the approximation and those of the original. This is accomplished by locally refining the approximation to the same space as the original—in building a simplification of an input NURBS surface, for example, this means converting the approximation losslessly to a NURBS and then finding the maximum difference between the NURBS control points of the refined approximation and of the original. The maximum of the control point differences is an upper bound on the maximum difference between the surfaces [27].

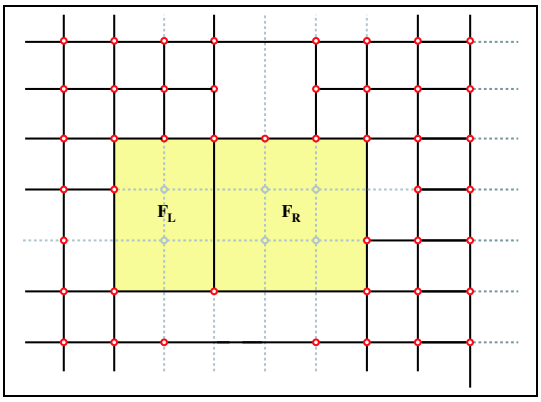
Lines 3–10 Loops until the fit of the approximation, ε', falls below the input tolerance, ε.

Lines 4–8 Iterates over each of the faces of the approximation, evaluating the error within the face.

Lines 5–7 Compares the error within face F_i to the input tolerance ε.



(a) During the refinement step, the error of the surface in the parameter range of F_1 exceeds the tolerance.



(b) F_1 is split along the center-most knot of the longer dimension of its range. In this (the even) case one of the two possible candidates is selected.

Figure 4.11: Splitting an above-tolerance face during iterative refinement.

Line 6 Performs local refinement of F_i. From experiment, the most effective method for refining a face is to split it in half, dividing along the parameter direction with the most knots. For example, the face labeled F_1 in Figure 4.11(a) contains 4 original knot lines in the s direction and 2 original knot lines in the t direction. Therefore, we split the face along one of the central knots of the face's s direction, as in Figure 4.11(b). Figure 4.12 contains some additional examples of this procedure.

Line 9 Repeats the fitting step on the refined approximation in preparation for the next iteration.

4.2.1 Analysis

The results of applying the iterative refinement method are presented in Chapter 6 and an analysis and comparison with other methods are given there. Even without considering these results, we can analyze some shortcomings of this method. For one, the procedure requires the generation of an over-simplified mesh. For meshes with

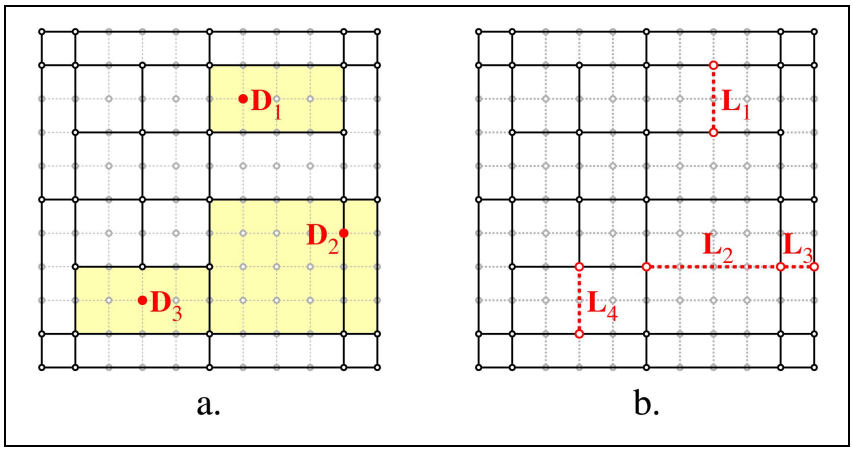


Figure 4.12: Splitting offending faces. The three highlighted faces in (a) have unacceptable tolerances measured at $\mathbf{D_1}$, $\mathbf{D_2}$ and $\mathbf{D_3}$. The faces are split as shown in (b).

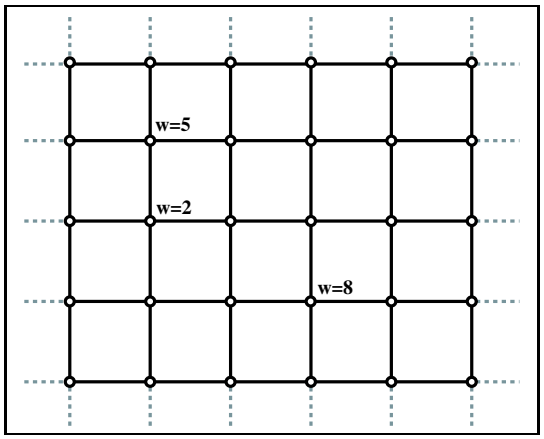


Figure 4.13: A weighted NURBS that is difficult to approximate. Each of the unlabeled control points has a weight of 1, with other weights as labeled. A linear least-squares approximation of this NURBS surface with a single Bézier cannot be performed, because the denominator of the Bézier surface cannot possibly equal that of this surface.

simple topologies this task is easily done, but a mesh with many extraordinary points may be difficult to over-simplify, which limits the applicability of the method.

In addition, the linear least-squares fit requires that the approximation have the same denominator as the more complex original input. This condition makes the initial approximation problem even more difficult. For example, Figure 4.13 diagrams a NURBS surface with a few weights varying from the others. In order to fulfill the condition on the denominator of the surface expression, the weights of the initial approximation need to match that of the NURBS. A single Bézier patch will not fulfill this condition in general. This illustrates that even a topologically simple problem may not be solved correctly with this process.

A final weakness of the iterative refinement method is that it must be applied globally to the mesh. That is, one may not be selective about which control points may or may not be removed. This weakness makes it difficult to direct the simplification—essentially, the only parameter available to the user is the tolerance value for the global fit. A user may want to preserve small perturbations in one section of the model, while allowing greater adjustment in another portion. The primary motivation for the iterative simplification method developed in the next chapter is to address the specific weaknesses of this method.

Chapter 5

T-Spline Simplification using Iterative Simplification

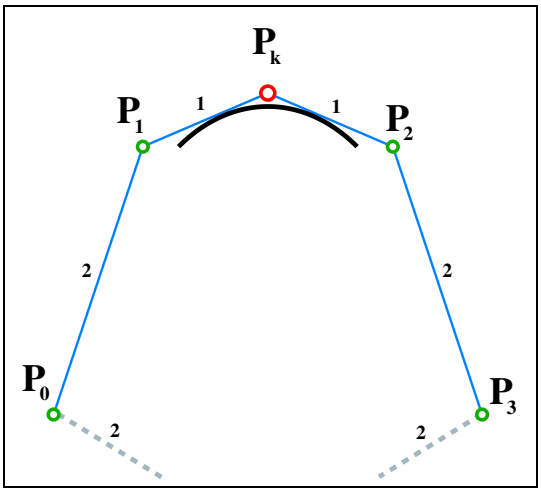
The iterative refinement method of the previous chapter is well-suited for specific T-spline simplification problems, but it lacks the following capabilities:

- Independence from mesh topology. Iterative refinement method requires the generation of an oversimplified model; the new method should not.
- Simplification of rational surfaces.
- Regional/user-specified simplification. The control points that may be removed should be an input into the system.

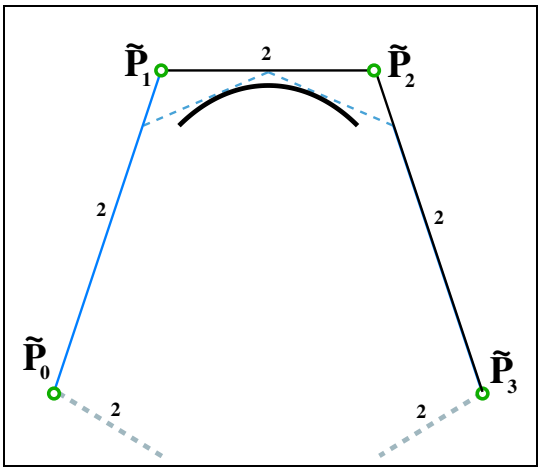
These deficiencies are addressed in the iterative T-spline simplification method, which we now present.

Building on the local refinement algorithm, a modification was presented in [36], permitting the removal of a single specified control point in a T-mesh. [36]'s algorithm allows only exact removal of the selected control point, which is not always possible. In spite of its restrictions, the control point removal algorithm is a useful tool, and we adapt it slightly to produce surface approximating control point removals that are the basis for iterative simplification.

Section 5.1 reviews in detail the control point removal method of [36]. Section 5.2 then adapts control point removal and combines it with least-squares fitting to produce a general framework for iterative simplification and uses that framework to produce a specific simplification method.



(a) Before removing a control point from a NURBS curve. $\mathbf{P_k}$ may be removed from the curve without altering it.



(b) After removing a control point from a NURBS curve. $\mathbf{P_k}$ in Figure 5.1(a) is removed from the curve and the new control points are computed as in (5.2). The curve remains the same over the operation.

5.1 Control Point Removal

Section 4.1.1 shows that a new control point can be added to a NURBS curve without changing the curve. In certain cases, a control point may be removed without changing the curve (the neighboring control points must be moved). We now explain this process.

$\mathbf{P_k}$ may be removed exactly from the curve in Figure 5.1(a) only if the control point configuration can be arrived at by inserting $\mathbf{P_k}$ back into the simplified control polygon. The equations for its insertion are

$$\begin{aligned} \mathbf{P_0} &= \tilde{\mathbf{P}}_\mathbf{0} \\ \mathbf{P_1} &= \tfrac{1}{6}\tilde{\mathbf{P}}_\mathbf{0} + \tfrac{5}{6}\tilde{\mathbf{P}}_\mathbf{1} \\ \mathbf{P_k} &= \tfrac{1}{2}\tilde{\mathbf{P}}_\mathbf{1} + \tfrac{1}{2}\tilde{\mathbf{P}}_\mathbf{2} \\ \mathbf{P_2} &= \tfrac{5}{6}\tilde{\mathbf{P}}_\mathbf{2} + \tfrac{1}{6}\tilde{\mathbf{P}}_\mathbf{3} \\ \mathbf{P_3} &= \tilde{\mathbf{P}}_\mathbf{3}, \end{aligned} \tag{5.1}$$

where the $\tilde{\mathbf{P}}_\mathbf{i}$ are the control points of the curve before the insertion of $\mathbf{P_k}$ and the $\mathbf{P_i}$ are the control points after the insertion. This implies

$$\begin{aligned} \tilde{\mathbf{P}}_\mathbf{0} &= \mathbf{P_0} \\ \tilde{\mathbf{P}}_\mathbf{1} &= 6\mathbf{P_1} - 5\mathbf{P_0} \\ \tilde{\mathbf{P}}_\mathbf{2} &= 6\mathbf{P_2} - 5\mathbf{P_3} \\ \tilde{\mathbf{P}}_\mathbf{3} &= \mathbf{P_3}. \end{aligned} \tag{5.2}$$

Since the removal is exact, the position of $\mathbf{P_k}$ is not needed in (5.2)[1] and the equations compute the modified control polygon without altering the curve (see Figure 5.1(b)). This technique may be generalized to NURBS curves of any degree.

5.1.1 Reverse Blending Function Transformations

We now consider how to undo the effect of a control point insertion in a T-spline. The control point removal method in [36] stems from a modification of the equation for blending function refinement (2.5), which we restate here:

$$B_i(u) = c_1 B_{i_1}(u) + c_2 B_{i_2}(u).$$

This equation is typically used to express a refinement operation—$B_i(u)$ is refined into the sum of two scaled blending functions.

Observe that one of the resulting smaller blending functions *always* retains the center knot of the original, while the other blending function has a different one at its center. For our discussion, we assume that B_{i_1} retains B_i's central knot, while B_{i_2} does not. We now write

$$B_{i_1}(u) = \frac{1}{c_1} B_i(u) - \frac{c_2}{c_1} B_{i_2}(u). \tag{5.3}$$

[1] Incidentally, for this example the exactness condition is: $\mathbf{P_k} = \frac{1}{2}\tilde{\mathbf{P}}_\mathbf{1} + \frac{1}{2}\tilde{\mathbf{P}}_\mathbf{2}$.

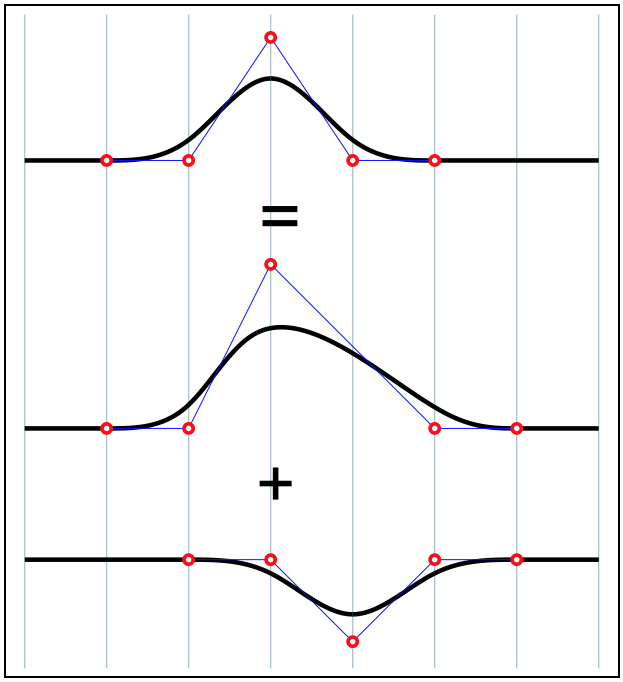


Figure 5.1: A graphical representation of the reverse blending function transformation. The gray vertical lines represent knot values.

Since $c_1, c_2 \in (0, 1]$, we can interpret this as transforming B_{i_1} into the sum of a blending function that is missing one of B_{i_1}'s knots and a *negative* blending function that contains the missing knot and whose central knot differs from B_{i_1}. Figure 5.1 illustrates this interpretation of the operation graphically. [36] calls this operation a *reverse blending function transformation*.

