

T-spline Simplification and Local Refinement

Thomas W. Sederberg, David L. Cardon
G. Thomas Finnigan, Nicholas S. North
Brigham Young University

Jianmin Zheng
Nanyang Technological University

Tom Lyche
Oslo University

Abstract

A typical NURBS surface model has a large percentage of superfluous control points that significantly interfere with the design process. This paper presents an algorithm for eliminating such superfluous control points, producing a T-spline. The algorithm can remove substantially more control points than competing methods such as B-spline wavelet decomposition. The paper also presents a new T-spline local refinement algorithm and answers two fundamental open questions on T-spline theory.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—curve, surface, solid and object representations;

Keywords: NURBS surfaces, T-splines, subdivision surfaces, local refinement, knot removal

1 Introduction

A serious weakness with NURBS models is that NURBS control points must lie topologically in a rectangular grid. This means that typically, a large number of NURBS control points serve no purpose other than to satisfy topological constraints. They carry no significant geometric information. In Figure 1.a, all the red NURBS control points are, in this sense, superfluous.

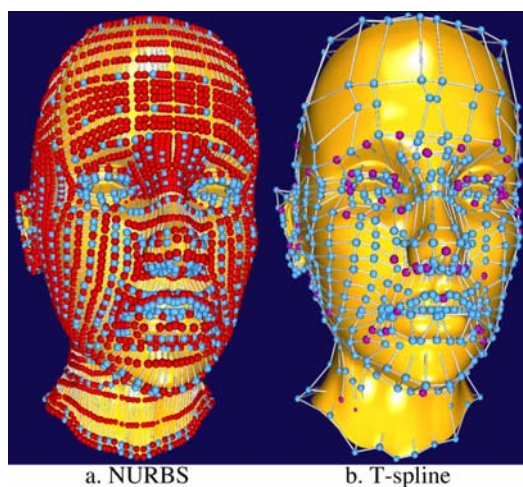


Figure 1: Head modeled (a) as a NURBS with 4712 control points and (b) as a T-spline with 1109 control points. The red NURBS control points are superfluous.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2004 ACM 0730-0301/04/0800-0276 \$5.00

T-splines [Sederberg et al. 2003] are a generalization of NURBS surfaces that are capable of significantly reducing the number of superfluous control points. Figure 1.b shows a T-spline control grid which was obtained by eliminating the superfluous control points from the NURBS model. The main difference between a T-mesh (i.e., a T-spline control mesh) and a NURBS control mesh is that T-splines allow a row of control points to terminate. The final control point in a partial row is called a T-junction. The T-junctions are shown in purple in Figure 1.b.

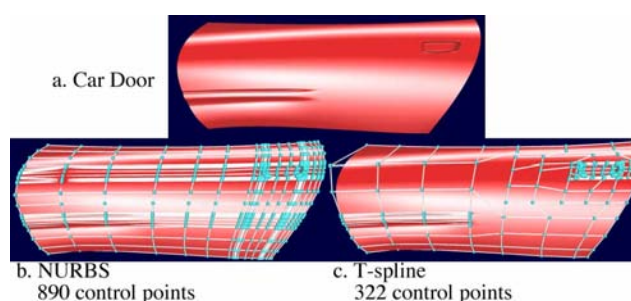


Figure 2: Car door modeled as a NURBS and as a T-spline.

Figure 2 shows another example in which the superfluous control points in a NURBS are removed to create a T-spline. The T-spline model is geometrically equivalent to the NURBS model, yet has only 1/3 as many control points.

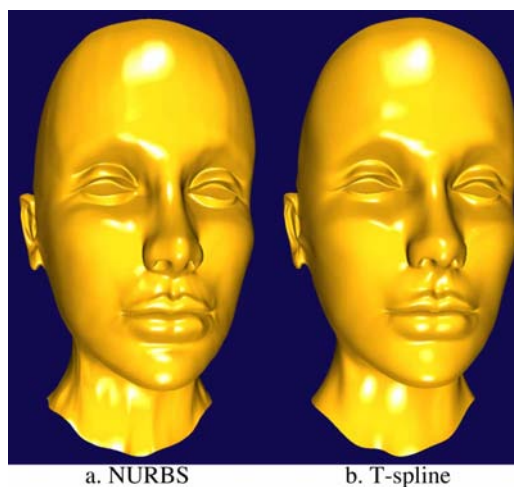


Figure 3: NURBS head model, converted to a T-spline.

Superfluous control points are a serious nuisance for designers, not merely because they require the designer to deal with more data, but also because they can introduce unwanted ripples in the surface as can be seen by comparing the forehead in the NURBS model in Figure 3.a with that of the T-spline model in Figure 3.b. Designers can waste dozens of hours on models such as this in tweaking the NURBS control points while attempting to remove unwanted ripples. Figure 1.a shows a NURBS head model. Figure 1 shows the

respective NURBS and T-spline control meshes for the surfaces in Figure 3. Over 3/4 of the 4712 NURBS control points are superfluous and do not appear in the T-spline control mesh.

This paper presents an algorithm for converting a NURBS model into a T-spline model. The process eliminates most superfluous control points. The algorithm was used to convert the NURBS model in Figure 1.a into the T-spline model in Figure 1.b. Conversion from NURBS to T-spline can be lossless (such as in the case of CAD objects like the car door) or an approximation error can be specified. An error threshold of 1% was used in the example in Figure 1, and the only observable difference between the two models is that the conversion to T-splines removed the unwanted ripples in the forehead. Thus, a problem for which designers can waste hours of time is resolved in the split second that it takes to convert from NURBS to T-spline using the algorithm presented in this paper.

