2008-07-07

# Arbitrary Degree T-Splines

Gordon Thomas Finnigan
*Brigham Young University - Provo*

# ARBITRARY DEGREE T-SPLINES

by

G Thomas Finnigan

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 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

G Thomas Finnigan

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 | Mark Clement |

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of G Thomas Finnigan 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<br>Chair, Graduate Committee |
| Accepted for the Department | Kent Seamons<br>Graduate Coordinator |
| Accepted for the College | Thomas W. Sederberg<br>Associate Dean, College of Physical and Mathematical Sciences |

ABSTRACT

ARBITRARY DEGREE T-SPLINES

G Thomas Finnigan

Department of Computer Science

Master of Science

T-Splines is a freeform surface type similar to NURBS, that allows partial rows of control points. Up until now, T-Splines have only been formally defined for the degree three case. This paper extends the definition to support all odd, even, and mixed degree T-Spline surfaces, making T-Splines a proper superset of all standard NURBS surfaces.

# ACKNOWLEDGMENTS

I would like to thank my advisor Tom Sederberg for allowing me to be a part of his research team. Working with Dr. Sederberg has been the best part of my BYU education. His persistent vision of improving lives through research and his drive to bring elegant and innovative ideas from research to application have been an inspiration. Working with Dr. Sederberg has been an exciting ride, and has laid a solid foundation for the rest of my career.

I would also like to thank those who helped with the completion of this thesis, including Nick North, Adam Helps, and David Cardon.

Most of all, I'll like to thank my wife for her support, help, love and patience during the completion of this thesis.

# Contents

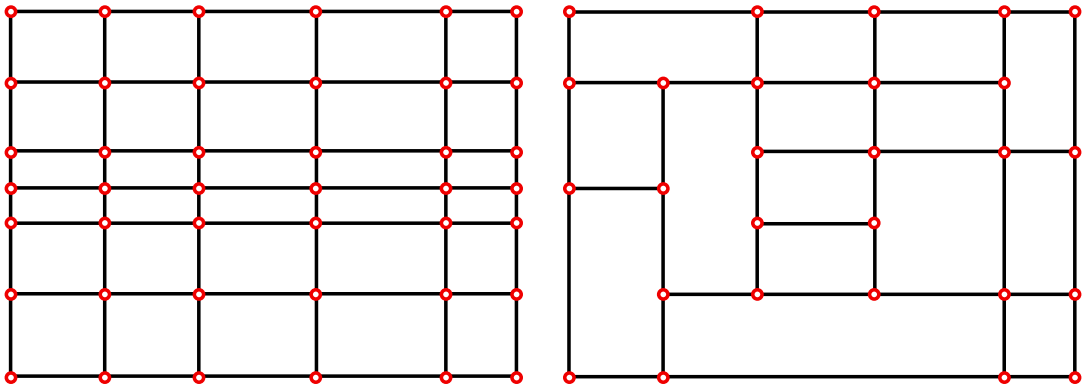
# Chapter 1

## Introduction

NURBS (Non-Uniform Rational B-Splines) are the standard free-form surface type in the CAD industry. However, it is universally recognized that NURBS suffer from serious limitations, such as the fact that NURBS control meshes are limited to rectangular grids, and that NURBS do not support local refinement. T-Splines [1] were invented to address these limitations.

While many other surface types have been proposed to remedy the weaknesses in NURBS, T-Splines hold an advantage in that T-Splines are a proper superset of degree three NURBS. This means that a T-Spline can be exactly converted to a bicubic NURBS and any bicubic NURBS can be exactly represented as a T-Spline. Compatibility with NURBS allows T-Splines to be adopted more easily by the CAD industry, without requiring the entire geometry pipeline to be rewritten.

A NURBS control grid is topologically a rectangular lattice (as illustrated in 1.1a), whereas a T-Spline control grid permits partial rows of control points (as illustrated in 1.1b). A control point that terminates a partial row is called a T-Junction. A fundamental operation in NURBS and T-Splines is refinement: the process of adding additional control points without altering the surface. Refinement of a NURBS control grid requires an entire row of control points to be added, whereas T-Splines can be refined by inserting a single control point, a procedure called local refinement. Local refinement is useful for creating large continuous surfaces with expressive shape control in high-detail areas, and automatic smoothness in lower



(a) A NURBS control grid (b) A T-Spline control grid

Figure 1.1: A NURBS surface requires a grid topology for its control points. A T-Spline may also include T-Junctions, where rows or columns of control points terminate mid-surface.

detail areas. Unlike NURBS, T-Splines can do this without resorting to modeling in small discontinuous patches or burdening the designer with many superfluous and unwieldy control points. T-Splines thus make it easier for artists to define shapes that have varying levels of detail. They also provide elegant solutions to various geometric algorithms such as surface merging, approximation, fitting, and intersection. [1, 2, 3, 4]

Unfortunately, the initial formulation of T-Splines [1] only defines degree three T-Splines, whereas NURBS are defined for arbitrary degree. While most NURBS models in common use are degree three, the use of other degrees is sometimes dictated by manufacturing technique, styling preferences or algorithm design. For example, higher degree surfaces are smoother and more expressive but take longer to evaluate, while lower degree surfaces such as cylinders, spheres, and torii, for which degree two suffices, are less expressive but require fewer control points.

The goal of this paper is to generalize T-Splines to arbitrary degree, thereby allowing T-Splines to be compatible with all NURBS surfaces in common use.

## 1.1 Background

The use of tensor product B-Spline surfaces for geometric modeling was first proposed in Riesenfeld's Ph.D. thesis [5], and a thorough treatment of NURBS can be found in numerous textbooks, such as [6], [7], [8].

T-Splines were introduced in a 2003 paper [1] which defines bicubic T-Splines, presents an algorithm for local refinement, and discusses merging and extraordinary points. A second paper [2] improves on the refinement method, and shows applications in shape simplification and approximation. Ipson's Master's Thesis [3] deals with merging multiple T-Spline surfaces together in cases where the parametrizations of the adjoining surfaces don't agree. Several other papers have explored various aspects of T-Splines [9, 10, 11, 12], but they all focus on the bicubic case.

The concept of knot intervals are explored in [13], which also introduces the idea of combining multiple degrees in the same curve.

## 1.2 Overview

While the original definition of T-Splines only gives rules for degree three surfaces, those rules can easily be extended to support surfaces of arbitrary odd degree, as shown in Chapter 2. Chapter 3 defines even degree T-Splines, and Chapter 4 discusses mixed degree T-Splines.

Local refinement is the ability to add control points anywhere in the surface, while exactly preserving the shape of the surface. Chapter 5 extends the local refinement algorithm in [2] to handle arbitrary degree T-Splines. Chapter 6 covers a conclusion and a list of related future work.

## 1.3 NURBS Review

This section briefly reviews NURBS theory adequately for the purposes of understanding the contributions of this paper. The presentation of NURBS in this section is slightly unorthodox, in that it first examines the simpler Bézier curves and surfaces, and then discusses the relationship between NURBS and Béziers.