To simplify (5.3) further, we rewrite the fractional coefficients as a_1 and a_2, respectively:

$$B_{i_1}(u) \quad = \quad a_1 B_i(u) + a_2 B_{i_2}(u). \tag{5.4}$$

Using this notation, we modify Table 2.1 to produce Table 5.1. This table identifies the coefficients and intervals for B_i and B_{i_2}, given B_{i_1} has the knot intervals (d_0, d_1, d_2, d_3) with neighboring intervals d_{-1} and d_4 as depicted in Figure 5.2. The contents of Table 5.1 refer to the knot interval and value labels of Figure 5.2:

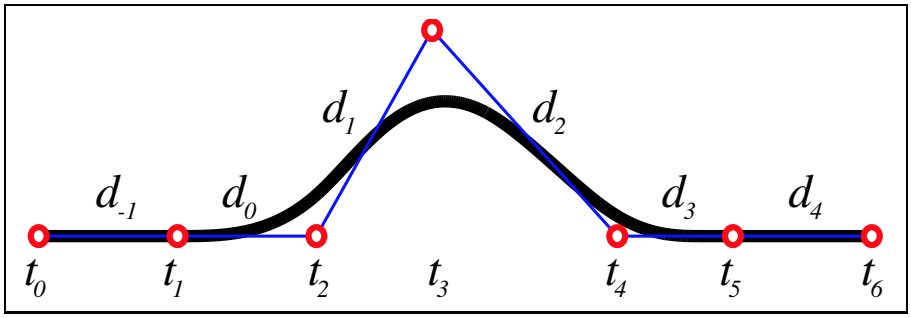


Figure 5.2: A labeled blending function before a reverse transform. where the d_i represent the knot intervals and the t_i (associated with control points) represent the parameter positions of the control points. Depending on the transform, different pairs of neighboring intervals may be joined over as specified in Table 5.1.

Joined Intervals	a_1 *value*	a_2 *value*	*Blending Function Intervals and Central Knots*
d_{-1}, d_0	1	$-\frac{d_{-1}}{d_{-1}+d_0+d_1+d_2}$	$B_i : (d_{-1}+d_0, d_1, d_2, d_3)$ at t_3 $B_{i_2} : (d_{-1}, d_0, d_1, d_2)$ at t_2
d_0, d_1	$\frac{d_0+d_1+d_2+d_3}{d_1+d_2+d_3}$	$-a_1 \cdot \frac{(d_0+d_{-1})}{(d_{-1}+d_0+d_1+d_2)}$	$B_i : (d_{-1}, d_0+d_1, d_2, d_3)$ at t_3 $B_{i_2} : (d_{-1}, d_0, d_1, d_2)$ at t_2
d_2, d_3	$\frac{d_0+d_1+d_2+d_3}{d_0+d_1+d_2}$	$-a_1 \cdot \frac{d_2+d_3}{d_1+d_2+d_3+d_4}$	$B_i : (d_0, d_1, d_2+d_3, d_4)$ at t_3 $B_{i_2} : (d_1, d_2, d_3, d_4)$ at t_4
d_3, d_4	1	$-\frac{d_4}{d_1+d_2+d_3+d_4}$	$B_i : (d_0, d_1, d_2, d_3+d_4)$ at t_3 $B_{i_2} : (d_1, d_2, d_3, d_4)$ at t_4

Table 5.1: Ordinate Values Resulting from a Reverse Blending Function Transformation

Similar to applying blending function refinement to control point insertion, we now apply the reverse blending function transformation to control point removal. We follow the same high-level process formulated in developing control point insertion (see Section 4.1.1): *decouple* the blending functions from the control polygon, *modify* it, and then *resolve* the blending functions to the control polygon. This time, the reverse blending function transformation may be used in the resolution process.

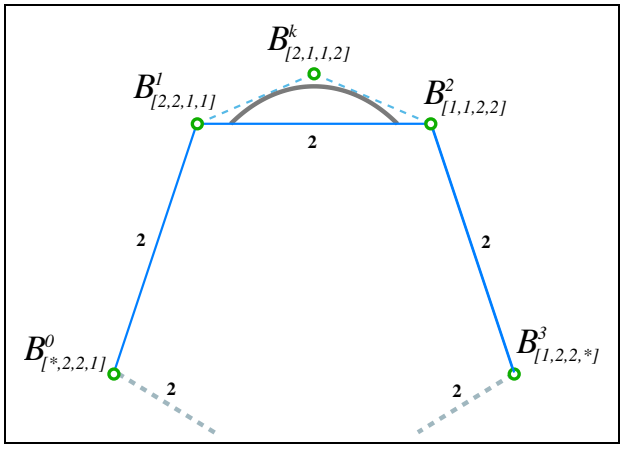


Figure 5.3: Modified control polygon of a curve with decoupled blending functions. Note that $\mathbf{P_k}$ and its blending function are still tracked, even though $\mathbf{P_k}$ is no longer part of the control polygon. Also, each blending function retains the knot intervals from before the modification as specified in their subscripts.

An example of the process will illustrate how this applies to control point removal. Figure 5.1(a) will serve as an example. In the first step, the blending functions decouple from the control polygon, so that they retain their local knot intervals even if the control polygon topology is modified. Then, we remove point $\mathbf{P_k}$, which produces the control polygon in Figure 5.3. Note that Figure 5.3 is annotated with the decoupled blending functions from the first step and that the control point geometries have not yet been modified.

In the resolution step, we examine the blending functions and we note that B_0's local knot vector is more refined than that implied by the control polygon. In addition, the knot location at which B_0 is more refined corresponds to the control point that we removed (the former $\mathbf{P_k}$). Two courses of action are possible:

1. Refinement of the control polygon to match B_0's knot intervals.
2. A reverse blending function transform on B_0 in an attempt to eliminate the offending knot.

Since the offending knot is not wanted (it was expressly removed), the second choice is the only option at this point.

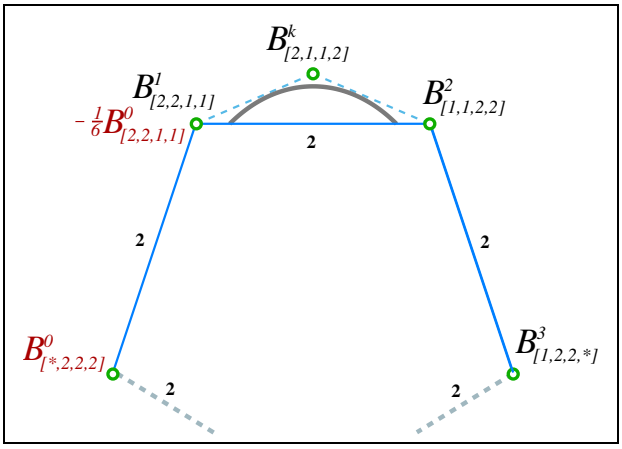


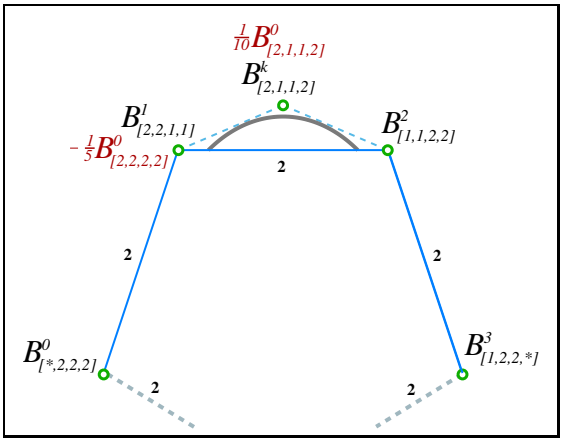
Figure 5.4: Control polygon after a single reverse blending function transform on $B^0_{[*,2,2,1]}$. Note that portions of $B^0_{[*,2,2,1]}$ are shown in red text and now reside at the two left-most control points.

Examining Table 5.1 we see that the last row of the table applies in this case: B_0 is replaced with two other blending functions, one of which does not contain the undesired knot and which is centered on the same knot value/control point as B_0. The other blending function's central knot is elsewhere (specifically, the knot associated with $\mathbf{P_1}$) and has a negative coefficient. The resulting blending function configuration is shown in Figure 5.4.

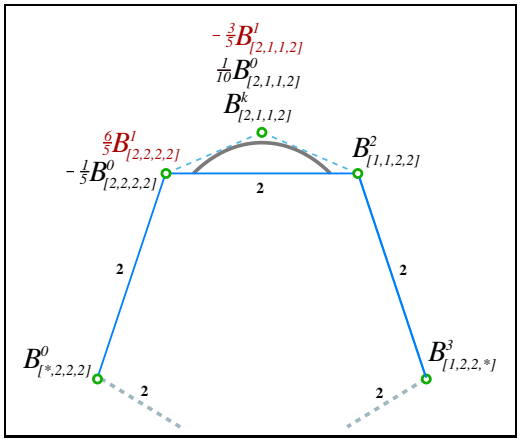
Processing of the remaining blending functions then follows and the remaining steps are shown graphically in Figure 5.5. The end effect of this process produces an interesting result: a set of blending functions with negative coefficients have central knots and knot vectors equivalent to those of B_k and the sum of the "negative" blending functions adds up to eliminate B_k altogether! With B_k eliminated, all of the remaining blending functions correspond exactly to the modified control polygon and produces control point positions[2] that match exactly (5.2). Thus, the process outlined here produces the same result as computed earlier by hand.

Recall that the blending function eliminated through the process (B_k in this example) corresponds to the control point that was eliminated in the mesh. [36] calls

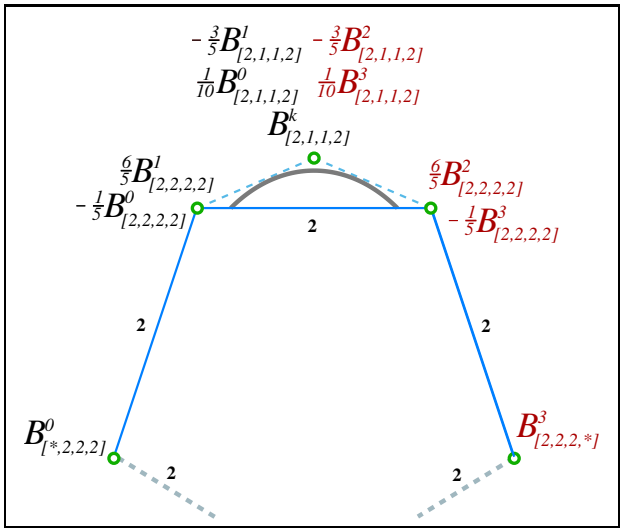
[2]The control point positions are determined using the process described in Section 4.1.1.



(a) A reverse transform is performed on $B^0_{[2,2,1,1]}$ (using row 3 of Table 5.1), producing $B^0_{[2,1,1,2]}$ and $B^0_{[2,2,2,2]}$.



(b) A similar transform is then performed on $B^1_{[2,2,1,1]}$.



(c) By symmetry, additional blending functions are generated. Note that the sum of the coefficients of blending functions centered on $\mathbf{P_k}$ is equal to zero—this eliminates $B^k_[2,1,1,2]$.

Figure 5.5: Completing the control point removal process.

this blending function the *residue* and identifies the goal of T-spline control point removal to eliminate the residue.

Algorithm 5.1 T-Spline Local Simplification

Inputs: A T-spline T with control points $\mathbf{P_i}$, weights w_i and blending functions B_i. A single control point removal operation R that removes $\mathbf{P_k}$ from T.

Output: Returns FALSE if the control point may not be removed. Otherwise, returns TRUE and the T-spline T (now T') includes the removal R and is equivalent to the input.

$\tilde{\mathbf{P}} \Leftarrow \{\mathbf{P_i}\}$
$B \Leftarrow \{(w_i, B_i^T)\}$
Apply R to T—remove $\mathbf{P_k}$. Store residue in $(\mathbf{P_k}, B_k)$ centered at (s_k, t_k).
repeat
  for all $(w_j^i, B_j^i) \in B$ **do**
    if $B_j^i \equiv B_k$ **then**
      $\mathbf{P_k} \Leftarrow \mathbf{P_k} + w_j^i \cdot \mathbf{P_i}$
    else if B_j^i contains but is not centered at (s_k, t_k) **then**
      do reverse transform: $w_j^i B_j^i = w_j^i(a_1 \tilde{B}_j^i + a_2 \tilde{B}_k^i)$
      $B \Leftarrow B - (w_j^i, B_j^i)$
      $B \Leftarrow B \bigcup \left\{(w_j^i \cdot a_2, \tilde{B}_k^i), (w_j^i \cdot a_1, \tilde{B}_j^i)\right\}$
    else if B_j^T is more refined than B_j^i **then**
      add a knot from B_i^T to B_j^i using refinement: $w_j^i B_j^i = w_j^i(c_1 \tilde{B}_j^i + c_2 \tilde{B}_k^i)$
      $B \Leftarrow B - (w_j^i, B_j^i)$
      $B \Leftarrow B \bigcup \left\{(w_j^i \cdot c_2, \tilde{B}_k^i), (w_j^i \cdot c_1, \tilde{B}_j^i)\right\}$
    else if B_i has a knot other than (s_k, t_k) that is not in B_i^T **then**
      insert a vertex at B_j^i's knot value into the T-grid
    end if
  end for
until neither T-grid nor blending functions change in an iteration
if $(\mathbf{P_k}, B_k) \equiv (\mathbf{0}, B_k)$ **then**
  for all $\mathbf{P_j}$ **do**
    $\mathbf{P_j} \Leftarrow \sum_i w_j^i \tilde{\mathbf{P}}_\mathbf{i}$ where $w_j^i \in (w_j^i, B_j^i) \in B$ and $\tilde{\mathbf{P}}_\mathbf{i} \in \tilde{\mathbf{P}}$
    $w_j \Leftarrow \sum_i w_j^i$ where $w_j^i \in (w_j^i, B_j^i) \in B$
  end for
  return TRUE
else
  Undo the modification of T's T-grid.
  return FALSE
end if

5.1.2 T-Spline Control Point Removal

Given the reverse blending function transformation and the goal to eliminate the residual blending function, [36] incorporates them into the local refinement algorithm to produce a local simplification/control point removal algorithm. This algorithm allows the user to specify a single control point to be removed in a T-spline and then either removes the control point successfully or reports that the control point may not be removed. The algorithm is summarized in Algorithm 5.1 and then examined in detail below.