We refer to this conversion process as T-spline simplification. Our algorithm makes heavy use of the operation of T-spline local refinement, introduced in [Sederberg et al. 2003]. However, [Sederberg et al. 2003] left this operation on uncertain footing. It was shown that in certain situations, a control point can be inserted into a particular location in a T-mesh topology only if some additional control points are also added. Furthermore, the number of additional control points can be quite large using the local refinement algorithm in [Sederberg et al. 2003]. This problem is illustrated in Figure 4.a, which shows a T-mesh into which we wish to insert control point P . The initial control points are the red, unlabeled ones. The algorithm in [Sederberg et al. 2003] requires the additional insertion of all eight control points labeled V . [Sederberg et al. 2003] concluded with an open question: Do there exist T-spline topologies into which it is impossible to insert a requested control point?

This paper answers that question by presenting a new local refinement algorithm with which it is provably always possible to insert a requested control point. Furthermore, the number of additional control points needed is reduced significantly. The algorithm is illustrated in Figure 4.b, where we insert P while only performing a single additional insertion V . This new local refinement algorithm plays a key role in the T-spline simplification algorithm presented in this paper.

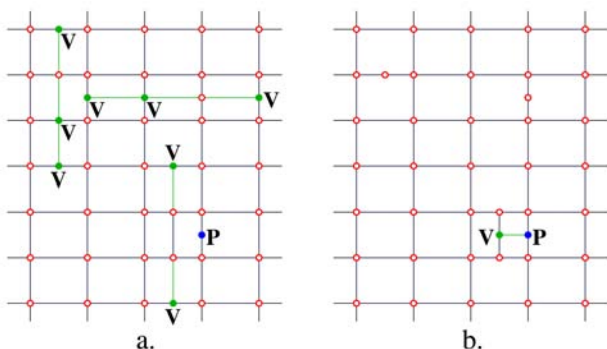


Figure 4: Local refinement using the old algorithm and the new algorithm.

The paper is organized as follows. Section 2 surveys pertinent literature on B-spline knot removal and insertion. Section 3 presents a brief review of T-splines. Section 4 provides equations for refining T-spline blending functions, introduces the notion of T-spline spaces, and presents the improved algorithm for local refinement (knot insertion) into a T-spline. This section also answers a second open question by presenting necessary and sufficient conditions for a T-spline to be standard (i.e., a T-spline whose blending functions form a partition of unity), and introduces semi-standard T-splines, a surprising type of rational T-spline in which some control

points have weights $\neq 1$, yet the T-spline is equivalent to a polynomial B-spline surface (all weights = 1). Section 5 presents the algorithm for local knot removal in a T-spline (of which B-spline is a special case). Section 6 concludes with some examples and discussion.

We restrict our discussion to the case of degree three B-splines and T-splines. However, adapting these algorithms to T-NURCCs (the arbitrary-topology version of T-splines) is straightforward.

2 Prior Work

Local refinement of NURBS curves and surfaces is accomplished through knot insertion, an exact process that does not alter the shape of the curve or surface. Several algorithms exist for knot insertion. The Oslo algorithm [Cohen et al. 1980] computes so-called discrete B-splines to define the B-spline basis transformation from a refined space of splines to a subspace. Boehm's algorithm [Boehm 1980] works directly with the B-spline coefficients. Mathematical insights like the blossoming principle [Seidel 1988; Goldman and Lyche 1993] for knot insertion have also been developed.

Knot insertion is a fundamental algorithm that can be used both as a mathematical tool for understanding and analyzing B-splines and as a practical tool for manipulating and rendering B-spline curves and surfaces [Goldman and Lyche 1993]. Applications of knot insertion include providing tools for straightforward evaluation of points on curves/surfaces, tessellation, rendering, and other geometric processing algorithms [Lane and Riesenfeld 1980]; performing bases conversion such as converting a B-spline curve to a Bézier [Boehm 1981]; adding extra degrees of freedom for shape modification or editing [Forsey and Bartels 1988].

Knot insertion works well to locally refine a B-spline curve: inserting one knot involves updating only a few control points in a local region. However, for a tensor-product B-spline surface, true local refinement is not possible because the insertion of one knot into either of the surface knot vectors causes an entire row or column of control points to be inserted [Lyche et al. 1985]. This problem has been addressed by hierarchical B-splines for local refinement and multiresolution editing [Forsey and Bartels 1988; Forsey and Wong 1998; Gonzalez-Ochoa and Peters 1999]. [Kraft 1997] presents a multilevel spline space that is a linear span of tensor product B-splines on different, hierarchically ordered grid levels. A selection mechanism for B-splines is provided, which guarantees linear independence to form a basis. [Weller and Hagen 1995] generalizes tensor-product B-spline surfaces by allowing the domain partition with knot segments and knot rays. The approach is restricted to so-called semi-regular bases. CHARMS [Grinspun et al. 2002] provides a method to produce adaptive refinements for 3D finite elements. The basic idea is to refine the basis functions, not elements. T-splines [Sederberg et al. 2003], the surface formulation upon which this paper is based, permit local refinement.