A Bézier curve is a parametric curve defined by $n+1$ control points, $P_0..P_n$. The point on the curve at a parameter value $t$ is

$$\mathbf{P}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \tag{1.1}$$

One important thing to note about this formula is that the degree of the equation (the highest exponent of any term in the simplified form) is related to the number of control points: A degree $n$ Bézier curve has $n+1$ control points. The parameter $t$ is in the range 0..1.

A tensor-product Bézier surface is defined by an $n+1$ by $m+1$ grid of control points. A point on a Bézier surface at a parameter location $s, t$ is

$$\mathbf{P}(s,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} \binom{n}{i} (1-s)^{n-i} s^n \binom{m}{j} (1-t)^{m-i} t^i \mathbf{P}_{i,j} \tag{1.2}$$

Note that we can have a different degree in each direction. When $n \neq m$ we call the surface a mixed degree surface. Similar to curves, the parameters $s$ and $t$ are each in the range 0..1.

The degree of a Bézier curve determines how expressive the curve can be. For example, a degree one Bézier curve has only two control points, and can only represent a line segment. A degree two Bézier curve has three control points, and represents a segment of a parabola. A degree three Bézier curve has four control points, and can represent segments of curves with inflection points.

NURBS are a way of defining several Bézier segments together without redundant control point specification. For example, a degree one NURBS curve can represent a connected, arbitrarily large set of line segments, and a degree two NURBS curve can represent an arbitrarily large set of continuous parabolic segments. Likewise, NURBS surfaces can represent an arbitrarily large grid of Bézier patches.

NURBS are traditionally defined in terms of control points and blending functions. That is, the point on a NURBS surface is:

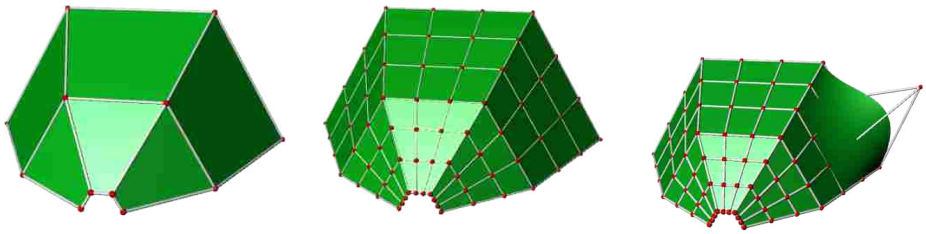
$$\mathbf{P}(s,t) = \sum_{i=0}^{k} N_{i,n}(s,t)\mathbf{P}_i \tag{1.3}$$

where $k$ is the number of control points, and $N_{i,n}$ is the NURBS blending function for $P_i$ of degree $n$.

Previous to this thesis, T-Splines were only defined for the degree three case. While this can be made to work for a large number of situations, there are several advantages to an arbitrary degree definition.

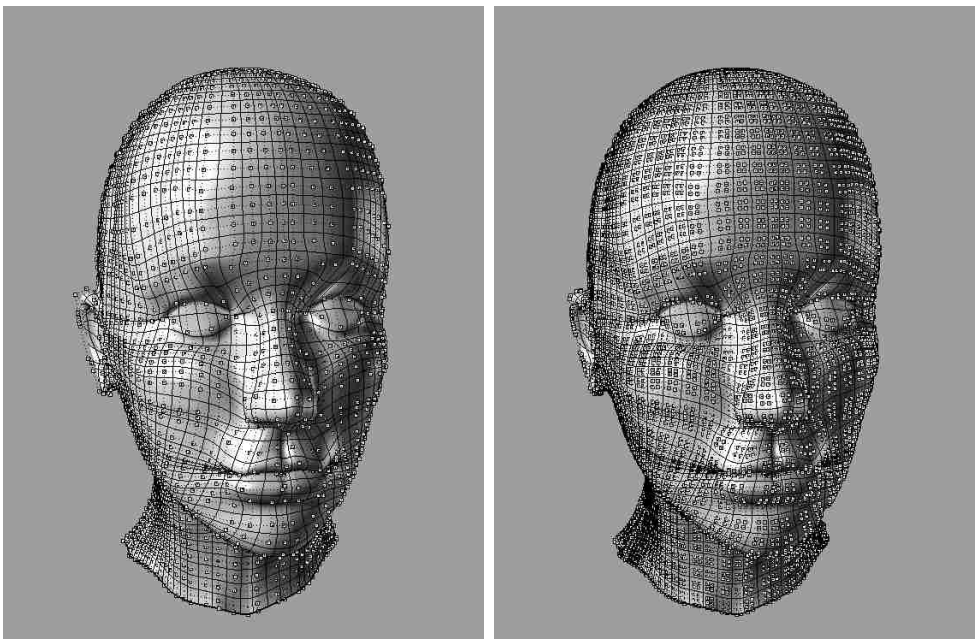
Any lower degree NURBS curve can be exactly represented by a higher degree NURBS curve, through a process called degree elevation. There are two disadvantages of degree elevation: First, it takes more data to define the higher degree NURBS curve. A line segment defined as a degree three NURBS curve still requires a minimum of four control points. Second, moving any single control point will increase the actual degree of the represented curve. For example, moving a control point of a degree elevated line will generally result in a curve that is no longer representable as a line. This is quite problematic if you require the output of your system to be linear.

These same problems extend to surfaces. Figure 1.3 shows the explosion in the data caused by degree elevation. Figure 1.2 shows an example of inadvertently increasing the actual degree of a surface through degree elevation and point editing.



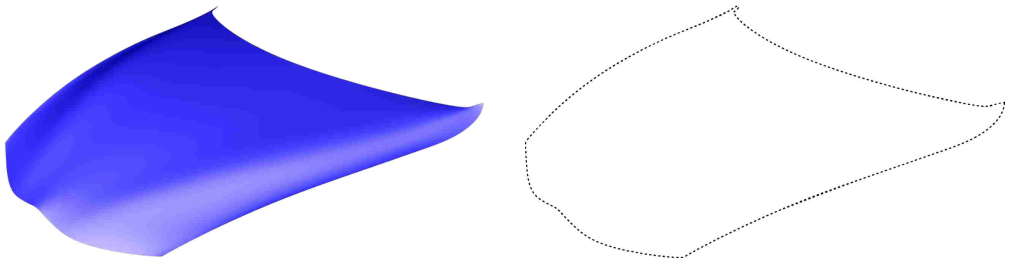
(a) (b) (c)

Figure 1.2: A degree one NURBS surface can be exactly converted to a degree three NURBS surface. However, this adds many control points, and moving the control points will generally change the shape such that it will no longer be representable as a degree one NURBS surface.



(a) (b)

Figure 1.3: A degree two NURBS head defined with 4712 control points. When the surface is exactly converted to a degree three NURBS, the same shape is defined by 18300 control points.



(a) A degree four NURBS car hood (b) No corresponding degree three surface

Figure 1.4: A degree four NURBS car hood. In general, higher degree surfaces cannot be exactly represented by lower degree surfaces. Because T-Splines have previously only been defined for degree three, the surface was not representable as a T-Spline.