Details of Algorithm 5.1

Algorithm 5.1 is essentially the local refinement algorithm with a few minor enhancements, so this sub-section focuses on the differences between the two methods and refers back to Algorithm 4.1 for the similarities.

Lines 1–2 See Algorithm 4.1, lines 1–2.

Line 3 Modifies the mesh by eliminating $\mathbf{P_k}$. Note that this may modify the mesh by either joining two parallel edges over $\mathbf{P_k}$ or by removing $\mathbf{P_k}$ altogether. The latter only occurs when $\mathbf{P_k}$ does not have a pair of parallel edges passing through it—e.g., $\mathbf{P_k}$ is an L or I junction.

Lines 4–20 The main resolution loop. This loop differs slightly from that of Algorithm 4.1, because this algorithm may not terminate in successful removal—we simply perform as many operations as possible and then see if an acceptable result is produced.

Lines 5–19 Loops over all of the decoupled blending functions, looking for discrepancies between mesh and blending functions.

Lines 6–7 If the current blending function (B_j^i) matches that of the residue, then add B_j^i's scaled geometry ($w_j^i \cdot \mathbf{P_k}$) to the residue's. Note that we perform the

comparison on the blending functions and perform the update on the geometry. Note also that we use *geometry* here to indicate both the Cartesian coordinate and the weight.

Lines 8–11 If the current blending function has the offending knot (the one we are trying to remove), then do a reverse blending function transformation to (hopefully) lead to its removal. Lines 10 and 11 replace the current blending function with the result of the reverse transformation.

Lines 12–15 See Algorithm 4.1, lines 6–10.

Lines 16–18 These lines are similar to lines 11–13 of Algorithm 4.1, but with one significant difference: The conditional (line 16) prevents refinement along the removed knot. This prevents $\mathbf{P_k}$ from being added to the mesh again.

Lines 21–26 After the main loop has terminated, we check the residue: if it has been eliminated (its geometries are zero), the procedure has succeeded, the new control point positions may be computed and success may be returned to the calling program.

Lines 22–25 See Algorithm 4.1, lines 16–19.

Lines 28–20 Since the residue has not been eliminated, the procedure has failed to produce a set of blending functions that correspond to the T-grid (the residue blending function is non-zero and is located where no control point resides). Therefore, we reverse the mesh operations[3] on T and return **false**, stating that $\mathbf{P_k}$ could not be removed as requested.

[36] provides a proof that Algorithm 5.1 will terminate.

[3]This may be done either by recording the mesh operations performed on T in an *undo stack* or by saving an original copy of T.

5.2 Iterative Simplification

Local T-spline simplification applies only to the exact removal case, limiting its usefulness. In order to fulfill the general requirements of T-spline simplification identified at the beginning of this chapter, some modifications are necessary.

In our discussion of T-spline spaces in Section 2.3, we determined that a linear least-squares fit could only be used to fit one T-spline to another if the denominator of the surface polynomials were equal. In the nomenclature of Section 2.3, T-splines defined over the same parameter domain and with equivalent denominators are elements of the same nested sequence of weighted T-spline spaces.

Viewing T-spline local simplification from the perspective of the weighted T-spline space, we observe the effect of applying the method to only the weights of T: Algorithm 5.1 computes $T' \in \mathcal{S}_1^w$ from $T \in \mathcal{S}_2^w$, where $\mathcal{S}_1^w \subset \mathcal{S}_2^w$. Once T' has been computed, it may then be fit to T (or to any other T-spline that is a superset of $\mathcal{S}_1^w$) using linear least-squares. Thus, with only a slight modification we can use Algorithm 5.1 in a T-spline simplification algorithm that fits the desired requirements—we need only replace $(\mathbf{P_k}, B_k)$ in the algorithm with (w_k, B_k), such that only the weights of the control points are processed.

The modified control point removal algorithm (where weights replace geometries) produces a T-spline that belongs to a subspace of the original. Thus, we may fit the geometric positions of the control points in the simplified T-spline to those of the original and compute the error of approximations using the methods in Chapter 4. Essentially, this modification of the T-spline local simplification algorithm permits us to compute approximate simplifications that represent best-fits (in the least-squares sense) to original T-splines.

5.2.1 A Framework for Iterative Simplification

With the tool of lossy T-spline control point removal in hand we now develop a framework that meets our desired requirements for simplifying a T-spline to a given tolerance, ε, on a given set of vertices, $\mathcal{R}$. The framework is summarized in Algorithm 5.2 and explored in detail below. We will use this framework as an outline for the iterative simplification algorithm presented in the Section 5.2.2.

Algorithm 5.2 T-Spline Simplification : Iterative Simplification Framework

Inputs: A T-spline T with control points $\mathbf{P_i}$, weights w_i and blending functions B_i.
A set of candidate control points, $\mathcal{R}$, for removal.
A tolerance value, ε, for the simplification.

Output: A T-spline $\tilde{T}$ simplified to the tolerance ε, from which only $\mathbf{P_i} \in \mathcal{R}$ have been removed.

1: $\tilde{T} \Leftarrow T$
2: $\mathbf{R} \Leftarrow$ GetBestRemoval$(\tilde{T}, \mathcal{R})$
3: **while** GetError$(\mathbf{R}, \tilde{T}, T) < \varepsilon$ **do**
4: PerformRemoval$(\tilde{T}, \mathbf{R})$
5: $\mathcal{R} \Leftarrow \mathcal{R} - \mathbf{R}$
6: LeastSquaresFit$(\tilde{T}, T)$
7: $\mathbf{R} \Leftarrow$ GetBestRemoval$(\tilde{T}, \mathcal{R})$
8: **end while**
9: **return** $\tilde{T}$

Details of Algorithm 5.2

Line 1 Saves a copy of the input T-spline (which will not be modified) into a working copy, $\tilde{T}$, from which control points will be removed. $\tilde{T}$ will always be fit to T.

Line 2 Stores locally the "best" removal candidate. The GetBestRemoval function should select a candidate control point $\mathbf{R} \in \mathcal{R}$. Since the search space of the candidates may potentially be very large, GetBestRemoval can employ any number of heuristics for evaluating candidate control points. Note that since valence four control points may be removed in either parameter direction, $\mathbf{R}$ may need to be coupled with a removal direction (depending on GetBestRemoval).

Lines 3–8 The main loop. The core idea is to remove and fit until no more within-tolerance removals may be performed. This loop ensures that the error produced by removing **R** from $\tilde{T}$ and then fitting $\tilde{T}$ to T is below the user-specified tolerance. For efficiency, the fitting error might be returned by GetBestRemoval along with **R**.

Line 4 Removes **R** and possibly other related control points from $\tilde{T}$ (depending on the removal heuristic).

Line 5 Removes **R** (and any others removed by PerformRemoval) from the set of candidates, $\mathcal{R}$.

Line 6 Fits $\tilde{T}$ to T, by refining $\tilde{T}$ to the space of T or to a common superspace. This fit is performed exactly as described for the iterative refinement algorithm (see Section 4.2). Essentially, a refinement matrix from $\tilde{T}$ to T is generated and then used to perform the least-squares fitting.

Line 7 Collects the next best removal candidate as in line 2.

Line 9 Returns the simplified T-spline, $\tilde{T}$, to the user.

5.2.2 Applying Algorithm 5.2

The abstract framework for iterative simplification involves the two heuristic functions: GetBestRemoval and PerformRemoval. In this section, we develop removal strategies for these functions. The solution developed here does not produce the result with the absolute fewest control points that falls below the tolerance—many models may have several thousand control points, making an exhaustive search of the space intractable. However, the solution produces reasonable results that are comparable to those produced by the iterative refinement method—and, it works on rational surfaces and arbitrary topology meshes.

In considering heuristics for selecting the best candidate for removal from a T-spline, we focus on two specific desirable properties of a simplified mesh. First, removing the candidate should produce a T-spline that fits the original as well as possible—if the selected removal candidate exceeds the tolerance then the main loop of our framework will cause the simplification to terminate. Second, candidates should be removed to promote the expansion of faces in the mesh—large and uniform faces (where possible) give the mesh a structure that is more intuitive to human users.

To meet these goals, we propose the following general principles:

- Each candidate's removal should be evaluated by measuring the fitting error of performing the removal. The removal that produces the best fit should be promoted.
- Since control points may be added in the T-spline local simplification algorithm[4], removing the candidate should decrease the number of control points in the mesh.
- Removal should avoid bias toward a specific parameter—faces open up in the mesh when a removal in the s parameter is followed by one in the t parameter. Removals that repeatedly expand a single edge should be avoided, since this will produce blending functions that have a lot of influence in one parameter, but comparatively little in the other. For example, Figure 5.6 illustrates a T-grid from which several control points along a single parameter direction have been removed. This pattern does not promote the expansion of faces in the mesh and tends to generate meshes that are non-intuitive for modelers.
- Several consecutive removals along a row in the parameter direction opposite the row should be encouraged. This promotes a uniformity in removals that is similar to knot removal methods for NURBS surfaces [25], which necessarily

[4]Algorithm 5.1 only guarantees that the specified point will be removed (if possible), not that other control points will not be created.

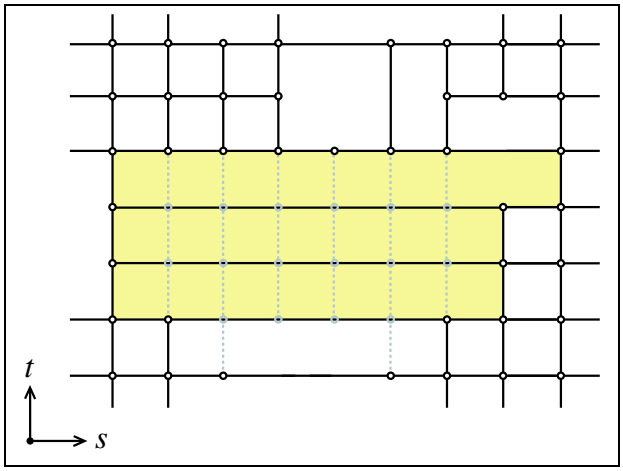


Figure 5.6: Thin faces created removing the grey control points in the s parameter direction. The long edges and faces tend to generate blending functions whose influence is large in one parameter direction, but small in the other.

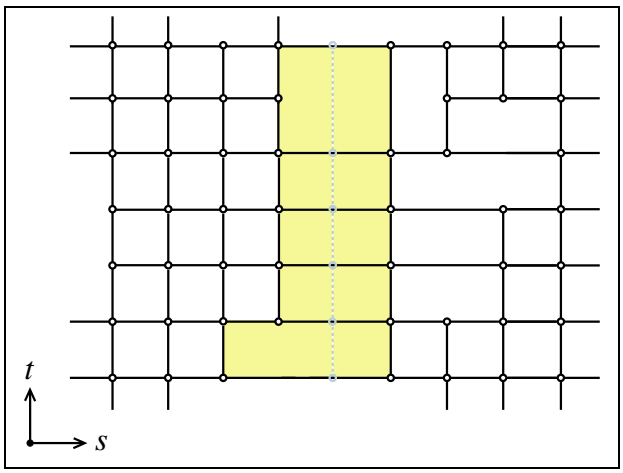


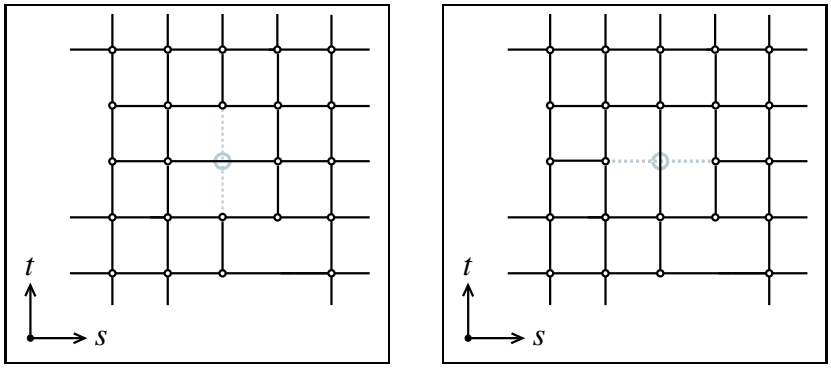
Figure 5.7: Faces created by several removals in the t parameter direction. By removing the gray control points along a line that is perpendicular to the direction of the removal, subsequent removals in the s parameter direction tend to open up square faces.

remove an entire row or column of control points at once. This principle is illustrated in Figure 5.7.

Combining these principles into an iterative simplification solution, we present processes for the GetBestRemoval and PerformRemoval functions in Algorithm 5.2:

GetBestRemoval: This function iterates over the list of candidate vertices, $\mathcal{R}$, and selects the vertex with the lowest perturbation error and which does not increase the number of control points in the mesh. In addition, each time this function is called the parameter direction of the removal for each vertex switches. For valence four vertices, the vertex may be removed in one of two directions s or t as illustrated in Figure 5.8. Each call to GetBestRemoval checks only one of the possible directions in determining perturbation error. On the first iteration the direction is arbitrary, but on each subsequent call to GetBestRemoval, the removal direction toggles to the opposite parameter. Switching between the two parameters balances the removals evenly between the two. If a control point may not be removed in the parameter direction of the current call to GetBestRemoval, its removal is ignored until the next iteration.

PerformRemoval: When the removal of the input vertex, $\mathbf{R}$, is specified, we use $\mathbf{R}$ as an initial position for removing a row of control points. We begin the removal at $\mathbf{R}$ and proceed away from it in the directions opposite of $\mathbf{R}$'s removal. For example, Figure 5.9 shows the removal of $\mathbf{R}$ in the s parameter direction with arrows in the positive and negative t directions from $\mathbf{R}$. Additional removals in s are performed in each of these directions until removals either exceed tolerance or are no longer possible in that direction.



(a) (b)

Figure 5.8: The two removal possibilities for a valence four control point. (a) illustrates the removal of the gray control point (and its edges) in the s parameter direction. (b) illustrates the removal of the same point in the t parameter direction.