Knot removal is the inverse of knot insertion [Handscomb 1987]. A primary motivation of knot removal is data reduction [Lyche 1993]. The problem is to minimize the number of knots subject to an error tolerance. Another application of knot removal is shape fairing [Farin et al. 1987]. The continuity order can be increased by removing knots. Like knot insertion, knot removal can also be used to perform bases conversion, multiresolution analysis, and wavelet decomposition [Daehlen and Lyche 1992].

While knot insertion for a B-spline curve is always possible without error, deleting a knot without changing the curve is possible only when the two adjacent curve segments are C^l continuous with $l > n - m$ where n is the degree of the curve and m is the multiplicity of the knot. Therefore knot removal usually involves approximation. In practice, for a B-spline curve, if a control point almost satisfies a knot removal condition, the inverse Oslo algorithm can be performed to delete the knot [Lyche and Mørken 1987] and the

resulting curve will not deviate from the original curve very much. [Lyche and Mørken 1987] also generalizes the algorithm to surfaces. For a B-spline surface, when all control points of a row or column nearly satisfy the knot removal condition, a knot can be deleted. However, it does not often happen that a whole row or column of control points satisfies the condition simultaneously. The hierarchical or multiresolution approach [Daehlen and Lyche 1992] can avoid this problem.

The T-spline simplification algorithm in this paper most closely follows the idea of spline wavelet decomposition (see [Lyche et al. 2001] and its references) and thresholding (see, for example, [Schröder and Sweldens 1995]). For spline wavelet decomposition on the sphere using tensor product B-splines see [Lyche and Schumaker 2000].

3 T-splines

This section gives a brief review of T-splines as presented in [Sederberg et al. 2003]. A control grid for a T-spline surface is called a T-mesh. If a T-mesh forms a rectangular grid, 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 5 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 5 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 5, we designate (s_0, t_0) to be the knot origin.

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 5, 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}_0 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 $(s_5, t_2 + e_6)$.

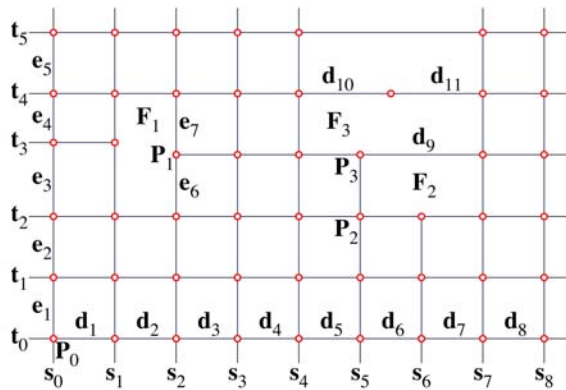


Figure 5: Pre-image of a T-mesh.

One additional rule for T-meshes explained in [Sederberg et al. 2003] 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:

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

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 points on the surface are given by

$$\frac{\sum_{i=1}^n (x_i, y_i, z_i) B_i(s, t)}{\sum_{i=1}^n w_i B_i(s, t)}. \quad (2)$$

The blending functions in (1) are $B_i(s, t)$ and are given by

$$B_i(s, t) = N[s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}](s) N[t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}](t) \quad (3)$$

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

$$\mathbf{s}_i = [s_{i0}, s_{i1}, s_{i2}, s_{i3}, s_{i4}] \quad (4)$$

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

$$\mathbf{t}_i = [t_{i0}, t_{i1}, t_{i2}, t_{i3}, t_{i4}]. \quad (5)$$

as illustrated in Figure 6. The designer is free to adjust the weights

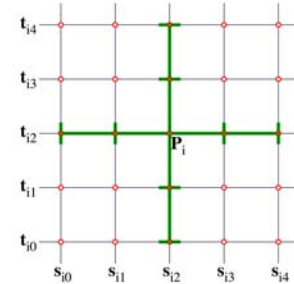


Figure 6: Knot lines for blending function $B_i(s, t)$.

w_i to obtain additional shape control, as in rational B-splines. As we shall see in Section 4, weights also play an important role in our new local refinement algorithm.

The T-spline equation is very similar to the equation for a tensor-product rational B-spline surface. The difference between the T-spline equation and a B-spline equation is in how the knot vectors \mathbf{s}_i and \mathbf{t}_i are determined for each blending function $B_i(s, t)$. Knot vectors \mathbf{s}_i (4) and \mathbf{t}_i (5) are *inferred from the T-mesh* neighborhood of \mathbf{P}_i . Since we will refer to the rule whereby the knot vectors are inferred, we formally state it as

Rule 1. Knot vectors \mathbf{s}_i (4) and \mathbf{t}_i (5) for the blending 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 5 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 , $\mathbf{s}_2 = [s_3, s_4, s_5, s_6, s_7]$ 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 $\mathbf{t}_3 = [t_1, t_2, t_2 + e_6, t_4, t_5]$. Once these knot vectors are determined for each blending function, the T-spline is defined using (1) and (3).

4 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 4.1. The notion of T-spline spaces is introduced in Section 4.2. This concept is used in the local refinement algorithm in Section 4.3 and in the T-spline simplification algorithm in Section 5.

4.1 Blending Function Refinement

If $\mathbf{s} = [s_0, s_1, s_2, s_3, s_4]$ is a knot vector and $\tilde{\mathbf{s}}$ is a knot vector with m knots with \mathbf{s} a subsequence of $\tilde{\mathbf{s}}$, then $N[s_0, s_1, s_2, s_3, s_4](s)$ can be written as a linear combination of the $m - 4$ B-spline basis functions defined over the substrings of length 5 in $\tilde{\mathbf{s}}$.

