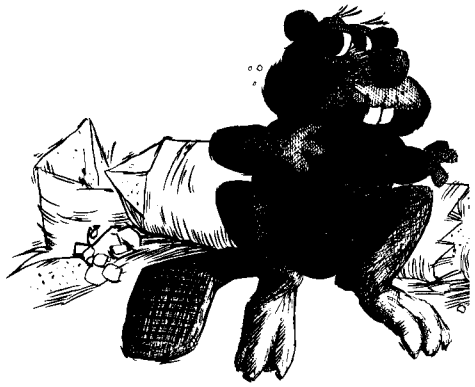
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



Tensor Product Slices

R.F. Pfeifle
R.H. Bartels
R.N. Goldman

Research Report
CS-89-56

November, 1989

Tensor Product Slices†

Ronald Pfeifle
Richard Bartels
Ronald Goldman

Computer Graphics Laboratory
Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada, N2L 3G1

ABSTRACT

Tensor products are widely used in computer graphics and computer aided geometric design for representing freeform surfaces. Standard tensor product surface patches retain vestiges of the original rectilinear shape of their domains. This feature is not always desired.

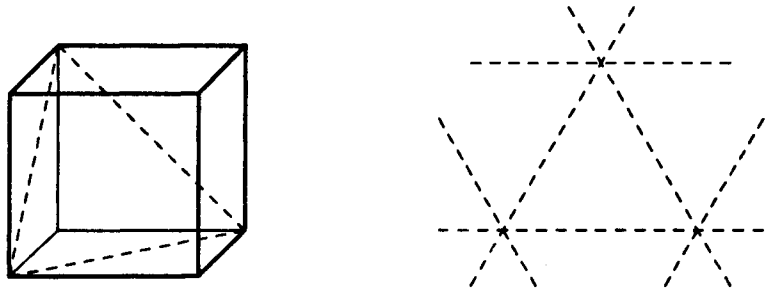
We introduce a variant of the tensor product, called the *tensor product slice*, which can be used to create multi-sided surface patches of non-rectilinear shape. We give examples to show that this variant retains many desirable properties of standard tensor products.

†This research is part of that being supported at the University of Waterloo Computer Graphics Laboratory by Canada's NSERC Operating, Strategic, and Infrastructure programs, by the Province of Ontario's ITRC program, by industrial grants from General Motors and Digital Equipment Corporation, and with equipment contributed by Digital Equipment Corporation and Silicon Graphics, Inc.

1. Introduction

A common technique used to represent surfaces in CAGD is tensor product splines [Böhm84]. Tensor products, however, retain vestiges of the geometry of their domains, producing four-sided surface patches with discernible corners. Domains can be trimmed so that individual patches are made to align properly with one another, but this is usually difficult to arrange.

Many properties are known about tensor products over rectilinear domains, as is information about the parametric and geometric continuity at joints in tensor product splines [Farin88]. Our purpose here is to find a method of specifying tensor product domains of 'multi-sided' shape, which still have predictable properties at their boundaries, that is, at the image of domain edges. We can do this if we ensure that the boundaries of the multi-sided domains we are trying to create coincide with the boundaries of some rectilinear domain. This leads us to consider cross sections of rectilinear domains, where the dimension of the rectilinear domain is greater than that of the domain we eventually wish to construct. For example, in Figure 1 we take a cube and slice it with a plane to obtain a triangle. Each edge of the triangle lies on the rectilinear boundary of the cube.



in 3-space *in the slicing plane*

Figure 1 — *Creating a Triangular Domain by Slicing a Cube*

If we examine such domain slices, and their images under tensor product mappings, we find that they indeed inherit most of the desirable properties of the original tensor products, including continuity properties along their boundaries. Moreover, by choosing the appropriate slice of a rectilinear domain of high enough dimension, we can form 2-dimensional surfaces with any number of sides.

To form triangular domains, we intersect a plane with a 3-dimensional box, as in Figure 1. To create a 2-dimensional domain with $n \geq 4$ sides, we consider the intersection of a plane with a rectilinear region of dimension $\left\lceil \frac{n}{2} \right\rceil$, over points $(u_0, \ldots, u_m) \in \mathbf{R}^m$, where $m = \left\lceil \frac{n}{2} \right\rceil - 1$, defined by the inequalities $a_k \leq u_k \leq b_k$ for $k = 0,\ldots,m$ ($\lceil x \rceil$ is the smallest integer greater than or equal to x). A region of the intersecting plane that satisfies one such pair of inequalities is a 'strip' in the plane. The intersection of all the strips is the set of all points both in the plane and the rectilinear domain. New strips can be introduced to increase the number of edges to the domain (see Figure 2 for the construction of a five-sided domain). This idea can be used constructively in order to generate domains for multi-sided patches — the intersecting plane is given a default parameterization, and the strips (and the variable associated with each strip) are defined in terms of this parameterization.