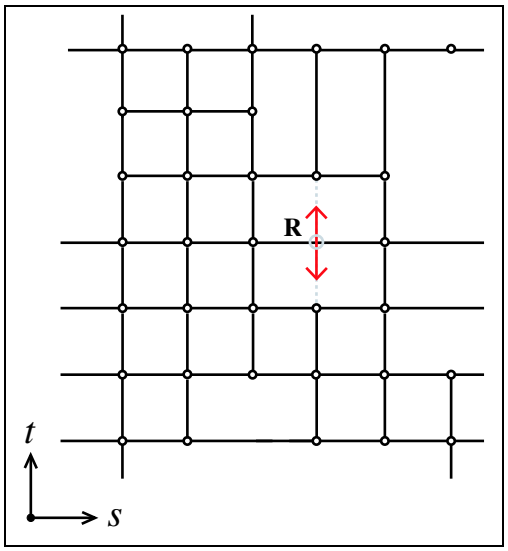


Figure 5.9: The expansion of removals in the parameter direction opposite that of the removal. The removal of **R** is performed in the s direction and then removals are attempted along the t direction as long as they are permitted and do not perturb the surface above tolerance.

An implementation of this iterative simplification solution was used to produce the results presented in Chapter 6, which compares them with the results from the iterative refinement method of Chapter 4.

Chapter 6

Results

This chapter gives the results of applying the methods of this thesis to professionally produced NURBS models. After each presentation, we briefly compare and analyze the two methods, their strengths and weaknesses.

6.1 Iterative Refinement

The results of applying Algorithm 4.2 to several professionally-produced, commercial-quality models are presented in Figures 6.1–6.4. The models in Figures 6.2–6.4 were carefully constructed by professional modelers, who exercised care to avoid the addition of unnecessary control points. In spite of this the iterative refinement algorithm produces approximations with around half of the control points of the initial models with very little perturbation. The model in Figure 6.1 was not as carefully constructed and the simplification method was thus able to eliminate a higher percentage of control points. Each simplification was performed with a tolerance of 1.5%.

One strength of the iterative refinement method, is the uniform structure of the T-spline approximations generated. This structure emerges from the refinement of faces in the process—the face subdivision produces a pattern of increasing refinement in the mesh, which resembles the uniform construction of the original models. This emergent behavior of the algorithm is to generate meshes that home in on the high error regions, while maintaining a meaningful structure in the model's control mesh.

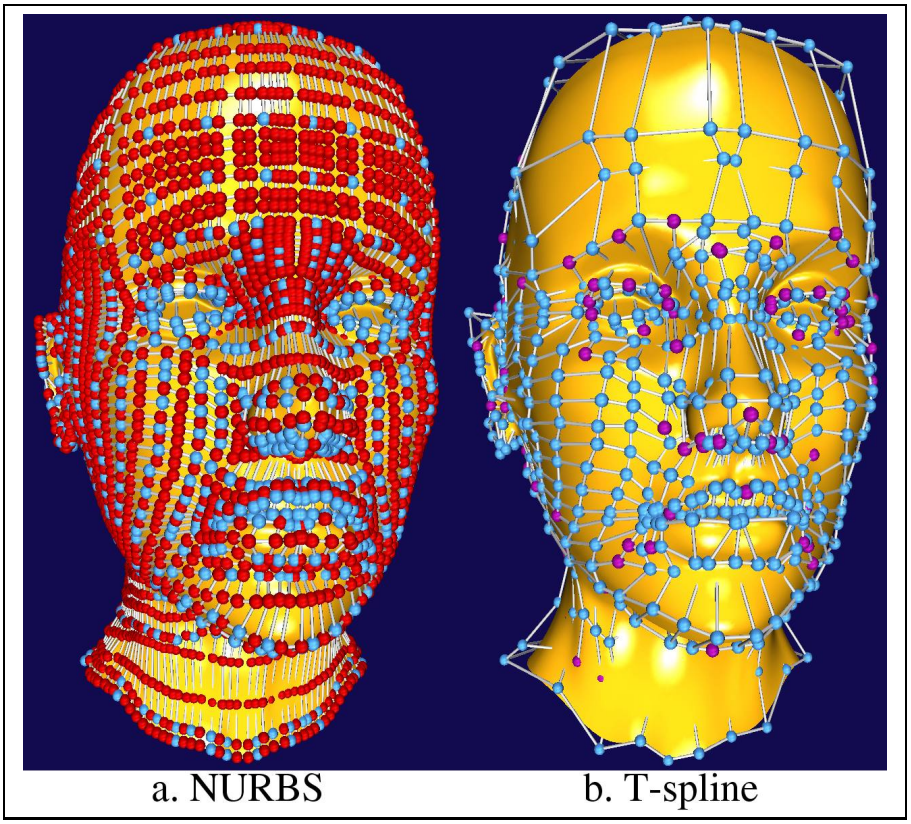


Figure 6.1: Iterative Refinement : model of a human head. The NURBS model (a) has 4,712 control points, which the iterative refinement method reduces to a T-spline with 1,109 control points. To accentuate the control point reduction, the superfluous control points are highlighted in red on the NURBS model. To emphasize that the right model is a T-spline, T-junctions are highlighted in purple.

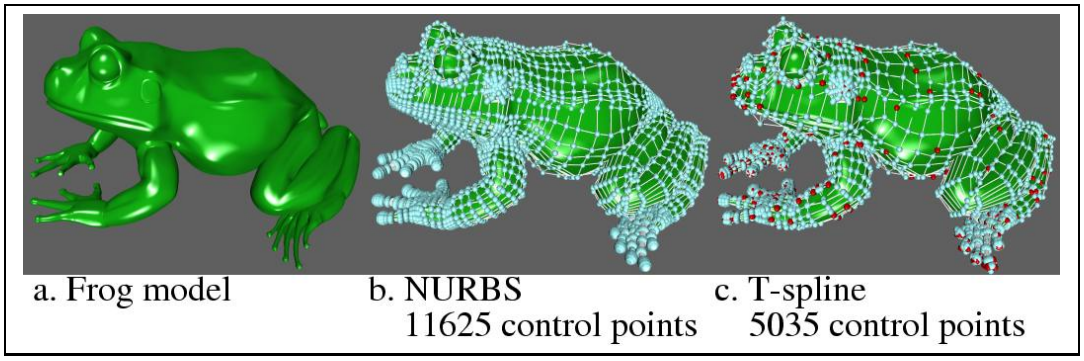
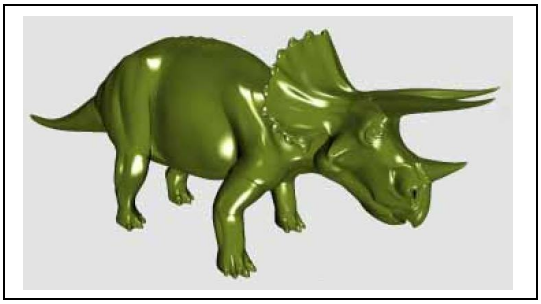
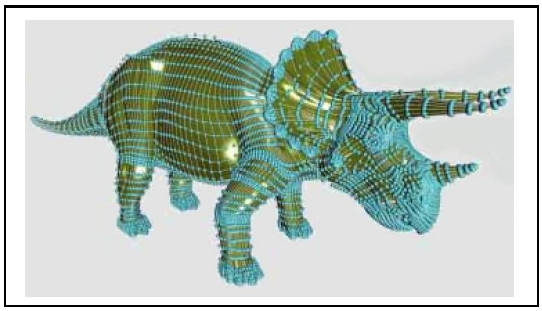


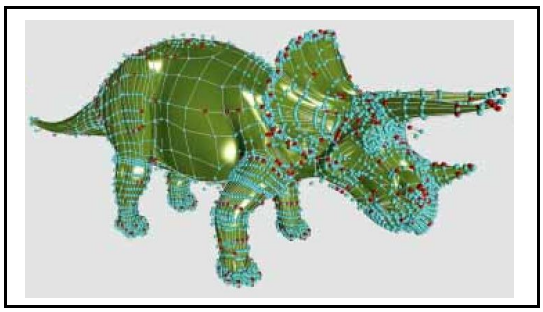
Figure 6.2: Iterative Refinement : Model of a Frog. T-junctions in the T-spline are shown as red control points in this simplification and in the next two models as well. (Courtesy Zygote Media Group)



(a) Triceratops model.



(b) NURBS with 15,588 control points.



(c) Simplified T-spline with 8,432 control points.

Figure 6.3: Iterative Refinement : Model of a Triceratops. (Courtesy Zygote Media Group)

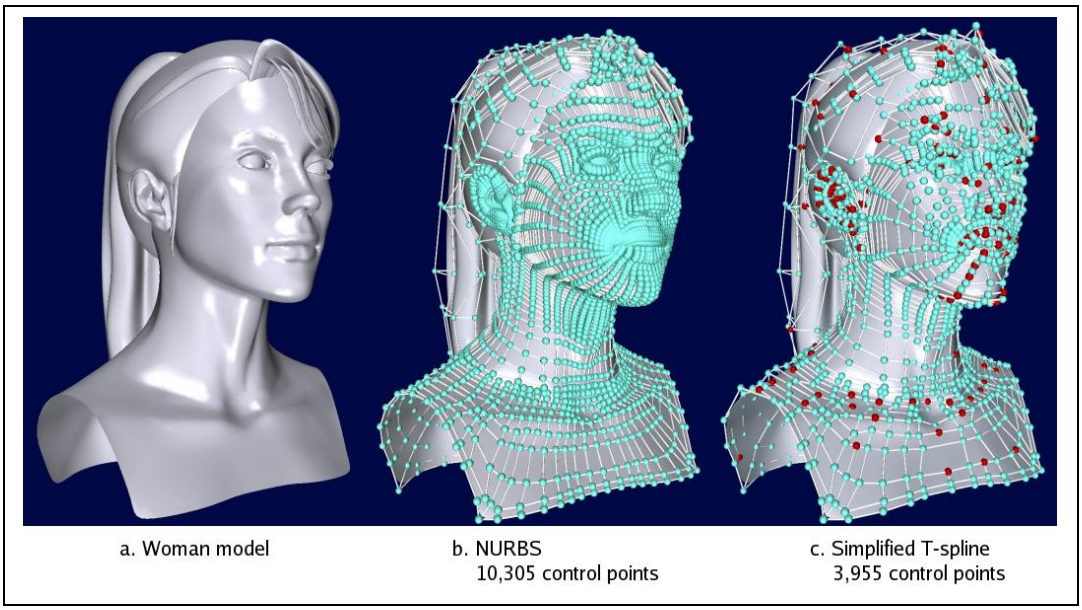


Figure 6.4: Iterative Refinement : Model of a Woman. (Courtesy Zygote Media Group)

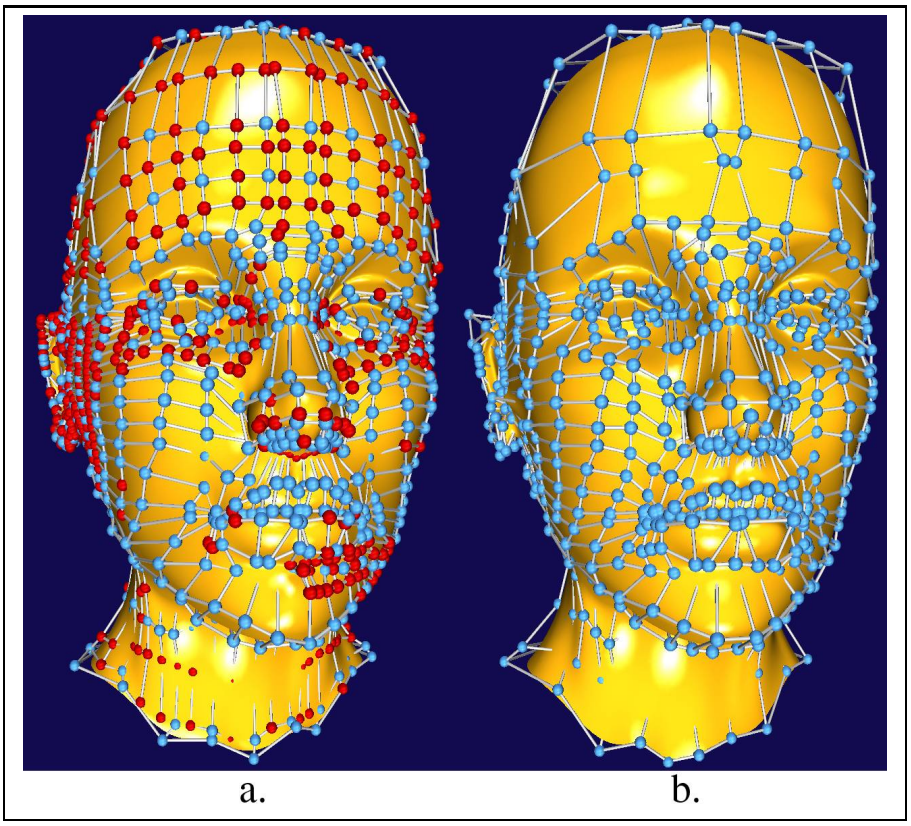
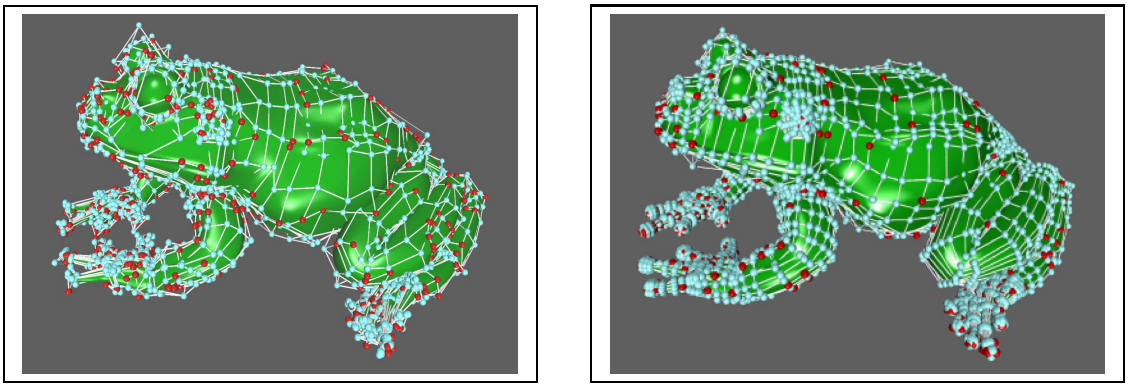


Figure 6.5: Iterative Refinement : Comparison with Wavelet Decomposition. A wavelet decomposition of the original model is "flattened" to a T-spline is shown in (a) and has 1,926 control points. The model generated using the Iterative Refinement method is shown in (b) with 1,109 control points. The additional control points are shown in red on the left, emphasizing the extra control points produced when flattening a wavelet decomposition hierarchy.