We now present all basis function refinement equations for the case $m = 6$. Equations for $m > 6$ can be found by repeated application of these equations.

If $\mathbf{s} = [s_0, s_1, s_2, s_3, s_4]$, $N(s) = N[s_0, s_1, s_2, s_3, s_4](s)$, and $\tilde{\mathbf{s}} = [s_0, k, s_1, s_2, s_3, s_4]$ then

$$N(s) = c_0 N[s_0, k, s_1, s_2, s_3](s) + d_0 N[k, s_1, s_2, s_3, s_4](s) \quad (6)$$

where $c_0 = \frac{k-s_0}{s_3-s_0}$ and $d_0 = 1$. If $\tilde{\mathbf{s}} = [s_0, s_1, k, s_2, s_3, s_4]$,

$$N(s) = c_1 N[s_0, s_1, k, s_2, s_3](s) + d_1 N[s_1, k, s_2, s_3, s_4](s) \quad (7)$$

where $c_1 = \frac{k-s_0}{s_3-s_0}$ and $d_1 = \frac{s_4-k}{s_4-s_1}$. If $\tilde{\mathbf{s}} = [s_0, s_1, s_2, k, s_3, s_4]$,

$$N(s) = c_2 N[s_0, s_1, s_2, k, s_3](s) + d_2 N[s_1, s_2, k, s_3, s_4](s) \quad (8)$$

where $c_2 = \frac{k-s_0}{s_3-s_0}$ and $d_2 = \frac{s_4-k}{s_4-s_1}$. If $\tilde{\mathbf{s}} = [s_0, s_1, s_2, s_3, k, s_4]$,

$$N(s) = c_3 N[s_0, s_1, s_2, s_3, k](s) + d_3 N[s_1, s_2, s_3, k, s_4](s) \quad (9)$$

where $c_3 = 1$ and $d_3 = \frac{s_4-k}{s_4-s_1}$. If $k \leq s_0$ or $k \geq s_4$, $N(s)$ does not change.

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 7.a shows the knot vectors for a T-spline blending function B_1 , and Figure 7.b shows a refinement of the knot vectors in Figure 7.a. By appropriate application of (6)–(9), we can obtain

$$B_1(s, t) = c_1^1 \tilde{B}_1(s, t) + c_1^2 \tilde{B}_2(s, t) + c_1^3 \tilde{B}_3(s, t) + c_1^4 \tilde{B}_4(s, t). \quad (10)$$

4.2 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 5. 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

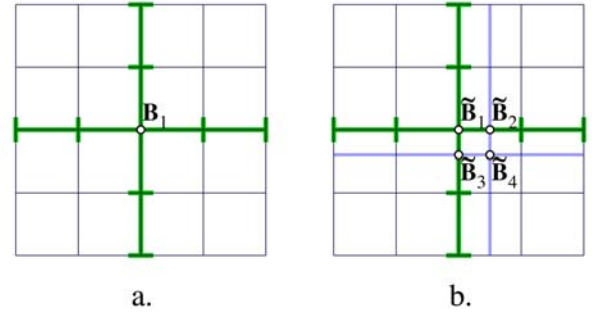


Figure 7: Sample Refinement of $B_1(s, t)$.

in S_1 will produce a T-spline in S_2 (discussed in Section 4.3). 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 8 illustrates a nested sequence of T-spline spaces, that is, $S_1 \subset S_2 \subset S_3 \subset \dots \subset S_n$.

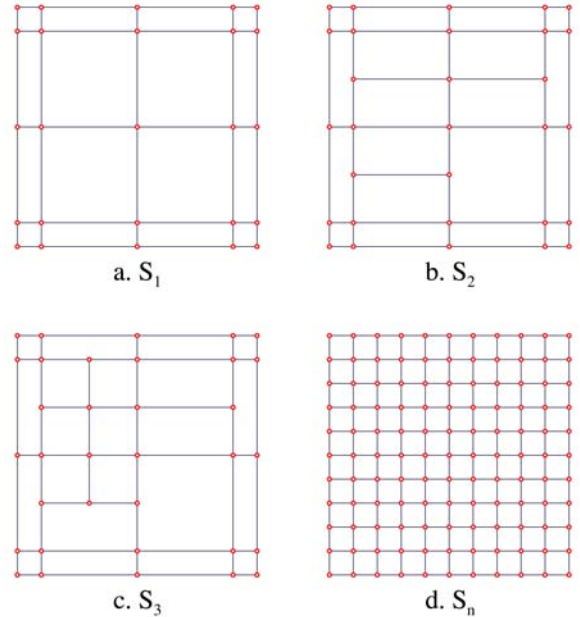


Figure 8: Nested sequence of T-spline spaces.

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

$$M_{1,2} \mathbf{P} = \tilde{\mathbf{P}}. \quad (11)$$

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

$\mathbf{P}(s, t)$ is given by (1), and

$$\tilde{\mathbf{P}}(s, t) = \sum_{j=1}^{\tilde{n}} \tilde{\mathbf{P}}_j \tilde{B}_j(s, t) \quad (12)$$

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

$$B_i(s, t) = \sum_{j=1}^{\tilde{n}} c_i^j \tilde{B}_j(s, t). \quad (13)$$

We require that

$$\mathbf{P}(s, t) \equiv \tilde{\mathbf{P}}(s, t). \quad (14)$$

This is satisfied if

$$\tilde{\mathbf{P}}_j = \sum_{i=1}^n c_i^j \mathbf{P}_i. \quad (15)$$

Thus, the element at row j and column i of $M_{1,2}$ in (11) 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. Recall how in Figure 4, it is not possible to simply add \mathbf{P} to the existing T-mesh without adding other control points as well. Section 4.3 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.