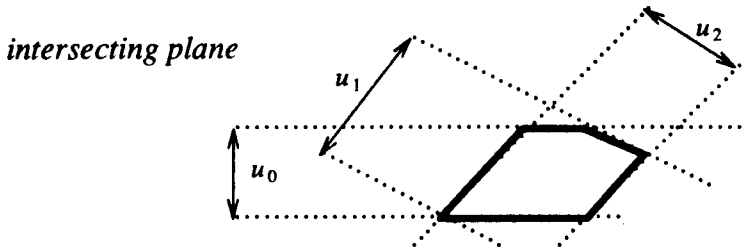


Figure 2 — *Five-Sided Domain for a Surface Patch*

Other approaches to multi-sided patches have been proposed. However, these techniques suffer either because they are complicated [Dahmen83, Dahmen84, Böhm84a, Hosaka84], or because they only produce patches with a fixed number of sides [Charrot84].

Our approach most closely resembles DeRose's S-patches [DeRose89]. However, DeRose uses arbitrary (non-planar) slices of high dimensional Bézier tetrahedra, while we use planar slices of arbitrary tensor products. The advantages of our approach are its simplicity and generality — a domain with an arbitrary number of sides can easily be created by slicing a tensor product of sufficiently high dimension.

2. Tensor Products

A set of blending functions (of size $j+1$) over an interval $\boldsymbol{D} = [a,b]$ in $\mathbf{R}$ is a set of functions $\{B_{i,j}(u) : i = 0,...,j\}$ such that $\sum_{i=0}^{j} B_{i,j}(u) = 1$ for all $u \in \boldsymbol{D}$.

Let n be a positive integer, and $[a_k,b_k]$ be real intervals, where $k = 0,...,n$. Let $\boldsymbol{D} = [a_0,b_0]\times[a_1,b_1]\times\cdots\times[a_n,b_n]$. Let j_k be positive integers for $k = 0,...,n$, and $B^k_{i_k,j_k}(u)$ for $i_k = 0,...,j_k$ be blending functions over $u_k \in [a_k,b_k]$. The superscript k in $B^k_{i_k,j_k}$ is present to emphasize that even if $j_l = j_k$, the sets of blending functions $\{B^k_{i_k,j_k} : i_k = 0,...,j_k\}$ and $\{B^l_{i_l,j_l} : i_l = 0,...,j_l\}$ are not necessarily identical. Let $(u_0,u_1,...,u_n) \in \boldsymbol{D}$ and let $P_{i_0,i_1,...,i_n}$ be vertices from $\mathbf{R}^m$, for $i_k = 0,...,j_k$, $k = 0,...,n$.

Definition:

$$F(u_0,u_1,...,u_n) = \sum_{i_0=0}^{j_0}\sum_{i_1=0}^{j_1}\cdots\sum_{i_n=0}^{j_n} P_{i_0,i_1,...,i_n} B^0_{i_0,j_0}(u_0)B^1_{i_1,j_1}(u_1)\cdots B^n_{i_n,j_n}(u_n)$$

is *the $n+1$-dimensional tensor product of the blending functions $B^k_{i_k,j_k}(u)$ over the domain $\boldsymbol{D}$ and the control vertices $P_{i_0,i_1,...,i_n}$.*

When $n = 1$, this definition gives us a standard tensor product surface [Farin88 p. 200], and when $n = 2$, this formulation describes a solid. Tensor products of arbitrary dimension will be necessary for our application of slicing.

3. Tensor Product Slices

Definition: Let m be some integer in the range $n-k+1,...,n$, and let $\gamma_m : \mathbf{R}^{n-k+1} \to \mathbf{R}$. An *$n-k+1$-dimensional slice of the $n+1$-dimensional domain $\boldsymbol{D}$* are those points $(u_0,u_1,...,u_n) \in \boldsymbol{D}$ that satisfy a set of constraints

$$u_m = \gamma_m(u_0, u_1, \ldots, u_{n-k}) \quad \text{for } m = n-k+1, \ldots, n$$

Definition: Let F be an $n+1$-dimensional tensor product over domain D. The *$n-k+1$-dimensional slice of F* is the set of points that form the image of the domain of the slice defined by the constraints γ_m, $m = n-k+1, \ldots, n$ under the tensor product mapping F.

Definition: A *linear slice* of a domain D is a slice of D with the property that each γ_m in the set of constraining functions is linear in its arguments.

The term *slice* will mean linear $n-k+1$-dimensional tensor product slice unless otherwise stated.

The definition of a tensor product slice requires that the dependent parameters of the slice occur in the final k positions. However, by reindexing, we can always force the dependent parameters to be the last k parameters. If the set of constraints is empty ($k = 0$), we obtain the original tensor product.

In general, the set of blending functions in $n-k+1$ variables formed by constraining the original blending functions of the tensor product will not be linearly independent, but will form a spanning set for a certain collection of functions. This is true even if the original set of blending functions was linearly independent.

Linear slices already give rise to interesting multi-sided domains for the case of 3-dimensional tensor products. We illustrate this phenomenon in the following two examples.