The "flattened hierarchy" appearance of the meshes produced by iterative refinement invites a comparison with hierarchical methods. In Figure 6.5 we compare the head model produced by iterative refinement to a B-spline wavelet decomposition represented as a T-spline. To generate the comparison model, we performed B-spline wavelet decomposition, thresholded the small coefficients, and then used T-spline local refinement to construct a T-spline from the remaining non-zero coefficients. The T-spline created using B-spline wavelet decomposition has approximately 74% more control points than that generated from the iterative refinement method using the same tolerance for each method. This difference between the two methods may likely be attributed to the additional control points that are created during the local refinement process.

6.2 Iterative Simplification

Results of the iterative simplification method developed in Chapter 5 are shown in Figures 6.7–6.9. Using the metric of control point count, similar results were attained with this method than with iterative refinement.



(a) Iterative Simplification solution with 3,975 control points.

(b) Iterative Refinement solution with 5,035 control points.

Figure 6.6: Model of a Frog. T-junctions in the T-spline are red.

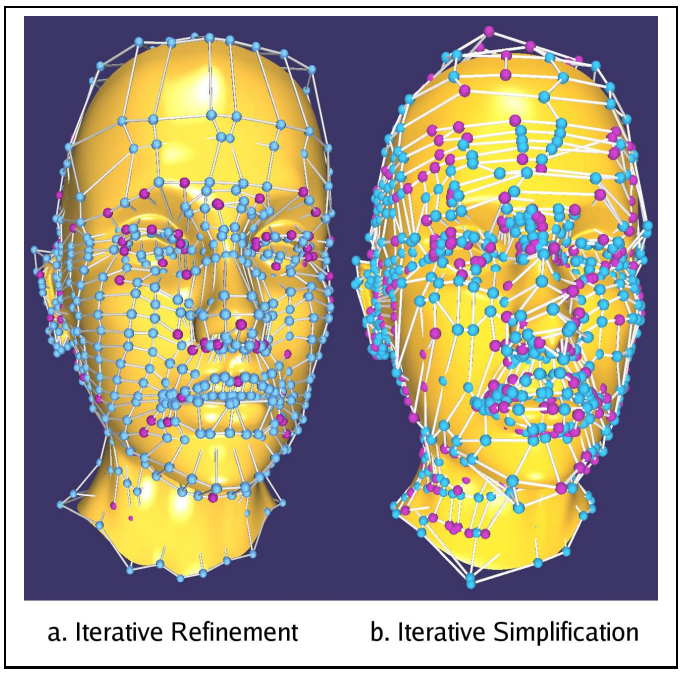


Figure 6.7: Iterative Simplification : model of a human head. Figures (a) and (b) have 1,208 and 1,109 control points, respectively. With T-junctions highlighted in purple, the control polygons illustrate the differences between the two methods. While both identify regions requiring a high-density of control points, iterative refinement homes in on the regions in a very structured manner. See also the topology comparison in Figure 6.10.

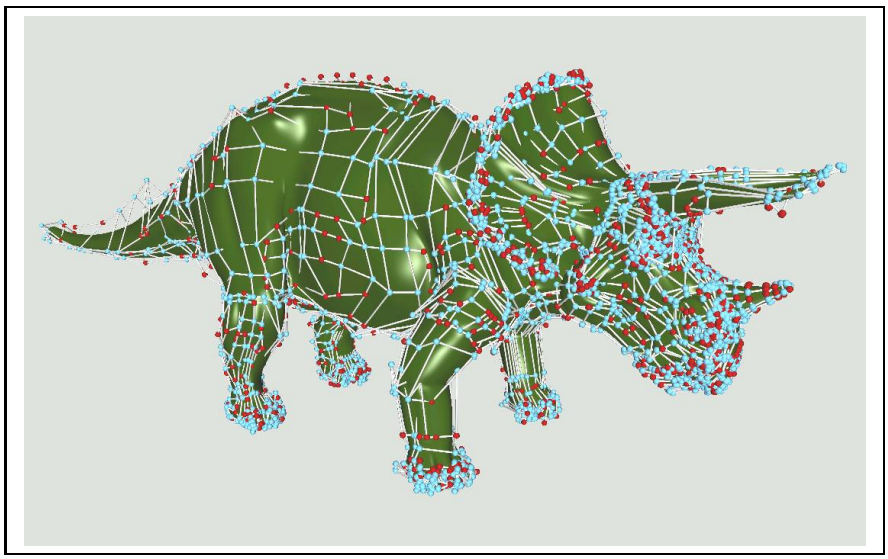


Figure 6.8: Iterative Simplification : Model of a Triceratops. This model has 6,389 control points compared to 8,432 control points in the model simplified to the same tolerance using the Iterative Refinement (see Figure 6.3(c)). T-junctions are red.

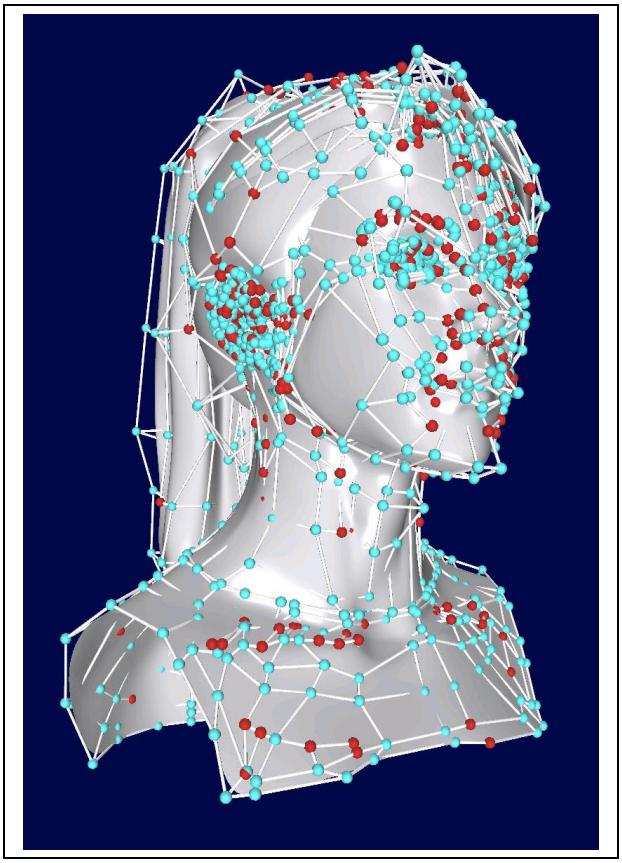


Figure 6.9: Iterative Simplification : Model of a Woman with 3,134 control points (compared to 3,955 using the iterative refinement method).

6.3 Comparison and Analysis

We now compare the quality of our results and the two T-spline simplification methods developed in this thesis. Figure 6.10 compares the simplified topologies of the human head model depicted in 6.1. We note that the iterative refinement method produces a more uniform pattern in its simplification. While both topologies do generate large open regions of the mesh, as evidenced in the figure, in this comparison and for this model, the iterative refinement method exhibits an apparent advantage.

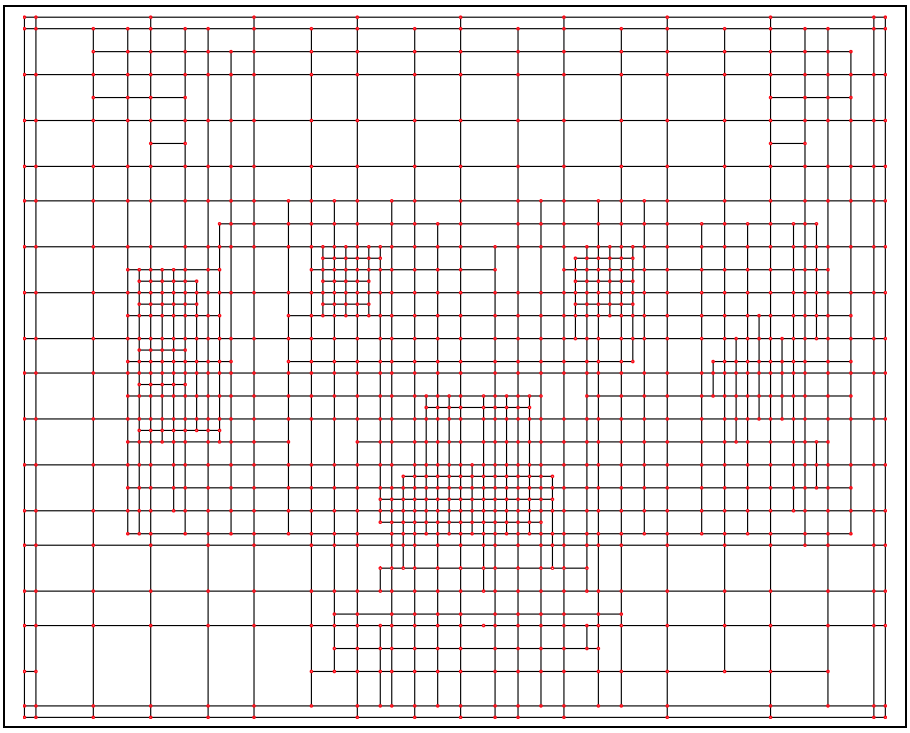
Another dimension for our comparison is to look at the limitations and advantages of the two methods. Some limitations of the iterative refinement method have already been mentioned at the end of Chapter 4. We summarize the comparison in Table 6.1.

Properties	*Iterative Refinement*	*Iterative Simplification*
Global Simplification	Yes	Yes
Selective Simplification (select control points)	No	Yes
Produces Uniform Mesh	Yes	Somewhat
Arbitrary Topology	Non-trivial	Yes
Rational Surfaces	Not generally	Yes
Fitting within a Tolerance	Yes	Yes

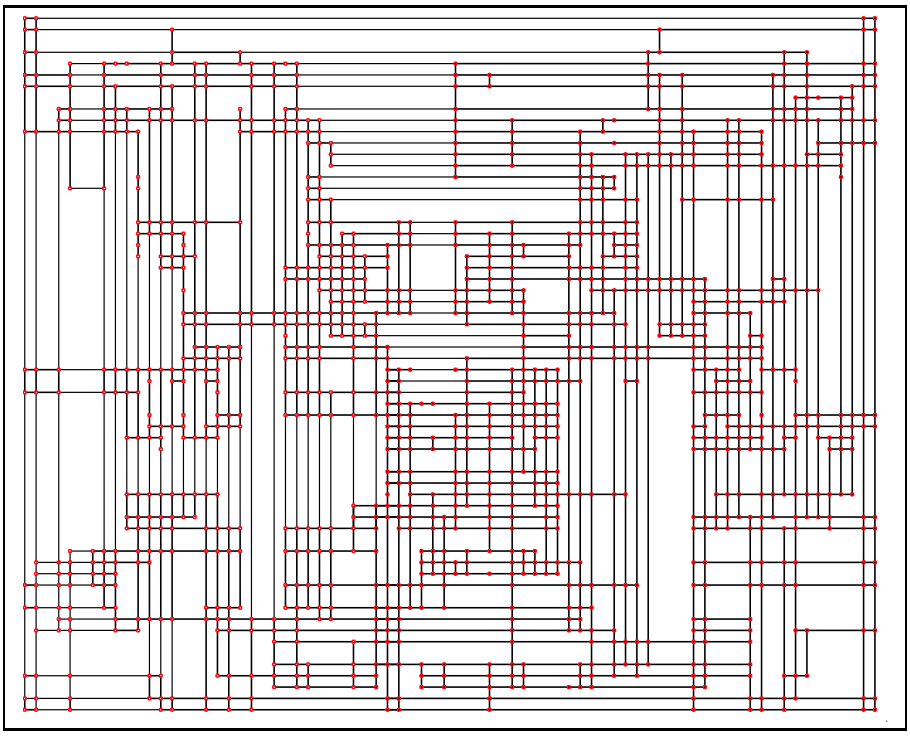
Table 6.1: Comparison of the Properties of the Iterative Refinement and Iterative Simplification Methods

As evident from this table, the iterative simplification method exhibits several clear advantages over iterative refinement. For one, the freedom to operate selectively on arbitrary topology T-splines is of particular note, because it significantly reduces the restrictions placed on iterative refinement. In addition, the selective simplification makes user-guided simplification possible: a user could select a *simplification tool* and use it to click-and-drag over regions of a mesh to simplify interactively. As the user drags the tool over the mesh, tolerance values may be altered for each control point in the iterative simplification algorithm. This user interface would have the net effect of locally and interactively *smoothing* the input surface. This flexibility of user-interface design is not an option with iterative refinement, due to its global approach.

Finally, iterative simplification also provides a supporting framework for additional research. By isolating the control point selection and removal heuristics, additional research can produce improved methods for determining where and when to remove control points. This framework facilitates the development of heuristics that produce results equivalent those of the iterative refinement method, but having all of the significant benefits of iterative simplification.



(a) Topology of the head model simplified using iterative refinement.



(b) Topology of the same model simplified using iterative simplification.

Figure 6.10: Simplification method topology comparison.

Chapter 7

Conclusions and Future Work

This thesis has presented two algorithms for simplifying T-spline surfaces, which are a generalization of the NURBS and Catmull-Clark subdivision surfaces in prevalent use today. The core contribution of the thesis has been the development of methods that organize fundamental operations on T-splines, such as local refinement and control point removal, to produce a simplified solution.