4.3 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 4.2. 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. Insert all desired control points into the T-mesh.
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.

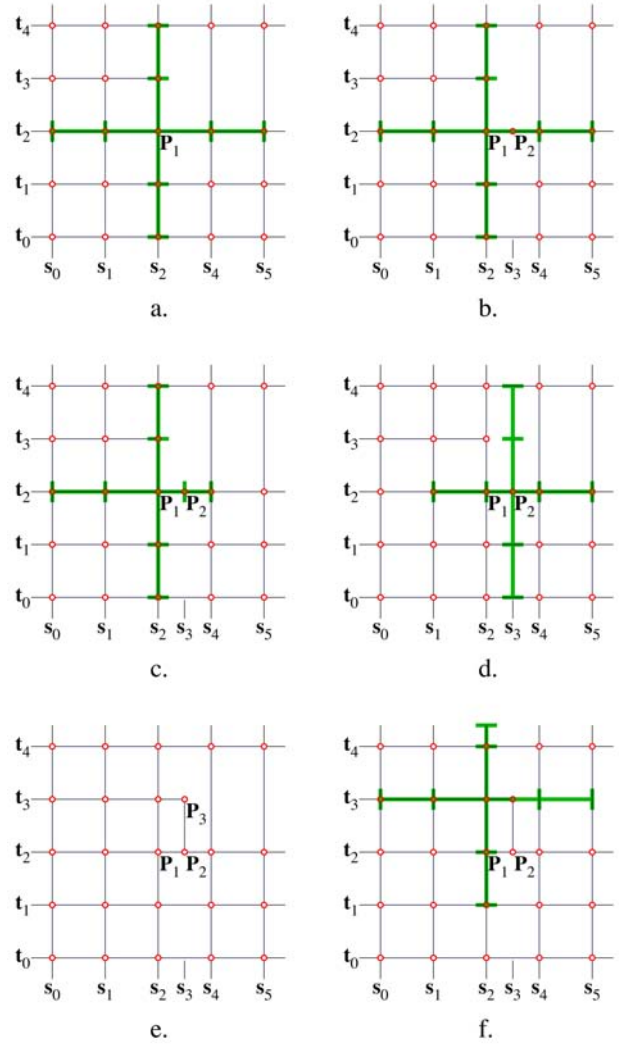


Figure 9: Local refinement example.

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

We illustrate the algorithm with an example. Figure 9.a shows an initial T-mesh into which we wish to insert one control point, \mathbf{P}_2 . Because the T-mesh in Figure 9.a is valid, there are no violations. But if we simply insert \mathbf{P}_2 into the T-mesh (Figure 9.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 4.1. 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 vector of this blending function splits it into two scaled blending functions: $c_2N[s_0, s_1, s_2, s_3, s_4](s)N[t_0, t_1, t_2, t_3, t_4](t)$ (Figure 9.c) and $d_2N[s_1, s_2, s_3, s_4, s_5](s)N[t_0, t_1, t_2, t_3, t_4](t)$ (Figure 9.d) as given in (8).

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 9.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 9.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 9.e. Inserting that control point fixes the case of Violation 2, but it creates a new case of Violation 1. As shown in Figure 9.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.

In summary, this refinement algorithm has two significant advantages over the algorithm in [Sederberg et al. 2003]: It is guaranteed to always work, and it normally requires far fewer unrequested control point insertions (as noted in Section 1 and illustrated in Figure 4).

4.4 Converting a T-spline into a B-spline surface

This refinement algorithm makes it easy to convert a T-spline $\in S_1$ into an equivalent B-spline surface $\in S_n$: simply compute the transformation matrix $M_{1,n}$ as discussed in Section 4.2.

[Sederberg et al. 2003] defines a *standard* T-spline to be one for which, if all weights $w_i = 1$, then $\sum_{i=1}^n w_i B_i(s, t) \equiv \sum_{i=1}^n B_i(s, t) \equiv 1$. This means that the denominator in (2) is identically equal to one; hence, the blending functions provide a partition of unity and the T-spline is polynomial. Thus, an algebraic statement of necessary and sufficient conditions for a T-spline to be standard is each row of $M_{1,n}$ sums to 1.

The insertion algorithm can produce a surprising result: a T-spline for which not all $w_i = 1$ but yet $\sum_{i=1}^n w_i B_i(s, t) \equiv 1$. We dub this a *semi-standard* T-spline. Figure 10 shows two simple examples of semi-standard T-splines. The integers (1 and 2) next to some edges are knot intervals. To verify that these T-splines

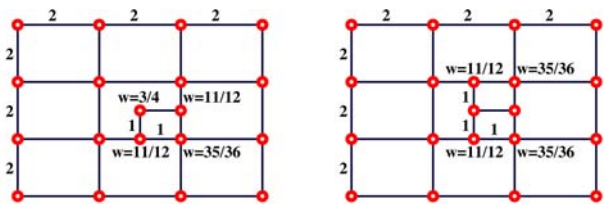


Figure 10: Semi-standard T-splines.

are semi-standard, convert them into B-spline surfaces; the control point weights will all be one.