This is a problem, because some manufacturing or analysis techniques require the surfaces output by the modeling system to be a certain degree.

While it is possible to represent a lower degree NURBS in a higher degree through degree elevation, in general a higher degree NURBS cannot be exactly represented by a lower degree NURBS. For example, a circle cannot be exactly represented by a finite set of line segments. NURBS are defined for arbitrary degree, and in practice higher degrees are somtimes used, as in 1.4. Higher degree NURBS cannot be exactly represented as degree three T-Splines, so any conversion necessarily involves approximation.

A definition of arbitrary degree T-Splines will allow T-Spline surfaces to represent surfaces in the degree most appropriate, improving the speed and compatibility of T-Splines by making them a proper superset of tensor-product NURBS surfaces of arbitrary degree.

# Chapter 2

## T-Splines of Odd Degree

Section 2.1 reviews the definition of cubic T-Splines, taken primarily from [2] and covering information introduced in [1]. Section 2.2 generalizes that definition to all odd degrees.

## 2.1 Cubic T-Splines Review

A control grid for a T-Spline surface is called a T-mesh. If a T-mesh forms a rectangular grid with no T-Junctions, the T-Spline degenerates to a B-spline surface.

Knot information for T-Splines is expressed using knot intervals, non-negative numbers that indicate the difference between two knots. A knot interval is assigned to each edge in the T-mesh. Figure 2.1 shows the pre-image of a portion of a T-mesh in $(s,t)$ parameter space; the $d_i$ and $e_i$ denote the knot intervals. Knot intervals are constrained by the relationship that the sum of all knot intervals along one side of any face must equal the sum of the knot intervals on the opposing side. For example, in Figure 2.1 on face $F_1$, $e_3 + e_4 = e_6 + e_7$, and on face $F_2$, $d_6 + d_7 = d_9$.

It is possible to infer a local knot coordinate system from the knot intervals on a T-mesh. To impose a knot coordinate system, we first choose a control point whose pre-image will serve as the origin for the parameter domain $(s,t) = (0,0)$. For the example in Figure 2.1, we designate $(s_0, t_0)$ to be the knot origin.

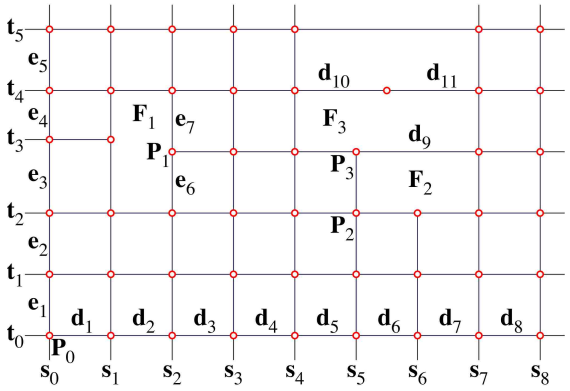



Figure 2.1: Pre-image of a T-mesh.

Once a knot origin is chosen, we can assign an $s$ knot value to each vertical edge in the T-mesh topology, and a $t$ knot value to each horizontal edge in the T-mesh topology. In Figure 2.1, those knot values are labeled $s_i$ and $t_i$. Based on our choice of knot origin, we have $s_0 = t_0 = 0$, $s_1 = d_1$, $s_2 = d_1 + d_2$, $s_3 = d_1 + d_2 + d_3$, $t_1 = e_1$, $t_2 = e_1 + e_2$, and so forth. Likewise, each control point has knot coordinates. For example, the knot coordinates for $\mathbf{P}_1$ are $(0, 0)$, for $\mathbf{P}_1$ are $(s_2, t_2 + e_6)$, for $\mathbf{P}_2$ are $(s_5, t_2)$, and for $\mathbf{P}_3$ are $(s_5, t_2 + e_6)$.

One additional rule for T-meshes explained in [1] is that if a T-junction on one edge of a face can legally be connected to a T-junction on an opposing edge of the face (thereby splitting the face into two faces), that edge must be included in the T-mesh. Legal means that the sum of knot vectors on opposing sides of each face must always be equal. Thus, a horizontal line would need to split face $F_1$ if and only if $e_3 = e_6$ and therefore also $e_4 = e_7$.

The knot coordinate system is used in writing an explicit formula for a T-Spline surface:

$$\mathbf{P}(s,t) = (x(s,t), y(s,t), z(s,t), w(s,t)) = \sum_{i=1}^{n} \mathbf{P}_i B_i(s,t) \tag{2.1}$$

where $\mathbf{P}_i = (x_i, y_i, z_i, w_i, )$ are control points in $P^4$ whose weights are $w_i$, and whose Cartesian coordinates are $\frac{1}{w_i}(x_i, y_i, z_i)$. Likewise, the Cartesian coordinates of

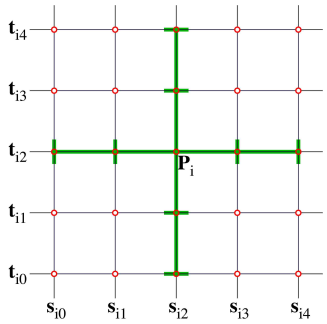



Figure 2.2: Knot lines for basis function $B_i(s,t)$.

points on the surface are given by

$$\frac{\sum_{i=1}^{n}(x_i, y_i, z_i)B_i(s,t)}{\sum_{i=1}^{n} w_i B_i(s,t)}. \tag{2.2}$$

The basis functions in (2.1) are $B_i(s,t)$ and are given by

$$B_i(s,t) = N[s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}](s)N[t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}](t) \tag{2.3}$$

where $N[s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}](s)$ is the cubic B-spline basis function associated with the knot vector

$$\mathbf{s}_i = [s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}] \tag{2.4}$$

and $N[t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}](t)$ is associated with the knot vector

$$\mathbf{t}_i = [t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}]. \tag{2.5}$$

as illustrated in Figure 2.2. The designer is free to adjust the weights $w_i$ to obtain additional shape control, as in rational B-splines.

The T-Spline equation is very similar to the equation for a tensor-product rational B-spline surface, the only difference being how the knot vectors $\mathbf{s}_i$ and $\mathbf{t}_i$ are determined for each basis function $B_i(s,t)$. Knot vectors $\mathbf{s}_i$ (2.4) and $\mathbf{t}_i$ (2.5) are

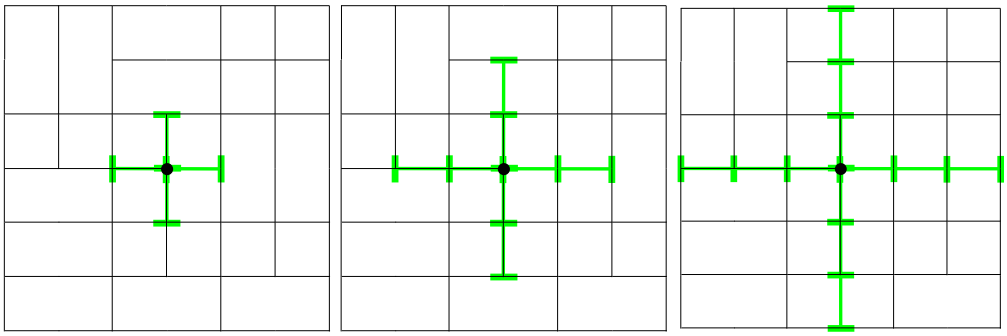
*inferred from the T-mesh* neighborhood of $\mathbf{P}_i$, as described in the following rule.