Although the algorithms presented are not optimal with respect to control point count, they do produce control meshes that mimic those produced by human modelers (as evidenced by the large face regions in the meshes) and which exhibit a noticeable correspondence between feature and control point density. The ultimate goal is to find a happy medium between maximal data reduction and useability. The results produced here mark a step closer to achieving the goal.

One important restriction on the methods developed here is the condition necessary for fitting. Future research that incorporates or develops an efficient non-linear least-squares method for fitting could open the door to many new possibilities. It is important to be aware that the linear-fit requirement of the methods of this thesis make necessary the use of T-spline local refinement and control point removal algorithms, which preserve the denominator polynomials of the surface operated on. Were non-linear methods applied to this problem, the topological restrictions on the simplification could be greatly reduced and new methods for computing the simplification could be explored.

Finally, T-splines are relatively new, and this work is the first to delve into this subject. While superficially related to work on NURBS surfaces, the difficulties introduced by the T-spline's signature feature (the T-junction) make the T-spline simplification task much more complicated than simplification of NURBS. In spite of the difficulties, it is hoped that this work may provide a good foundation for future fruitful work in improving the quality of 3D surface models through conversion to and simplification of T-splines.

Bibliography

[1] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, and T. Lyche, "T-spline simplification and local refinement," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 276–283, 2004.

[2] T. W. Sederberg, "Computer aided geometric design." on-line class notes, January 2006. URL: http://cagd.cs.byu.edu/text/ch1.pdf, page 1.

[3] W. Böhm, G. Farin, and J. Kahmann, "A survey of curve and surface methods in CAGD," *Comput. Aided Geom. Des.*, vol. 1, no. 1, pp. 1–60, 1984.

[4] C. de Boor, "On calculating with B-splines," *Journal of Approximation Theory*, vol. 6, pp. 50–62, 1972.

[5] W. Böhm, "Inserting new knots into B-spline curves," *Computer-Aided Design*, vol. 12, pp. 199–201, July 1980.

[6] E. Cohen, T. Lyche, and R. F. Riesenfeld, "Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics," *Computer Graphics and Image Processing*, vol. 14, pp. 87–111, 1980.

[7] L. Ramshaw, "Blossoming: A connect-the-dots approach to splines," Research Report 19, Digital Systems Research Center, Palo Alto, CA, USA, 1987.

[8] C. de Boor, "Bicubic spline interpolaton," *Journal of Mathematics and Physics*, vol. 41, pp. 212–218, 1962.

[9] G. M. Chaikin, "An algorithm for high speed curve generation," *Computer Graphics and Image Processing*, vol. 3, pp. 346–349, 1974.

[10] E. Catmull and J. Clark, "Recursively generated B-spline surfaces on arbitrary topological meshes," *Computer-Aided Design*, vol. 10, no. 6, pp. 350–355, 1978.

[11] D. Doo and M. Sabin, "Behaviour of recursive division surfaces near extraordinary points," *Computer-Aided Design*, vol. 10, no. 6, pp. 356–360, 1978.

[12] T. W. Sederberg, J. Zheng, D. Sewell, and M. Sabin, "Non-uniform recursive subdivision surfaces," in *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 387–394, ACM Press, 1998.

[13] D. R. Forsey and R. H. Bartels, "Hierarchical B-spline refinement," in *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 205–212, ACM Press, 1988.

[14] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, "T-splines and T-NURCCs," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 477–484, 2003.

[15] A. Bakenov, "T-splines : tensor product B-spline surfaces with T-junctions," Master's thesis, Brigham Young University, Provo, UT 84602, USA, 2001. ID: 2855005.

[16] M. Dæhlen and T. Lyche, "Decomposition of splines," in *Mathematical methods in computer aided geometric design II*, pp. 135–160, San Diego, CA, USA: Academic Press Professional, Inc., 1992.

[17] W.-K. Jeong, K. Kähler, and H.-P. Seidel, "Subdivision surface simplification," in *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, (Washington, DC, USA), p. 477, IEEE Computer Society, 2002.

[18] H. Ipson, "T-spline merging," Master's thesis, Brigham Young University, Provo, UT 84602, USA, 2005. ID: 3363860.

[19] G. T. Finnigan, "Arbitrary degree T-splines," Master's thesis, Brigham Young University, Provo, UT 84602, USA, 2007 (in progress).

[20] D. C. Lay, *Linear Algebra and its Applications*. Reading, Mass.: Addison Wesley, 3rd edition ed., 2000.

[21] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1992.

[22] W. Heidrich, R. Bartels, and G. Labahn, "Fitting uncertain data with NURBS," in *Curves and Surfaces in Geometric Design* (A. L. Mehaute, C. Rabut, and L. L. Schumaker, eds.), Vanderbilt University Press, 1997.

[23] W. Ma and J.-P. Kruth, "Mathematical modeling of free-form curves and surfaces from discrete points with NURBS," in *Proceedings of the international conference on Curves and surfaces in geometric design*, (Natick, MA, USA), pp. 319–326, A. K. Peters, Ltd., 1994.

[24] W. Heidrich, "Spline extensions for the maple plot system," Master's thesis, Department of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1, 1995.

[25] T. Lyche and K. Morken, "Knot removal for parametric B-spline curves and surfaces," *Comput. Aided Geom. Des.*, vol. 4, no. 3, pp. 217–230, 1987.

[26] D. Forsey and D. Wong, "Multiresolution surface reconstruction for hierarchical B-splines," in *Graphics Interface*, pp. 57–64, 1998.

[27] W. Tiller, "Knot-removal algorithms for NURBS curves and surfaces.," *Computer-Aided Design*, vol. 24, pp. 445–453, August 1992.

[28] M. Eck and J. Hadenfeld, "Knot removal for B-spline curves," *Computer Aided Geometric Design*, vol. 12, no. 3, pp. 259–282, 1995.

[29] M. Eck and H. Hoppe, "Automatic reconstruction of B-spline surfaces of arbitrary topological type," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 325–334, ACM Press, 1996.

[30] W. Ma and N. Zhao, "Smooth multiple B-spline surface fitting with catmull–clark subdivision surfaces for extraordinary corner patches.," *The Visual Computer*, vol. 18, no. 7, pp. 415–436, 2002.

[31] Y. He, K. Wang, H. Wang, X. Gu, and H. Qin, "Manifold T-spline," in Kim and Shimada [37], pp. 409–422.

[32] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification envelopes," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 119–128, ACM Press, 1996.

[33] H. Hoppe, "View-dependent refinement of progressive meshes," in *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 189–198, ACM Press/Addison-Wesley Publishing Co., 1997.

[34] J. C. Xia and A. Varshney, "Dynamic view-dependent simplification for polygonal models," in *VIS '96: Proceedings of the 7th conference on Visualization '96*, (Los Alamitos, CA, USA), pp. 327–ff., IEEE Computer Society Press, 1996.

[35] T. W. Sederberg, J. Zheng, and X. Song, "Knot intervals and multi-degree splines," *Computer Aided Geometric Design*, vol. 20, no. 7, pp. 455–468, 2003.

[36] Y. Wang and J. Zheng, "Control point removal algorithm for T-spline surfaces.," in Kim and Shimada [37], pp. 385–396.

[37] M.-S. Kim and K. Shimada, eds., *Geometric Modeling and Processing - GMP 2006, 4th International Conference, Pittsburgh, PA, USA, July 26-28, 2006, Proceedings*, vol. 4077 of *Lecture Notes in Computer Science*, Springer, 2006.

[38] J. Stam, "Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values," in *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 395–404, ACM Press, 1998.

Appendix A

Catmull-Clark Local Refinement

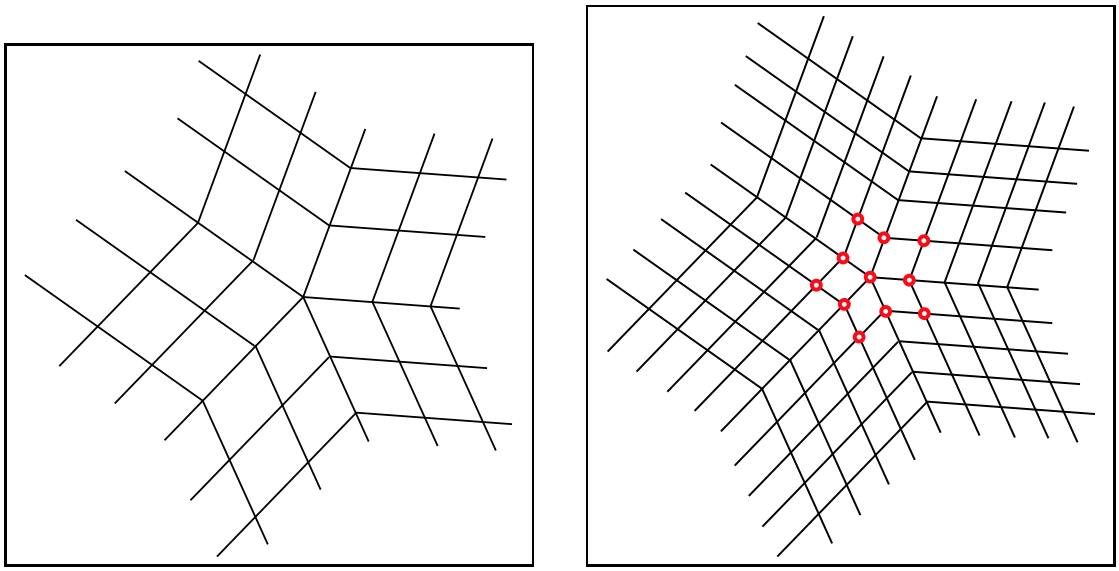
As noted in Chapter 2, some blending functions near extraordinary points are not tensor product B-splines, but are defined using subdivision rules. In this thesis we use only the subdivision rules from [12]. In order to compute the surface at regions near extraordinary points, the subdivision rules must be applied repeatedly until a desired resolution is achieved. Mathematically, the surface consists of an infinite sequence of piecewise polynomial patches that converge on the extraordinary point.

In this appendix, we address specifically the question of local refinement of the blending functions near extraordinary points, which we refer to here as *extraordinary point blending functions* or simply *extraordinary blending functions*. This appendix describes methods for handling the refinement of extraordinary blending functions in the context of T-spline local refinement. The outputs of these methods interface directly with the T-spline local refinement algorithm discussed in Chapter 4.

A.1 Extraordinary Blending Functions

This section reviews the properties of blending functions in a degree-three NURSS mesh, which are the foundation for the extraordinary blending functions in a T-mesh. Instead of reiterating subdivision rules and the details of NURSS meshes here, this section only focuses on the properties of the blending functions of a NURSS mesh as they are applied in a T-mesh. The remainder of this appendix assumes familiarity with NURSSs and their meshes, so a review of [12] is recommended before proceeding.

As noted in Section 2.1.3, each edge in a T-mesh has a parameter interval associated with it—just as in a NURSS mesh. However, a T-mesh imposes an important restriction on its intervals, which NURSSs do not: the sum of the intervals assigned to edges on opposite sides of a face must be equal. So, a T-spline without T-junctions is equivalent to a NURSS, whose intervals on opposite sides of each face are equal. This restriction prevents the unusual infinite sequences of Bézier patches observed in sections 5 and 6 of [12].



(a) Mesh before a global NURSS refinement.

(b) Mesh after a global refinement. Only the blending functions for the highlighted control points are not B-splines.

Figure A.1: Blending functions produced by NURSS subdivision.

The fundamental operation of a NURSS is its subdivision and the NURSS's blending functions are also computed via this operation. As with T-splines and NURBS, we can consider the expression for a NURSS surface as the sum of each control points times its associated blending function:

$$\mathbf{P}(M) \quad = \quad \frac{\sum_{i=0}^{n} \mathbf{P_i} w_i B_i(M)}{\sum_{i=0}^{n} w_i B_i(M)}.$$

Due to the arbitrary topology, each blending function is a function of the entire NURSS mesh, M. To determine the blending function for a control point $\mathbf{P_i}$, we associate with each vertex in the mesh a scalar value. We assign one (1) to $\mathbf{P_i}$'s scalar value and zero (0) to every other vertex in the mesh. Then, we subdivide the mesh substituting the scalar values for the geometries in the subdivision rules. The scalar values resulting from the subdivision correspond to the ordinate values for the blending function.

As observed and illustrated in [38], a single subdivision of a mesh produces regular bicubic patches of C^2 continuity with each refinement. Consequently, a subdivision may produce some combination of tensor product B-splines and extraordinary blending functions as illustrated in Figure A.1.

A.2 Local Extraordinary Point Refinement

Note that scalar values outside of the strict two-neighborhood[1] of the blending function's central vertex are unaffected by any of the subdivisions. Therefore, as long as there are no T-junctions within the two-neighborhood of an extraordinary point, the subdivision operation may be correctly defined in a T-spline.

With our two-neighborhood restriction, we can perform a kind of local refinement of extraordinary blending functions by choosing to subdivide the faces within that neighborhood and ignore any parts of the T-grid outside of it. One problem with this, is that the subdivision rules are defined when the whole mesh is subdivided and not when just a portion of the mesh is refined. This problem stems from the origins of the subdivision rules, which are derived in general from the full refinement of a NURBS surface. In order to derive rules for local refinement of extraordinary points, we must base our derivation on a local subdivision of a T-spline.

Figure A.2 illustrates a local subdivision around a valence four control point in a T-mesh. We can immediately make certain observations about this operation:

- The face points (labeled F_i), the interior edge points (E_0, E_1, E_2, E_3) and the central vertex point (V_c) are all located in the same geometric positions as they have in the full refinement. This becomes more evident if we follow up this local subdivision with a subdivision of the remaining faces—the geometries of these points do not change in the additional subdivisions.
- The exterior edge and vertex points are computed are functions of the vertices in the two-neighborhood of the central vertex, but not of the central vertex itself.
- Thinking ahead to such a subdivision around an extraordinary point, the control points on the border of V_c's two-neighborhood (before refinement) are all tensor product B-splines. So, the contributions of those control points to the positions of the outer edge and vertex points may be determined using B-spline refinement rules.