The notion of T-spline spaces in Section 4.2 enables a more precise definition of semi-standard and non-standard T-splines. A semi-standard T-spline space S is one for which there exists some elements of S for which $\sum_{i=1}^n w_i B_i(s, t) \equiv 1$, and not all $w_i = 1$. A non-standard T-spline space is one for which no elements exist for which $\sum_{i=1}^n w_i B_i(s, t) \equiv 1$. These definitions are more precise because they allow for the notion of a rational T-spline (weights not all = 1) that is either standard, semi-standard, or non-standard. The distinction is made based on which type of T-spline space it belongs to.

5 T-spline Simplification

By T-spline simplification we mean the process of removing superfluous control points, as discussed in Section 1. Although not entirely straightforward, it is possible to derive a T-spline knot removal algorithm that is essentially the inverse of the local refinement algorithm in Section 4.3. Such an algorithm is useful to remove a single control point. However, it is difficult to remove large numbers of control points in this fashion.

Instead, our knot removal algorithm adapts multi-resolution techniques as follows. Consider a nested sequence of T-spline spaces

$$S_0 \subset S_1 \subset \dots \subset S_n \quad (16)$$

as discussed in Section 4.2. $T_n \in S_n$ is the surface (a T-spline or NURBS) that we wish to simplify. Denote by $T_i \in S_i$ the best least squares approximation to T_n (see [Lyche 1993]). Choose S_0 to consist of a 4×4 grid of control points such that the domain of T_0 and T_n are the same. Denote by \mathbf{P}^i the vector of control points for T_i . Then $D_{i,n} = (M_{i,n} \mathbf{P}^i - \mathbf{P}^n) \in S_n$ expresses the approximation error (see Section 4.2 for the meaning of $M_{i,n}$). Refinement of S_i into S_{i+1} is done by splitting offending faces in half. $D_{i,n}$ is used to identify offending faces. An offending face in S_i is one whose domain $[s_{min}, s_{max}] \times [t_{min}, t_{max}]$ contains the knot coordinates of a control point in $D_{i,n}$ whose length exceeds the threshold. Each con-

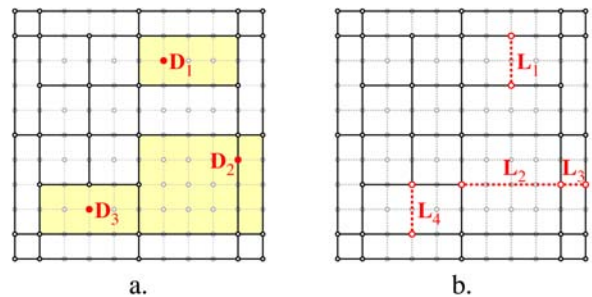


Figure 11: Identifying and splitting offending faces in S_i .

trol point in $D_{i,n}$ belongs to P^4 . We take as its length the square root of the sum of the squares of the four components. All of our examples have been standard T-splines and polynomial B-splines, so the fourth element of each control point in $D_{i,n}$ is always 0. This measure of length will also work fairly well for rational surfaces if the error tolerance is small.

Figure 11 illustrates this procedure. The control points D_1 , D_2 , and D_3 in Figure 11.a are control points in $D_{i,n}$ whose length is greater than the error threshold. The four shaded faces in Figure 11.a contain one of those three points, and are flagged for splitting. We experimented with several different ways to split faces and found that the most important principle is that faces should, in general, be split roughly in half in the direction which the face has the most knot lines. Specifically, if a face has m interior knot lines in the s direction and n interior knot lines in the t direction and $m > n$, then split along the $(m+1)/2^{th}$ knot line in the s direction. Thus, the face that contains D_3 in Figure 11.a is split with line L_4 in Figure 11.b, and the face that contains D_1 is split with line L_1 . D_2 is on the border between two faces, both of which are split. The face containing L_2 has three interior s knot lines and three interior t knot lines, so either direction could have been chosen for the split.

When we speak of splitting a face, we mean performing a local refinement in which the two endpoints of the line segment used to split the face are inserted into the T-mesh using the topology phase of the local refinement algorithm in Section 4.3. The refinement algorithm will sometimes need to insert a few additional control

points into the T-mesh in order to satisfy Rule 1. The refinement algorithm must be invoked, or else we will not form a nested sequence of T-spline spaces.

In summary, our algorithm parallels the methods of B-spline wavelet decomposition [Daehlen and Lyche 1992] with two important differences. First, we use a nested sequence of T-spline spaces instead of B-spline spaces. This allows us to split only the faces where the error is too large. Second, we do not formally compute a decomposition. In fact, we have no need to store the sequence of T_i ; they are used to identify which faces to split in forming S_{i+1} and can then be discarded.

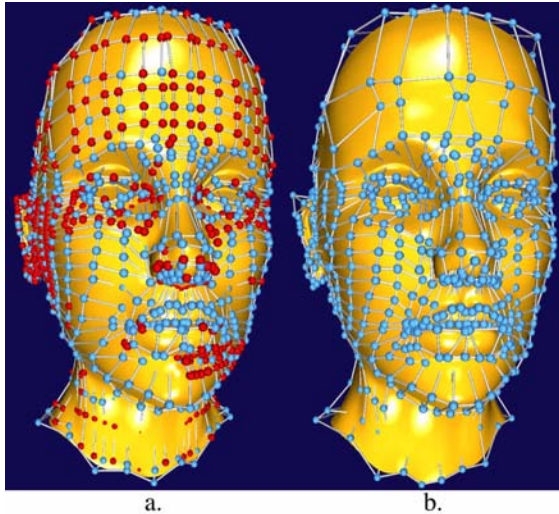


Figure 12: (a) T-spline representation of B-spline wavelet decomposition with 1926 control points. The red control points are superfluous. (b) The results of direct T-spline simplification, with 1109 control points. Approximation error = 1.5%.