**Rule 1**. Knot vectors $\mathbf{s}_i$ (2.4) and $\mathbf{t}_i$ (2.5) for the basis function of $\mathbf{P}_i$ are determined as follows. $(s_{i2}, t_{i2})$ are the knot coordinates of $\mathbf{P}_i$. Consider a ray in parameter space $\mathbf{R}(\alpha) = (s_{i2} + \alpha, t_{i2})$. Then $s_{i3}$ and $s_{i4}$ are the $s$ coordinates of the first two $s$-edges intersected by the ray, not including the initial $(s_{i2}, t_{i2})$. By $s$-edge, we mean a vertical line segment of constant $s$. The other knots in $\mathbf{s}_i$ and $\mathbf{t}_i$ are found in like manner.

We illustrate Rule 1 by a few examples. The knot vectors for $\mathbf{P}_1$ in Figure 2.1 are $\mathbf{s}_1 = [s_0, s_1, s_2, s_3, s_4]$ and $\mathbf{t}_1 = [t_1, t_2, t_2 + e_6, t_4, t_5]$. For $\mathbf{P}_2$, $\mathbf{s}_3 = [s_3, s_4, s_5, s_6, s_7]$ and $\mathbf{t}_2 = [t_0, t_1, t_2, t_2{+}e_6, t_4]$. For $\mathbf{P}_3$, $\mathbf{s}_3 = [s_3, s_4, s_5, s_7, s_8]$ and $\mathbf{t}_2 = [t_1, t_2, t_2{+}e_6, t_4, t_5]$. Once these knot vectors are determined for each basis function, the T-Spline is defined using (2.1) and (2.3).

## 2.2 Extending to Arbitrary Odd Degree

This scheme can easily be extended to handle T-Splines of arbitrary odd degree, simply by changing the number of knots that are collected in each direction, as shown in Figure 2.3. Here, a degree one T-Spline collects one knot interval in each direction, a degree three T-Spline collects two knot intervals in each direction, and in general a degree $n$ T-Spline collects $\frac{n+1}{2}$ knot intervals in each direction.



(a) Degree one (b) Degree three (c) Degree five

Figure 2.3: Odd degree T-Spline knot inference examples. The degree determines how many knot intervals to gather in each direction.

# Chapter 3

## T-Splines of Even Degree

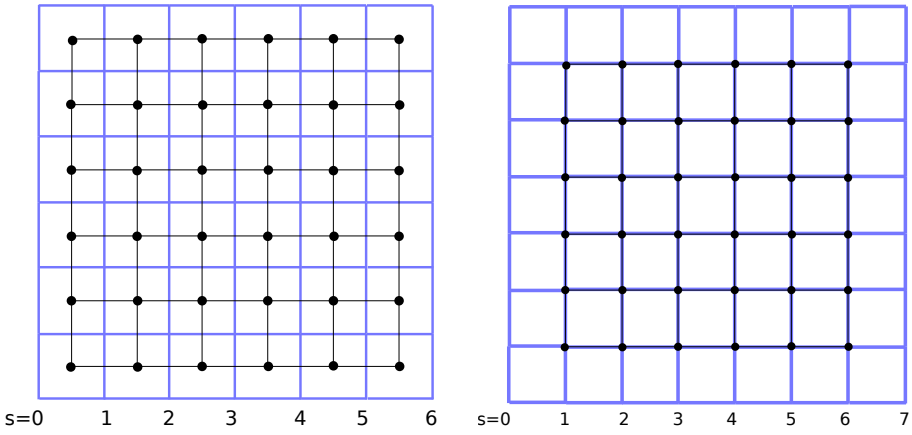
Chapter 2 shows how easily the definition of cubic T-Splines can be extended to all odd degrees. Extending the definition to even degrees is not so straightforward. The challenge arises from the fact that in odd degree NURBS and T-Splines, knot intervals correspond to edges, while in even degrees, knot intervals correspond to vertices, as shown in Figure 3.1.

### 3.1 The Knot Inference Mesh

In order to unambiguously deal with even degree T-Splines, we introduce a new construct called the knot inference mesh.

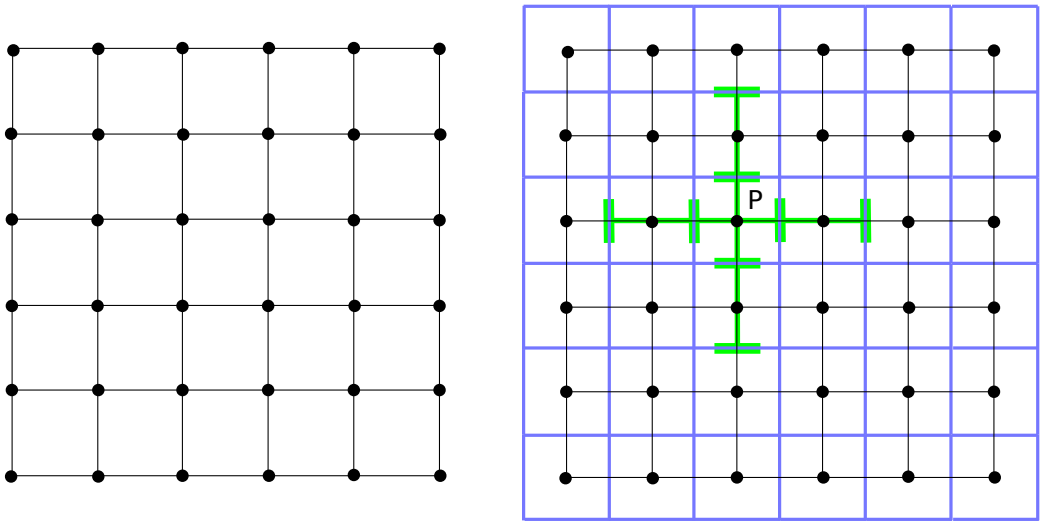
Consider the degree two NURBS in Figure 3.2a. A pair of knot intervals is stored for each control point, one for the $s$ direction and one for the $t$ direction. We take the knot intervals at each control point to define the width and height of a rectangle in parameter space, called the control point's *knot rectangle*. The knot rectangles of adjacent control points are likewise adjacent, and the set of all knot rectangles form a tiling, as shown in 3.2b. In the grid case, the tiling is straightforward. We will refer to this tiling as the *knot inference mesh*.