Drawing from the last observation, we can partition the local subdivision operation into two steps:

[1] We define the one-neighborhood of control point as the set of faces incident to the control point along with each face's incident edges and vertices. The two-neighborhood of a control point is then the union of the one-neighborhoods the vertices in the point's one-neighborhood.

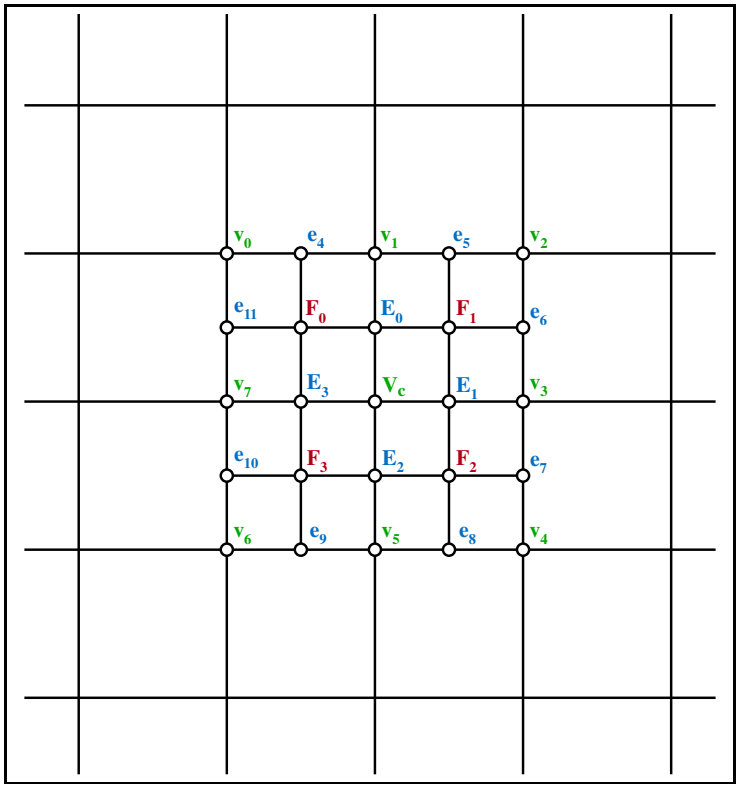


Figure A.2: Local subdivision around a valence-four control point. Control points labeled with upper-case letters are in the same positions as in a global NURSS subdivision of this mesh.

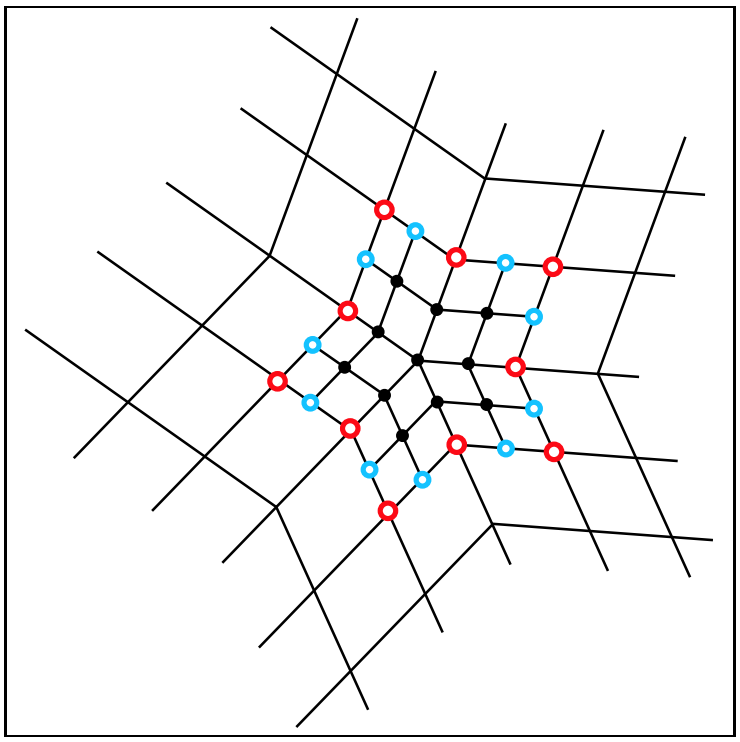


Figure A.3: The red control points need to determine their contributions to the smaller blue ones. Through the NURSS refinement rules, they have already contributed to the small black control points near the extraordinary point.

1. Refine the blending functions associated with the vertices in the one neighborhood of the central vertex (including the central vertex itself).
2. Run the resolution step of the local refinement algorithm on the remaining vertices of the mesh.

This partitioning allows us to compute the refinement of the blending functions near the extraordinary point without requiring them to be tensor product B-splines, while leaving the refinement of the outer blending functions (those associated with points on border of the two-neighborhood of the central vertex) to be resolved by the local refinement algorithm.

Gathering these observations together, we determine that the difficult task of the local subdivision is to determine the contributions of the vertices in the one-neighborhood of the central vertex (excluding the vertex itself) to the outer edge and vertex points as illustrated in Figure A.3. All of the other points may be handled using existing methods: the inner face, edge and central vertex points are computed using NURSS rules and the blending functions of the outer blending functions are

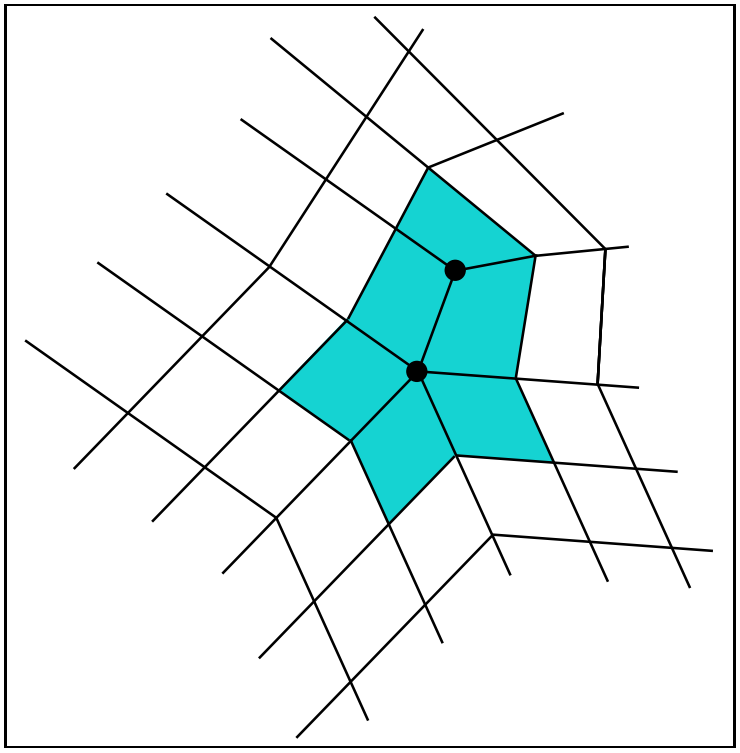


Figure A.4: A cluster of faces near extraordinary points. Any edge or vertex points surrounded by faces in the cluster may be computed using the NURSS subdivision rules.

left to local refinement's resolution step. The rules for computing the refinement of these "one-neighborhood" blending functions is the final topic of this appendix.

Before writing the refinement rules for the points in question, we should observe that only the vertices on the border of the refinement pattern need to have special rules applied to them. So, if multiple extraordinary points lie in close proximity to one another, their incident one-neighborhood faces may be grouped into a cluster of faces (see Figure A.4. Within this face cluster, all face points, inner edge and inner vertex points are computed using the NURSS rules. n-sided faces and all of their incident faces may also be added to such clusters. In some face cluster configurations it is possible to have a faces in the cluster border opposite sides of non-cluster faces as in Figure A.5. In these cases, the non-cluster faces should be added to the cluster.

A.2.1 Refinement Rules for Bordering Vertices

After determining the cluster of faces to refine, all of the vertices on the border of the cluster need to have refinement rules specified for them. The refinement rules for

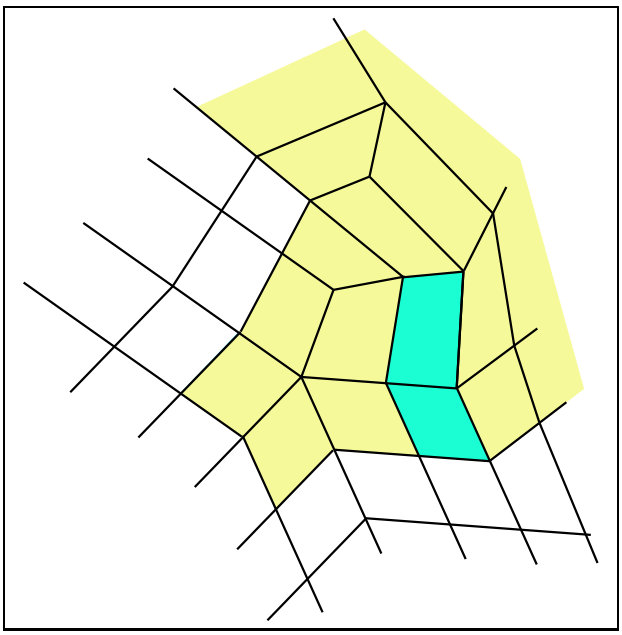


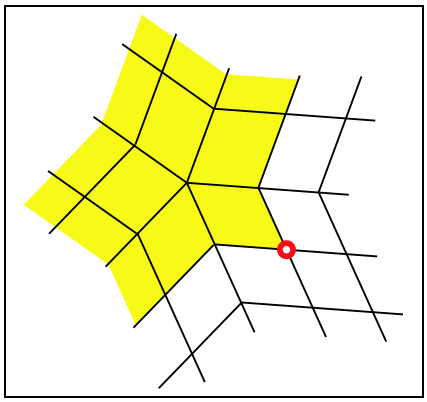
Figure A.5: The faces in yellow belong to the same cluster (they are all incident to extraordinary points). The light blue faces should be refined as well to avoid invalid topological configurations.

such a vertex depend on the refinement pattern near it. Since the bordering vertices are always of valence four, there are four possible refinement patterns that must be accounted four. These patterns are illustrated in Figure A.6.

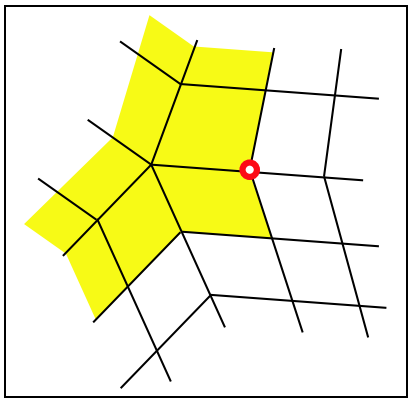
In considering each of these cases, the general strategy is to use the known refinement as a template for each case. This method grows from the observation that each of the blending functions for the bordering vertices are the union of a portion of a B-spline basis function and an infinite series of Bézier patches defined by the subdivision rules. Essentially, our strategy will be to let the subdivision rules take care of the infinite series and treat the remainder as though the blending function were actually a B-spline.

Case 1 : The Outside Corner Vertex

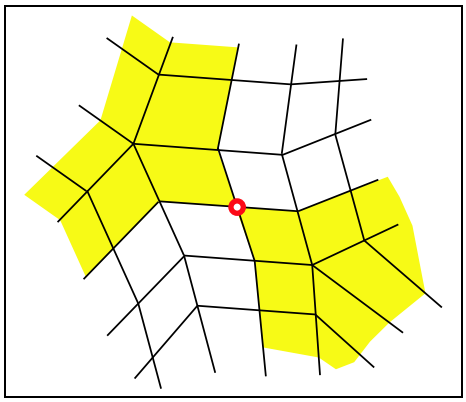
In this case, we observe that three of the four blending functions resulting from subdivision are B-splines. If we consider the refinement of the blending function in the valence-four case, we can think of this as though all of the "non-B-spline information" is transferred completely to the face point. So, we treat the vertex as though it were a B-spline in computing the refined blending functions, throwing away the portion that goes to the face point (since that portion is already accounted for



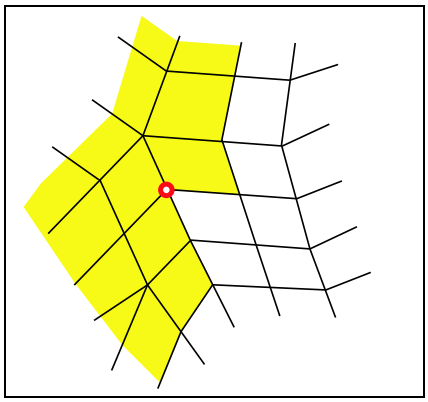
(a) Case 1 : The Outside Corner Vertex



(b) Case 2 : The Cluster Side Vertex



(c) Case 3 : The Shared Outside Corner Vertex



(d) Case 1 : The Inside Corner Vertex

Figure A.6: The four possible refinement patterns of vertices on the border of an extraordinary face cluster. The face clusters are highlighted in yellow.

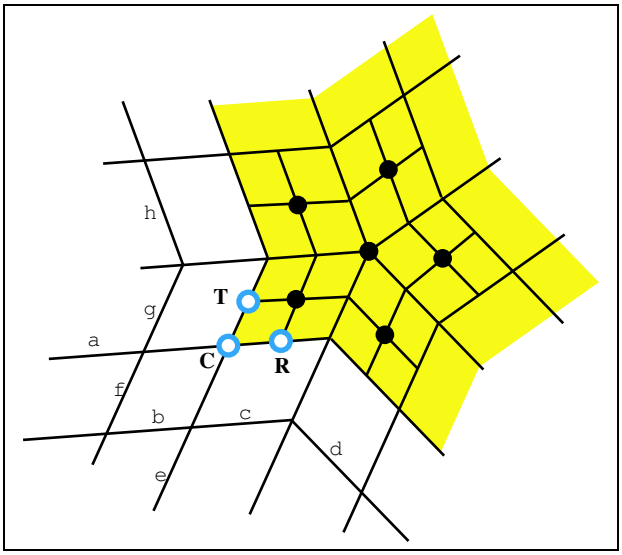


Figure A.7: Case 1 labeled for reference in Table A.1.

in the NURSS rules). This yields three refined B-splines as depicted in Figure A.7. The intervals and scale values for the blending functions corresponding to this figure are given in Table A.1.