It is noteworthy that similar results can be obtained by performing B-spline wavelet decomposition, thresholding the small coefficients, and then constructing a T-spline from the remaining non-zero wavelet coefficients. We implemented such an algorithm using the B-spline wavelet decomposition method in [Daehlen and Lyche 1992]. The non-zero wavelet terms were then merged into a T-spline using the local refinement algorithm in Section 4.3. We used an error threshold of 1.5%, which is the same value used in computing the T-spline simplification. The T-spline created using B-spline wavelet decomposition in Figure 12.a required 74% more control points than the direct T-spline simplification algorithm. We believe that this is mainly due to the extra control points that the local refinement algorithm often inserts. We stress that Figure 12.a is a T-spline, and not merely a tensor product grid. The conversion from B-spline wavelets to T-splines does not produce as many T-junctions as our direct T-spline simplification algorithm.

6 Results and Discussion

We conclude by presenting the results of performing T-spline simplification on three commercial-quality NURBS models that were provided to us courtesy of Zygot Media Group, Inc. In each case, the artists who created these models exercised care to avoid superfluous control points. Yet, our algorithm succeeded in eliminating about half of the control points. Furthermore, these models are comprised of several different NURBS surfaces. Using the techniques presented in [Sederberg et al. 2003], it is possible to merge adjoining NURBS surfaces into a single air-tight T-spline, without

adding new control points. By contrast, merging them into a single NURBS can cause the number of control points in the merged model to increase dramatically. The image in Figures 13.a, 14.a, and 15.a were all rendered using the T-spline control grid. The red control points in Figures 13.c, 14.c, and 15.c denote T-junctions.

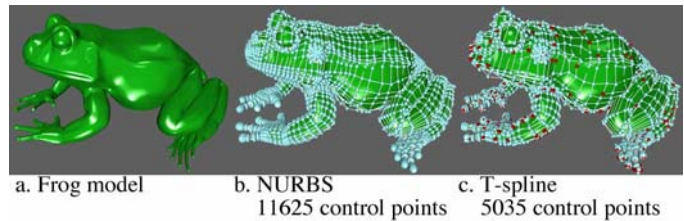


Figure 13: Model of a Frog (courtesy of Zygot Media Group, Inc.)

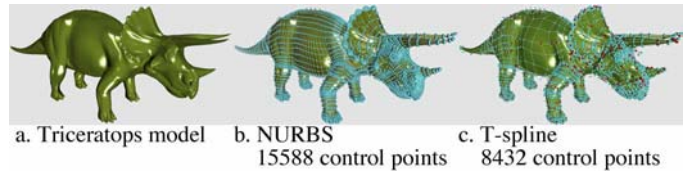


Figure 14: Model of a Triceratops (courtesy of Zygot Media Group, Inc.)

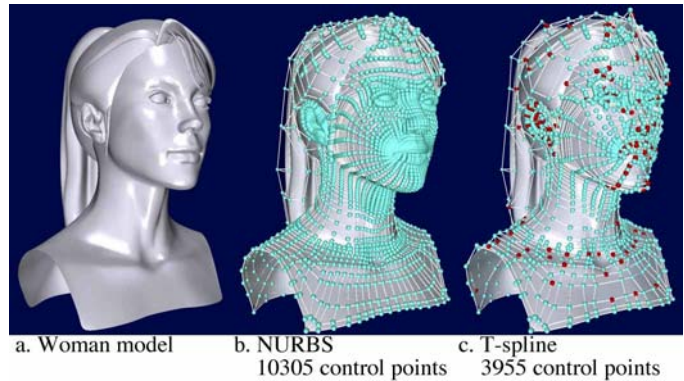


Figure 15: Model of a Woman (courtesy of Zygot Media Group, Inc.)

In summary, the local refinement algorithm presented in Section 4.3 performs much more efficiently than the algorithm in [Sederberg et al. 2003]. Furthermore, it is shown that this algorithm always works. The T-spline simplification algorithm is able to identify and remove a large percentage of superfluous control points, even in high-quality NURBS models.

Most of the examples in this paper have focused on T-spline simplification in which an existing NURBS model is converted into a T-spline. However, a potentially more common design scenario would be one in which an artist begins from scratch to create a T-spline model. In this setting, the local refinement algorithm in Section 4.3 will allow a designer to create a model that at each step has a minimal number of superfluous control points to get in the way. Figure 16 illustrates how this capability can simplify the design process. The artist begins with the coarse model in Step 1, then performs a series of refinements by adding control points in regions where more detail is called for, and then adjusting those control points. Note how the control point insertion is local to the

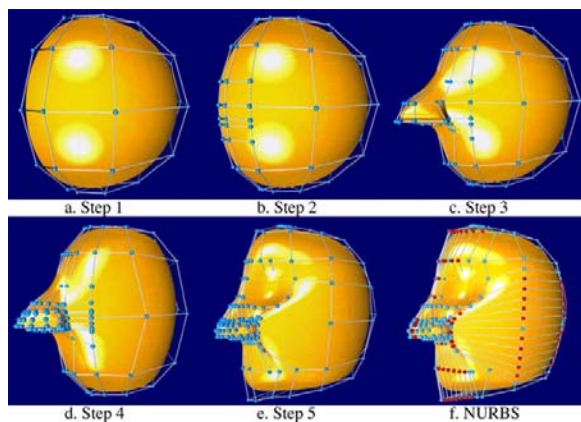


Figure 16: Initial steps in creating a model of a head using T-splines. (f) is the result of converting (e) into a NURBS surface.

region being refined. Figure 16.f shows the result of converting the T-spline in Figure 16.e into a NURBS.