As its name implies, we can determine the local knot vectors of each control point by using the knot inference mesh in conjunction with Rule 1: intersect rays originating at the control point with neighboring knot lines in parameter space. We



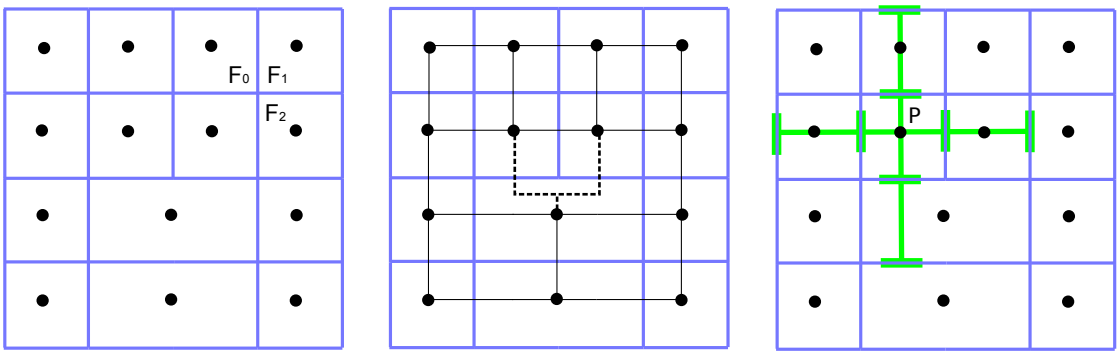

(a) Even degree (b) Odd degree

Figure 3.1: The preimage of an even degree NURBS control grid and odd degree NURBS control grid. The control mesh edges are shown in black, while the knot lines are shown in blue. Note that for odd degrees, the control mesh is aligned with the knot lines.




(a) A degree two NURBS control grid (b) With knot lines and the local knot vector of $P$

Figure 3.2: Local knot vectors can by determined by intersecting a ray with knot lines instead of control mesh lines




(a) Knot inference mesh (b) Control mesh (c) The local knot vector of $P$

Figure 3.3: Converting a knot inference mesh to a control mesh with step edges. Faces $F_0..F_2$ are labeled to illustrate adjacency.

take the control point to be at the center of its knot rectangle. In the case of a degree two NURBS, collect a total of four values in each parameter direction, as illustrated for $P$ in 3.2b. In the general degree $n$ case, collect a total of $n + 2$ knot values for each parameter direction. While shooting a ray to determine the local knot vector of blending functions is overkill for a NURBS, where you have grid topology and global knot vectors, the knot inference mesh becomes useful when T-Junctions are present.

We now consider even degree T-Splines. It is more straightforward to focus on the knot inference mesh than the control mesh.

In a knot inference mesh, the edges, faces, and vertices are formed by the tiling of knot rectangles. It is helpful to observe that for odd degree T-Splines, the control mesh is identical to the knot inference mesh, with each control point corresponding to a vertex in the knot inference mesh, as shown in Figure 3.1. For the even degree knot inference mesh, each control point corresponds to a knot rectangle, or a face in the knot inference mesh, as shown in Figure 3.3.

The even degree knot inference mesh has the same topology and knot interval relationships as an odd degree T-Spline control mesh. Therefore, the validity rules for an even degree knot inference mesh are the same as an odd degree T-mesh: the sums of the knot intervals on opposite sides of a face must be equal.

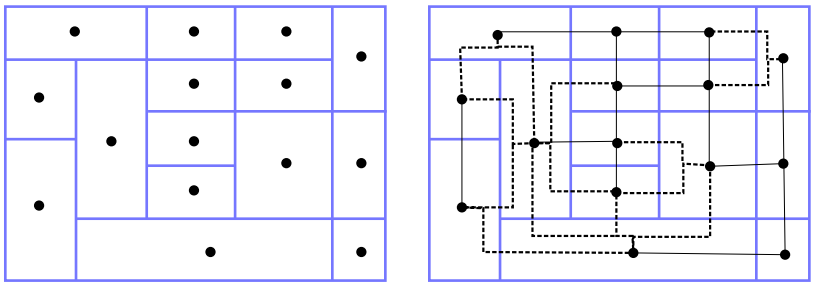
We now explain how to infer local knot vectors for blending functions corresponding to a control point $P$ in an even degree knot inference mesh. The local knot vectors for blending functions are determined using Rule 1 applied to the knot inference mesh. From the center of the knot rectangle we shoot a ray in parameter space, traversing the knot inference mesh in the same way as odd degree control meshes. We collect a total of $n + 2$ knot values to form each local knot vector. Finally, the surface is defined using (2.1), where $B_i(s, t)$ are of the appropriate degree.

## 3.2 Even Degree Control Meshes

The knot inference mesh is an elegant way to define the surface. However, the conventional method of defining a NURBS surface is a control mesh. For even degree T-Splines to be compatible with NURBS surfaces, their control mesh topology must be compatible with NURBS control meshes. In this section, we describe the even degree T-Spline control mesh, and explain conversion between the control mesh and the knot inference mesh.

To determine the topology of an even degree T-Spline control mesh from its knot inference mesh, first put a control point in the center of each face in the knot inference mesh. Next, draw connecting lines between pairs of control points lying on adjacent faces. (Faces are considered adjacent if they share an edge — for example, in Figure 3.3a, $F_0$ is adjacent to $F_1$, but it is not adjacent to $F_2$.)

If the control points are isoparametric (i.e., have the same $s$ and $t$ parameter values), they are connected with a single straight line segment. If the control points are on adjacent faces but are not isoparametric, they are connected with a *step edge*. A step edge is drawn as a step using three line segments — two "zigs" and a "zag". It conveys connectivity information between two control points that are not isoparametric, as illustrated in Figure 3.3. The knot interval on the zag is the parametric offset between the control points, and is stored in the control mesh.



(a) Knot inference mesh (b) Control mesh

Figure 3.4: A more complex example of an even degree T-Spline.

Step edges occur when the common edge between two faces in the knot inference mesh has a T-Junction that terminates on one of the faces. When drawing the step line, it is useful to draw it halfway between the side of the face that has the T-Junction and that face's control point.

Figure 3.4 shows a more complicated example of an even degree T-Spline.

In our presentation we have begun with a knot inference mesh, and derived the control mesh from it. In practice, we will often begin with a control mesh, and produce a knot inference mesh to determine the local knot vectors of the blending functions.

# Chapter 4

## T-Splines of Mixed Degree

A surface with "mixed degree" is one that has one degree in $s$ and a different degree in $t$. In cases where a mixed degree T-Spline surface is odd by odd, we use the odd degree control mesh/knot inference mesh, adjusting the number of knot values gathered in each direction as appropriate. Likewise, when the surface is even by even, we use the even degree knot inference mesh.

When the surface is odd by even, we associate control points with either vertical or horizontal edges of the surface. We can continue to use Rule 1 to collect knots by shooting rays in parameter space. The control points are taken to be in the center of their associated edges. Figure 4.2 is an example of a mixed degree T-Spline.