Vertex	*Scalar Value*	*Blending Function Intervals*
C	$\left(\frac{a+b+\frac{1}{2}c}{a+b+c}\right)\cdot\left(\frac{e+f+\frac{1}{2}g}{e+f+g}\right)$	$s:(a,b,\frac{1}{2}c,\frac{1}{2}c)$ $t:(e,f,\frac{1}{2}g,\frac{1}{2}g)$
T	$\left(\frac{a+b+\frac{1}{2}c}{a+b+c}\right)\cdot\left(\frac{\frac{1}{2}g+h}{f+g+h}\right)$	$s:(a,b,\frac{1}{2}c,\frac{1}{2}c)$ $t:(f,\frac{1}{2}g,\frac{1}{2}g,h)$
R	$\left(\frac{\frac{1}{2}c+d}{b+c+d}\right)\cdot\left(\frac{e+f+\frac{1}{2}g}{e+f+g}\right)$	$s:(b,\frac{1}{2}c,\frac{1}{2}c,d)$ $t:(e,f,\frac{1}{2}g,\frac{1}{2}g)$

Table A.1: Outside Corner Vertex Refinement

Case 2 : The Cluster Side Vertex

We look at this case, by examining one possible sequence of refinements of this vertex's blending function. Observe that in the valence four case, a single refinement to the inner edge point is sufficient to account for all of the contributions of the side vertex to the face points and to the inner vertex point. What remains after this single refinement is a B-spline associated with the location of the originating vertex.

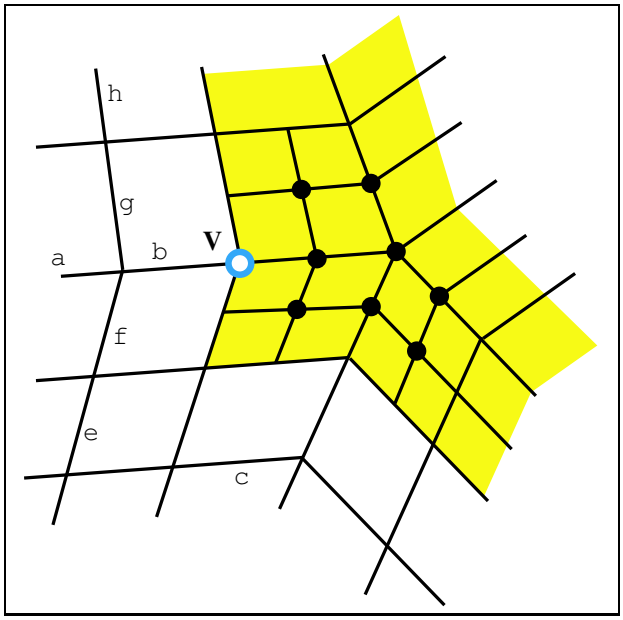


Figure A.8: Case 2 requires only that the blending function's intervals be altered and that it be scaled.

Referring to Figure A.8 the blending function at $\mathbf{V}$ has knot intervals $s = (a, b, \frac{1}{2}c, \frac{1}{2}c)$ and $t = (e, f, g, h)$ and a scalar coefficient of $\left(\frac{a+b+\frac{1}{2}c}{a+b+c}\right)$.

Case 3 : The Shared Outside Corner Vertex

For this case, we consider that the refinement of this vertex's blending function takes two refinements on each side of the blending function. This produces a propagation pattern as depicted in Figure A.9. Once the face points are accounted for, there are three remaining B-spline blending functions. Using Figure A.10 as a reference, we compute blending functions at vertices $\mathbf{C}, \mathbf{L}$, and $\mathbf{R}$ as listed in Table A.2.

Case 4 : The Inside Corner Vertex

This case is by far the most complicated. The reason for the complication, is that the inner face and edge points must contribute to position of the inside corner vertex. So, to handle this case, we partition the work into two parts:

- The refinement of the blending function to itself and the two outside edge points.
- The contributions from the inner edge and face points to the inside corner vertex's position.

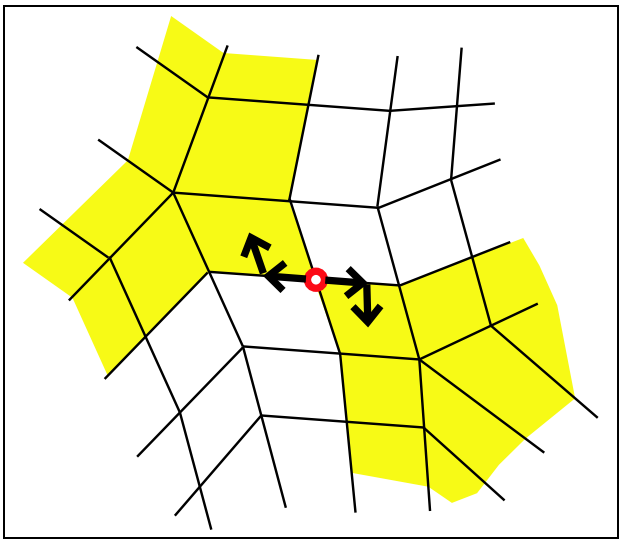


Figure A.9: Case 3 is derived from the series of blending function refinements, that result in blending functions at the visited locations.

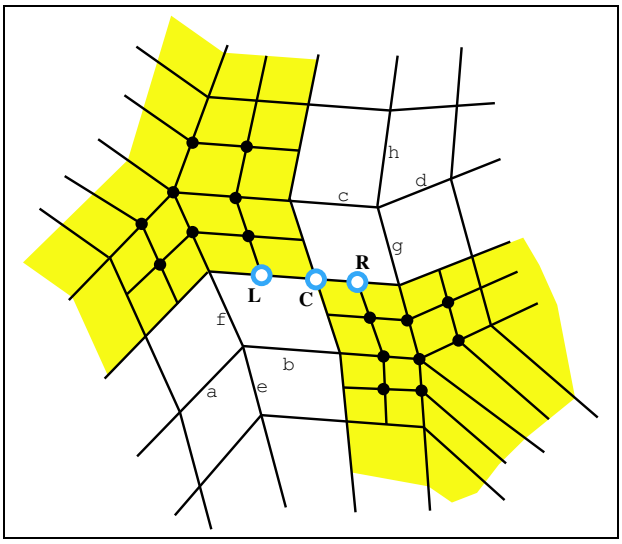


Figure A.10: Case 3 labeled for reference in Table A.2.

Vertex	*Scalar Value*	*Blending Function Intervals*
C	$\left(\frac{\frac{1}{2}b+c+d}{b+c+d}\right)\cdot\left(\frac{\frac{1}{2}c+b}{c+b}\right)+\left(\frac{\frac{1}{2}c}{b+c}\right)\cdot\left(\frac{\frac{1}{2}b+a}{a+b+c}\right)$	$s:(\frac{1}{2}b,\frac{1}{2}b,\frac{1}{2}c,\frac{1}{2}c)$ $t:(e,f,g,h)$
L	$\left(\frac{a+\frac{1}{2}b}{a+b+c}\right)\cdot\left(\frac{\frac{1}{2}f+g+h}{f+g+h}\right)$	$s:(a,\frac{1}{2}b,\frac{1}{2}b,c)$ $t:(\frac{1}{2}f,\frac{1}{2}f,g,h)$
R	$\left(\frac{\frac{1}{2}c+d}{b+c+d}\right)\cdot\left(\frac{e+f+\frac{1}{2}g}{e+f+g}\right)$	$s:(\frac{1}{2}b,\frac{1}{2}c,\frac{1}{2}c,d)$ $t:(e,f,\frac{1}{2}g,\frac{1}{2}g)$

Table A.2: Outside Corner Vertex Refinement

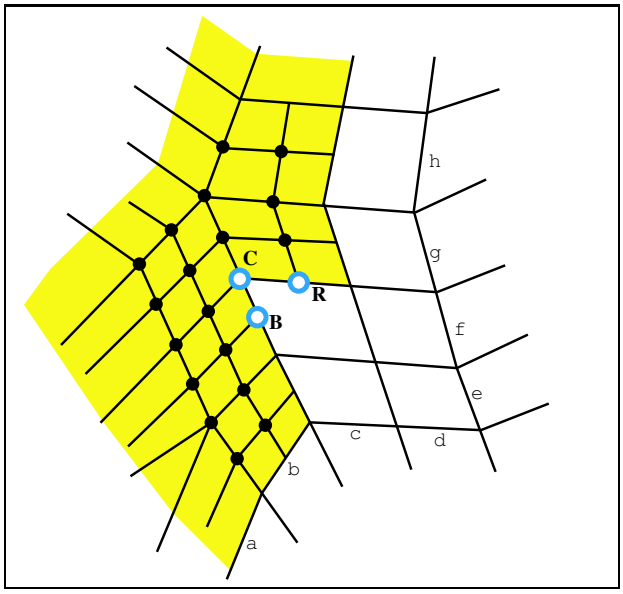


Figure A.11: Case 4 labeled for reference in Table A.3.

The results of each of these parts are B-spline blending functions, but they are computed in very different ways. For the first part, we treat the blending function of the inside corner vertex as though its non-B-spline portions have already been refined away using blending function refinements. This produces blending functions as listed in Table A.3, which refers to Figure A.11.

The computation of the contributions from the face and inner edge points is a bit more complicated. The essential idea is to compute the NURSS vertex refinement around the inside corner vertex, *leaving out* the portions from the non-cluster face and the non-inner edges. This produces an expression in terms of the computed face points and the edge midpoints of the inner edges, which are labeled in Figure A.12.

Vertex	*Scalar Value*	*Blending Function Intervals*
C	$\left(\frac{\frac{1}{2}b+c+d}{b+c+d}\right)\cdot\left(\frac{\frac{1}{2}c+b}{c+b}\right)\cdot\left(\frac{e+f+\frac{1}{2}g}{e+f+g}\right)\cdot\left(\frac{\frac{1}{2}f+g}{f+g}\right)$	$s:(\frac{1}{2}b,\frac{1}{2}b,\frac{1}{2}c,\frac{1}{2}c)$ $t:(\frac{1}{2}f,\frac{1}{2}f,\frac{1}{2}g,\frac{1}{2}g)$
B	$\left(\frac{\frac{1}{2}b+c+d}{b+c+d}\right)\cdot\left(\frac{e+\frac{1}{2}f}{e+f+g}\right)\cdot\left(\frac{b+\frac{1}{2}c}{b+c}\right)$	$s:(\frac{1}{2}b,\frac{1}{2}b,\frac{1}{2}c,\frac{1}{2}c)$ $t:(e,\frac{1}{2}f,\frac{1}{2}f,\frac{1}{2}g)$
R	$\left(\frac{\frac{1}{2}c+d}{b+c+d}\right)\cdot\left(\frac{e+f+\frac{1}{2}g}{e+f+g}\right)$	$s:(\frac{1}{2}b,\frac{1}{2}c,\frac{1}{2}c,d)$ $t:(e,f,\frac{1}{2}g,\frac{1}{2}g)$

Table A.3: Inside Corner Vertex Refinement

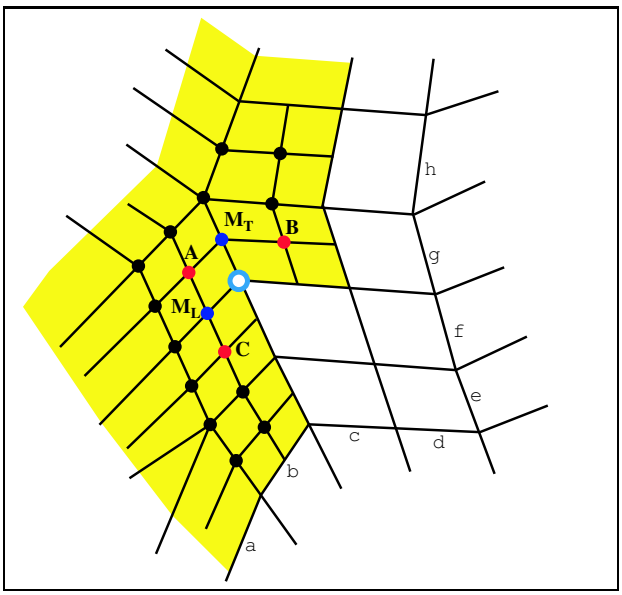


Figure A.12: In case 4, a new blending function at the blue vertex is derived by combining the face and edge mid-points.

The edge midpoints are computed as specified in [12] (Equation 15 in that work). The blending function has knot intervals $s:(\frac{1}{2}b,\frac{1}{2}b,\frac{1}{2}c,\frac{1}{2}c)$ and $t:(\frac{1}{2}f,\frac{1}{2}f,\frac{1}{2}g,\frac{1}{2}g)$. Position of the vertex with this blending function is given by

$$\begin{aligned}\mathbf{V} &= \left(\frac{\frac{1}{2}c}{b+c}\right)\cdot\left(\frac{\frac{1}{2}f}{f+g}\right)\cdot\mathbf{A}\\ &+\left(\frac{\frac{1}{2}b}{b+c}\right)\cdot\left(\frac{\frac{1}{2}f}{f+g}\right)\cdot\mathbf{B}\\ &+\left(\frac{\frac{1}{2}c}{b+c}\right)\cdot\left(\frac{\frac{1}{2}g}{f+g}\right)\cdot\mathbf{C}\end{aligned}$$

$$+\left(\frac{\frac{1}{2}f}{f+g}\right)\cdot\frac{1}{2}\mathbf{M_T}$$
$$+\left(\frac{\frac{1}{2}c}{b+c}\right)\cdot\frac{1}{2}\mathbf{M_L}.$$

A.2.2 Final Note

Each of these expressions may be verified by performing additional refinements and then comparing the result against the full refinement. Note also that many of the B-spline blending functions produced disagree with the T-mesh as they are given in the previous section. However, since they are B-splines, they may be injected into the T-spline local refinement algorithm for further processing. One of the keys to simplifying local subdivision is to determine B-splines from the extraordinary blending functions as soon as possible and then leave the resolution of those blending functions to the T-spline local refinement process. Otherwise, very complex interactions can produce a large number of difficult to manage cases.