In closing, we pose two open questions. First, Section 4.4 presents an algebraic statement of necessary and sufficient conditions for a T-spline space to be standard. What is the topological interpretation of those conditions? That is, what T-mesh configurations yield a standard T-spline? Second, in this paper we have referred to T-spline *blending* functions instead of *basis* functions. Are T-spline blending functions always linearly independent (and hence can they rightly be called basis functions)?

7 Acknowledgments

We gratefully recognize excellent feedback from several reviewers. The first five authors were supported in part by NSF grant CCR-9912411. Jianmin Zheng is partially supported by URC-SUG of Nanyang Technological University.

References

- BOEHM, W. 1980. Inserting new knots into B-spline curves. *Computer-Aided Design* 12 (July), 199–201.
- BOEHM, W. 1981. Generating the Bezier points of B-spline curves and surfaces. *Computer-Aided Design* 13 (Nov.), 365–366.
- COHEN, E., LYCHE, T., AND RIESENFELD, R. F. 1980. Discrete B-spline subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing* 14 (Oct.), 87–111.
- DAEHLEN, M., AND LYCHE, T. 1992. Decomposition of splines. In *Mathematical Methods in Computer Aided Geometric Design II*, Academic Press, New York, T. Lyche and L. L. Schumaker, Eds., 135–160.
- FARIN, G., REIN, G., SAPIDIS, N., AND WORSEY, A. J. 1987. Fairing cubic B-spline curves. *Computer Aided Geometric Design* 4, 1-2 (July), 91–103.
- FORSEY, D. R., AND BARTELS, R. H. 1988. Hierarchical B-spline refinement. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, J. Dill, Ed., vol. 22, 205–212.
- FORSEY, D., AND WONG, D. 1998. Multiresolution surface reconstruction for hierarchical b-splines. In *Graphics Interface*, 57–64.
- GOLDMAN, R. N., AND LYCHE, T. 1993. *Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces*. Philadelphia: SIAM.
- GONZALEZ-OCHOA, C., AND PETERS, J. 1999. Localized-hierarchy surface splines (less). In *Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, 7–15.
- GRINSPUN, E., KRYSL, P., AND SCHRÖDER, P. 2002. CHARMS: a simple framework for adaptive simulation. In *Proceedings of SIGGRAPH02*, ACM Press, 281–290.
- HANDSCOMB, D. 1987. Knot elimination: reversal of the Oslo algorithm. *International Series of Numerical Mathematics* 81, 103–111.
- KRAFT, R. 1997. Adaptive and linearly independent multilevel B-splines. In *Surface Fitting and Multiresolution Methods*, Vanderbilt University Press, A. L. Mhaut, C. Rabut, and L. L. Schumaker, Eds., vol. 2, 209–218.
- LANE, J. M., AND RIESENFELD, R. F. 1980. A theoretical development for the computer generation of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-2*, 1 (Jan.), 35–46.
- LYCHE, T., AND MØRKEN, K. 1987. Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design* 4, 3 (Nov.), 217–230.
- LYCHE, T., AND SCHUMAKER, L. L. 2000. A multiresolution tensor spline method for fitting functions on the sphere. *SIAM Journal of Scientific Computing* 22, 724–746.
- LYCHE, T., COHEN, E., AND MØRKEN, K. 1985. Knot line refinement algorithms for tensor product B-spline surfaces. *Computer Aided Geometric Design* 2, 1-3, 133–139.
- LYCHE, T., MØRKEN, K., AND QUAK, E. 2001. Theory and algorithms for non-uniform spline wavelets. In *Multivariate Approximation and Applications*, N. Dyn, D. Leviatan, D. Levin, and A. Pinkus, Eds. Cambridge University Press, 152–187.
- LYCHE, T. 1993. Knot removal for spline curves and surfaces. In *Approximation Theory VII*, Academic Press, E. W. Cheney, C. K. Chui, and L. L. Schumaker, Eds., 207–227.
- SCHRÖDER, P., AND SWELDENS, W. 1995. Spherical wavelets: Efficiently representing functions on the sphere. In *Proceedings of SIGGRAPH 95*, 161–172.
- SEDERBERG, T. W., ZHENG, J., BAKENOV, A., AND NASRI, A. 2003. T-splines and T-nurccs. *ACM Transactions on Graphics* 22, 3 (July), 477–484.
- SEIDEL, H.-P. 1988. Knot insertion from a blossoming point of view. *Computer Aided Geometric Design* 5, 1 (June), 81–86.
- WELLER, F., AND HAGEN, H. 1995. Tensor product spline spaces with knot segments. In *Mathematical Methods for Curves and Surfaces*, M. Daehlen, T. Lyche, and L. L. Schumaker, Eds. Vanderbilt University Press, Nashville, 563–572.