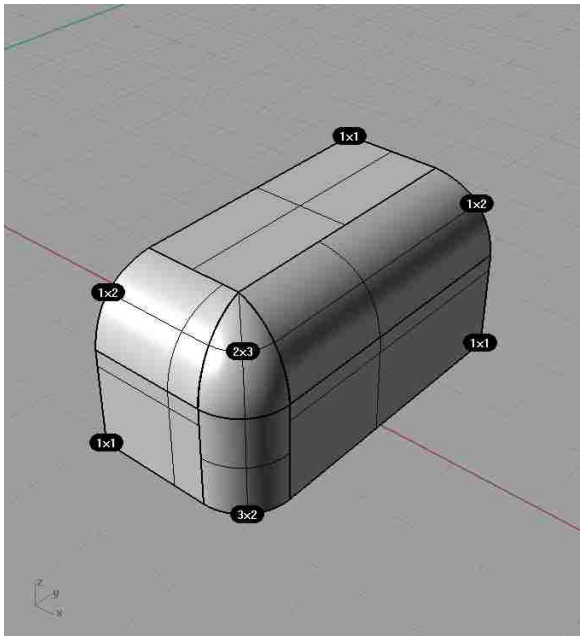



Figure 4.1: A portion of a typical CAD model with degrees labeled. Note that the model contains surfaces of varying degrees, including mixed degree surfaces.

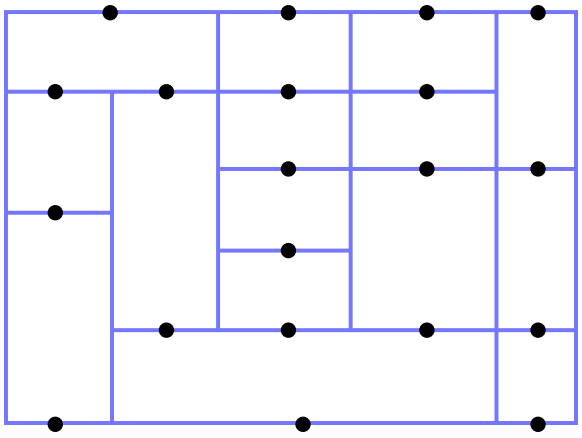


Figure 4.2: A mixed degree T-Spline. Note that each control point is on an edge of the knot inference mesh.

# Chapter 5

## Local Refinement

T-Spline local refinement means to insert one or more control points into a T-Spline mesh without changing the shape of the T-Spline surface. This procedure can also be called local knot insertion, since the addition of control points to a T-mesh must be accompanied by knots inserted into neighboring blending functions.

Section 5.1 covers refinement, taken primarily from [2]. This section is reproduced here to be self-contained, and modified to work on arbitrary degree surfaces.

## 5.1 T-Spline Local Refinement

This section presents our new algorithm for local refinement of T-Splines. Blending function refinement plays an important role in this algorithm, and is reviewed in Section 5.1.1. The notion of T-Spline spaces is introduced in Section 5.1.2. This concept is used in the local refinement algorithm in Section 5.1.3.

### 5.1.1 Blending Function Refinement

Denote by

$$N[s_0, s_1, \ldots, s_{n+1}](s)$$

a B-Spline basis function of degree $n$ over a knot vector $\{s_0, s_1, \ldots, s_{n+1}\}$ and with support $[s_0, s_{n+1}]$. Upon inserting a knot $k$ for which $s_i \leq k \leq s_{i+i}$, the basis function

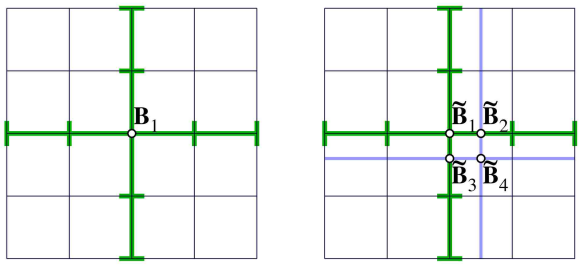



Figure 5.1: Sample Refinement of $B_1(s,t)$.

is split into two scaled basis functions:

$$N[s_0, s_1, \ldots, s_{n+1}](s) = c_i N[s_0, \ldots, s_i, k, s_{i+1}, \ldots, s_n](s) + d_i N[s_1, \ldots, s_i, k, s_{i+1}, \ldots, s_{n+1}](s) \tag{5.1}$$

where

$$c_i = \begin{cases} \frac{k-s_0}{s_{n+1}-s_0} & k < s_n \\ 1 & k \geq s_n \end{cases}$$

$$d_i = \begin{cases} \frac{s_{n+1}-k}{s_{n+1}-s_1} & k > s_1 \\ 1 & k \leq s_1 \end{cases}$$

A T-Spline function $B(s,t)$ can undergo knot insertion in either $s$ or $t$, thereby splitting it into two scaled blending functions that sum to the initial one. Further insertion into these resultant scaled blending functions yields a set of scaled blending functions that sum to the original. For example, Figure 5.1.a shows the knot vectors for a cubic T-Spline blending function $B_1$, and Figure 5.1.b shows a refinement of the knot vectors in Figure 5.1.a. By appropriate application of (5.1), we can obtain

$$B_1(s,t) = c_1^1 \tilde{B}_1(s,t) + c_1^2 \tilde{B}_2(s,t) + c_1^3 \tilde{B}_3(s,t) + c_1^4 \tilde{B}_4(s,t). \tag{5.2}$$

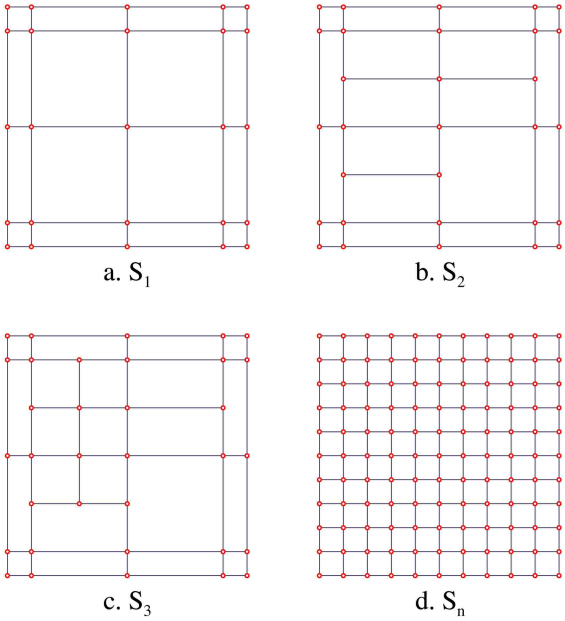



Figure 5.2: Nested sequence of cubic T-Spline spaces.

### 5.1.2 T-Spline Spaces

We define a T-Spline space to be the set of all T-Splines that have the same T-mesh topology, knot intervals, and knot coordinate system. Thus, a T-Spline space can be represented by the diagram of a pre-image of a T-mesh such as in Figure 2.1. Since all T-Splines in a given T-Spline space have the same pre-image, it is proper to speak of the pre-image of a T-Spline space. A T-Spline space $S_1$ is said to be a subspace of $S_2$ (denoted $S_1 \subset S_2$) if local refinement of a T-Spline in $S_1$ will produce a T-Spline in $S_2$ (discussed in Section 5.1.3). If $T_1$ is a T-Spline, then $T_1 \in S_1$ means that $T_1$ has a control grid whose topology and knot intervals are specified by $S_1$.

Figure 5.2 illustrates a nested sequence of cubic T-Spline spaces, that is, $S_1 \subset S_2 \subset S_3 \subset \cdots \subset S_n$.

Given a T-Spline $\mathbf{P}(s,t) \in S_1$, denote by $\mathbf{P}$ the column vector of control points for $\mathbf{P}(s,t)$, and given a second T-Spline $\tilde{\mathbf{P}}(s,t) \in S_2$, such that $\mathbf{P}(s,t) \equiv \tilde{\mathbf{P}}(s,t)$. Denote by $\tilde{\mathbf{P}}$ the column vector of control points for $\tilde{\mathbf{P}}(s,t)$. There exists a linear

transformation that maps $\mathbf{P}$ into $\tilde{\mathbf{P}}$. We can denote the linear transformation

$$M_{1,2}\mathbf{P} = \tilde{\mathbf{P}}. \tag{5.3}$$

The matrix $M_{1,2}$ is found as follows.

$\mathbf{P}(s,t)$ is given by (2.1), and

$$\tilde{\mathbf{P}}(s,t) = \sum_{j=1}^{\tilde{n}} \tilde{\mathbf{P}}_j \tilde{B}_j(s,t) \tag{5.4}$$

Since $S_1 \subset S_2$, each $B_i(s,t)$ can be written as a linear combination of the $\tilde{B}_j(s,t)$:

$$B_i(s,t) = \sum_{j=1}^{\tilde{n}} c_i^j \tilde{B}_j(s,t). \tag{5.5}$$

We require that

$$\mathbf{P}(s,t) \equiv \tilde{\mathbf{P}}(s,t). \tag{5.6}$$

This is satisfied if

$$\tilde{\mathbf{P}}_j = \sum_{i=1}^{n} c_i^j \mathbf{P}_i. \tag{5.7}$$

Thus, the element at row $j$ and column $i$ of $M_{1,2}$ in (5.3) is $c_i^j$. In this manner, it is possible to find transformation matrices $M_{i,j}$ that maps any T-Spline in $S_i$ to an equivalent T-Spline in $S_j$, assuming $S_i \subset S_j$.

The definition of a T-Spline subspace $S_i \subset S_j$ means more than simply that the preimage of $S_j$ has all of the control points that the preimage of $S_i$ has. In some cases it is not possible to refine a T-Spline simply by adding a single control point to an existing T-mesh — other control points must also be added. Section 5.1.3 presents insight into why that is and presents our local refinement algorithm for T-Splines. This, of course, will allow us to compute valid superspaces of a given T-Spline space.

### 5.1.3 Local Refinement Algorithm

T-Spline local refinement means to insert one or more control points into a T-mesh without changing the shape of the T-Spline surface. This procedure can also be called local knot insertion, since the addition of control points to a T-mesh must be accompanied by knots inserted into neighboring blending functions.

The refinement algorithm we now present has two phases: the topology phase and the geometry phase. The topology phase identifies which (if any) control points must be inserted in addition to the ones requested. Once all required new control points are identified, the Cartesian coordinates and weights for the refined T-mesh are computed using the linear transformation presented in Section 5.1.2. We now explain the topology phase of the algorithm.

An important key to understanding this discussion is to keep in mind how in a T-Spline, the blending functions and T-mesh are tightly coupled: To every control point there corresponds a blending function, and each blending function's knot vectors are defined by Rule 1. In our discussion, we temporarily decouple the blending functions from the T-mesh. This means that during the flow of the algorithm, we temporarily permit the existence of blending functions that violate Rule 1, and control points to which no blending functions are attached.

Our discussion distinguishes three possible violations that can occur during the course of the refinement algorithm:

- **Violation 1** A blending function is missing a knot dictated by Rule 1 for the current T-mesh.
- **Violation 2** A blending function has a knot that is not dictated by Rule 1 for the current T-mesh.
- **Violation 3** A control point has no blending function associated with it.

If no violations exist, the T-Spline is valid. If violations do exist, the algorithm resolves them one by one until no further violations exist. Then a valid superspace has been found.

The topology phase of our local refinement algorithm consists of these steps:

1. Create violations. This is done by inserting new control points into the T-mesh and/or new knots in blending functions.
2. If any blending function is guilty of Violation 1, perform the necessary knot insertions into that blending function.
3. If any blending function is guilty of Violation 2, add an appropriate control point into the T-mesh.
4. Repeat Steps 2 and 3 until there are no more violations.

Resolving all cases of Violation 1 and 2 will automatically resolve all cases of Violation 3.

In the odd degree case, cases of Violation 2 can be resolved by splitting edges, and therefore knots. In cases where the required knot is mid-face, the entire face will need to be split for Rule 1 to produce the desired local knot vector. This is identical when using the knot inference mesh for arbitrary degree, the only difference being that the mid-face case will happen more often.

We illustrate the algorithm with an example. Figure 5.3.a shows an initial T-mesh into which we wish to insert one control point, $\mathbf{P}_2$. Because the T-mesh in Figure 5.3.a is valid, there are no violations. But if we simply insert $\mathbf{P}_2$ into the T-mesh (Figure 5.3.b) *without changing any of the blending functions*, we introduce several violations. Since $\mathbf{P}_2$ has knot coordinates $(s_3, t_2)$, four blending functions become guilty of Violation 1: those centered at $(s_1, t_2)$, $(s_2, t_2)$, $(s_4, t_2)$, and $(s_5, t_2)$. To resolve these violations, we must insert a knot at $s_3$ into each of those blending functions, as discussed in Section 5.1.1. The blending function centered at $(s_2, t_2)$ is $N[s_0, s_1, s_2, s_4, s_5](s)N[t_0, t_1, t_2, t_3, t_4](t)$. Inserting a knot $s = s_3$ into the $s$ knot

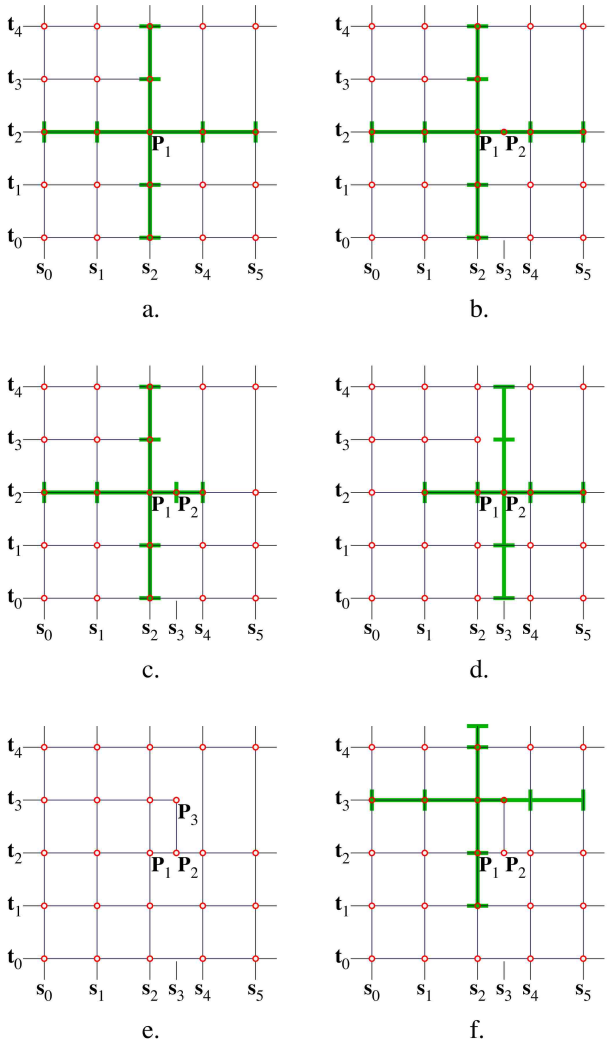



Figure 5.3: Local refinement example.

vector of this blending function splits it into two scaled blending functions:

$$c_2 N[s_0, s_1, s_2, s_3, s_4](s) N[t_0, t_1, t_2, t_3, t_4](t) \tag{5.8}$$

(Figure 5.3.c) and

$$d_2 N[s_1, s_2, s_3, s_4, s_5](s) N[t_0, t_1, t_2, t_3, t_4](t) \tag{5.9}$$

(Figure 5.3.d) as given in (5.1).

The blending function $c_2N[s_0, s_1, s_2, s_3, s_4](s)N[t_0, t_1, t_2, t_3, t_4](t)$ in Figure 5.3.c satisfies Rule 1. Likewise, the refinements of the blending functions centered at $(s_1, t_2)$, $(s_4, t_2)$, and $(s_5, t_2)$ all satisfy Rule 1. However, the $t$ knot vector of blending function $d_2N[s_1, s_2, s_3, s_4, s_5](s)N[t_0, t_1, t_2, t_3, t_4](t)$ shown in Figure 5.3.d is guilty of Violation 2 because the blending function's $t$ knot vector is $[t_0, t_1, t_2, t_3, t_4]$, but Rule 1 does not call for a knot at $t_3$. This problem cannot be remedied by refining this blending function; we must add an additional control point into the T-mesh.

The needed control point is $\mathbf{P}_3$ in Figure 5.3.e. Inserting that control point fixes the case of Violation 2, but it creates a new case of Violation 1. As shown in Figure 5.3.f, the blending function centered at $(s_2, t_3)$ has an $s$ knot vector that does not include $s_3$ as required by Rule 1. Inserting $s_3$ into that knot vector fixes the problem, and there are no further violations of Rule 1.

This algorithm is always guaranteed to terminate, because the only blending function refinements and control point insertions must involve knot values that initially exist in the T-mesh, or that were added in Step 1. In the worst case, the algorithm would extend all partial rows of control points to cross the entire surface. In practice, the algorithm typically requires few if any additional new control points beyond the ones the user wants to insert.

# Chapter 6

## Conclusion and Future Work

One interesting side effect of using the knot inference mesh to define the T-Spline surface is that it is possible to independently assign degrees for each control point. Control points can then be associated with vertices, edges, and faces of the same knot inference mesh, depending on the degree. Such a surface is called a multi-degree surface, similar to [13]. An example of a multi-degree T-Spline is shown in Figure 6.1.

While a multi-degree T-Spline is a generalization of odd, even, and mixed degree T-Splines and NURBS, there are several unanswered questions that arise: Is it possible to maintain partition of unity in such a surface, and what are the conditions for doing so? What are the constraints on the proximity of control points in order to define a surface with desired continuity? Is it possible to create a control mesh for a multi-degree T-Spline? How would a multi-degree T-Spline be refined? It would be useful to create a multi-degree surface through merging several surfaces of different degrees, and removing creases. What are the requirements and limitations of merging and crease removal in such a system?

Arbitrary degree extraordinary points are beyond the scope of this paper, but would vastly increase the value of this work. Since arbitrary degree extraordinary points have never been attempted with non-uniform knot intervals, there is little work to draw on.

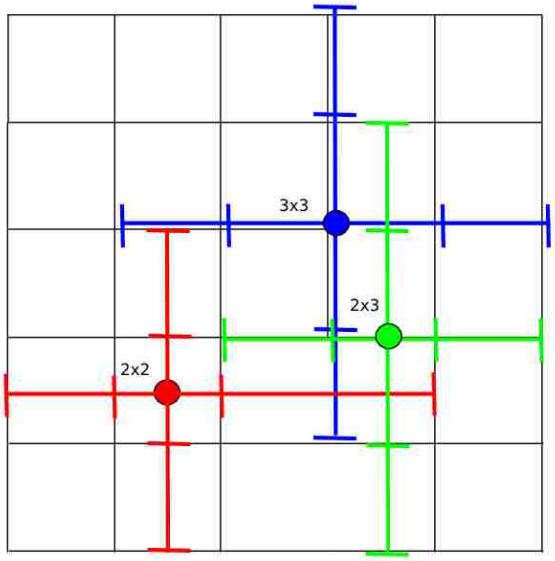



Figure 6.1: A multi-degree T-Spline. Various degree control points coexist on a knot inference mesh. The control points are on faces (even by even), edges (odd by even, even by odd), or vertices (odd by odd) as appropriate.

Algorithms that depend on arbitrary degree T-Splines have not been explored, such as degree elevation, hodograph computation, and Bézier patch extraction.

# Bibliography


[1] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, "T-Splines and T-NURCCS," *SIGGRAPH*, 2003.

[2] T. W. Sederberg, J. Zheng, T. Lyche, D. Cardon, G. T. Finnigan, and N. North, "T-Spline simplification and local refinement," *SIGGRAPH*, 2004.

[3] H. Ipson, "T-Spline merging," Master's thesis, Brigham Young University, 2005.

[4] D. L. Cardon, "T-Spline simplification," Master's thesis, Brigham Young University, 2007.

[5] R. F. Riesenfeld, "Applications of B-Spline approximation to geometric problems of computer-aided design." Ph.D. dissertation, Syracuse University, Syracuse, NY, USA, 1973.

[6] L. Piegl and W. Tiller, *The NURBS book (2nd ed.)*. New York, NY, USA: Springer-Verlag New York, Inc., 1997.

[7] G. Farin, *Curves and surfaces for CAGD: a practical guide*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[8] D. F. Rogers, *An introduction to NURBS: with historical perspective*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[9] W.-C. Li, N. Ray, and B. Lévy, "Automatic and interactive mesh to T-Spline conversion," in *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 191–200.

[10] Y. He, K. Wang, H. Wang, X. Gu, and H. Qin, "Manifold T-Spline," in *GMP*, 2006, pp. 409–422.

[11] Y. Wang, J. Zheng, and H. S. Seah, "Conversion between T-Splines and hierarchical B-Splines," in *Computer Graphics and Imaging*, 2005, pp. 8–13.

[12] J. Zheng, Y. Wang, and H. S. Seah, "Adaptive T-Spline surface fitting to z-map models," in *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. New York, NY, USA: ACM, 2005, pp. 405–411.

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