Example 1: Consider $\sum_{i_0=0}^{1} \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} P_{i_0,i_1,i_2} B_{i_0,1}(u) B_{i_1,1}(v) B_{i_2,1}(w)$, where $B_{i_0,1}(u)$, $B_{i_1,1}(v)$, and $B_{i_2,1}(w)$ are the linear Bernstein polynomials. Let $w = \gamma(u,v) = 1\frac{1}{2} - u - v$. This results in the 2-dimensional slice over a hexagonal domain (see Figure 3 for a diagram of the domain) given by

$$\begin{aligned}
&- P_{0,0,0}(1-u)(1-v)(\tfrac{1}{2}-u-v) + P_{0,0,1}(1-u)(1-v)(1\tfrac{1}{2}-u-v) - P_{0,1,0}(1-u)v(\tfrac{1}{2}-u-v) \\
&+ P_{0,1,1}(1-u)v(1\tfrac{1}{2}-u-v) - P_{1,0,0}u(1-v)(\tfrac{1}{2}-u-v) + P_{1,0,1}u(1-v)(1\tfrac{1}{2}-u-v) \\
&- P_{1,1,0}uv(\tfrac{1}{2}-u-v) + P_{1,1,1}uv(1\tfrac{1}{2}-u-v)
\end{aligned}$$

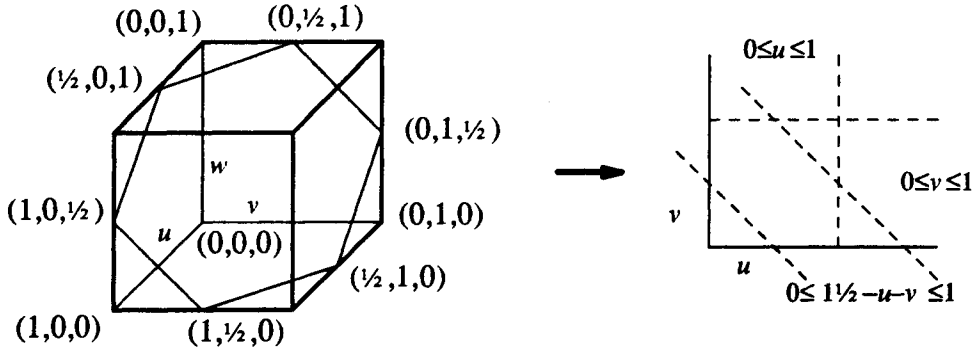


Figure 3 — *Hexagonal Domain given by $w = 1\frac{1}{2} - u - v$*

Example 2: The domain defined in the previous example can be used to tile the plane, as shown in Figure 4. Using cubic B-splines as our blending functions, we can create a surface with C^2-continuity. These domains do not line up with one another. Not shown in Figure 4 is the domain substructure, which is appropriately aligned.

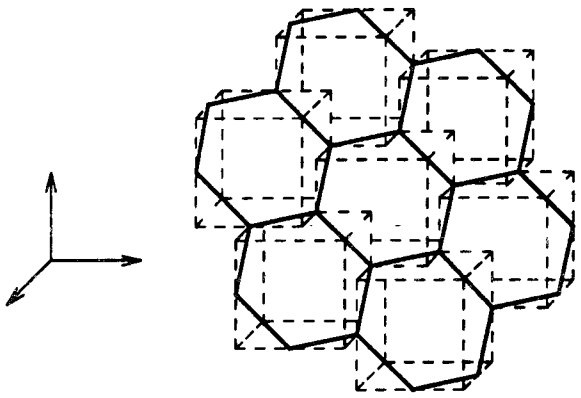


Figure 4 — *Tiling a Plane with Regular Hexagons Embedded in Cubes*

4. Generally Inherited Properties

Slicing preserves a number of tensor product properties. The value of a tensor product at a domain point is unaltered by slicing, so the original tensor product can be used to evaluate the tensor product slice. The only change in the method is that the values of dependent parameters must first be computed from the constraint equations. The slice of a tensor product with C^k-continuity will still be C^k-continuous. Polynomials of degree n (the sum of the powers of the variables of any term of the polynomial is $\leq n$) remain polynomials of degree n (the sliced polynomial has fewer variables, but they are of higher degree).

Since the blending functions of the tensor product sum to one over the domain of the tensor product, they also sum to one over the slice domain; therefore slices are affine invariant. If the blending functions are also non-negative over the domain, then slices will also have the convex hull property. If some control vertex is interpolated by a tensor product at some domain point that is in the slice domain, then that vertex is interpolated by the tensor product slice. Special evaluation techniques, subdivision algorithms, degree elevation formulae and other blending-function specific operations are also directly applicable to the slices of tensor products.

4.1. Dimension Raising

The shape of the $n-k+1$-dimensional slice domain depends upon the number of n-dimensional boundaries of the $n+1$-dimensional domain intersected by the slice. Increasing the dimensionality of the original domain increases the potential number of domain boundaries that may be intersected by the slice, and therefore allows for more complex slice domains. We can increase the dimension as follows:

$$
\begin{aligned}
&F(u_0,u_1,\ldots,u_n)\\
&\quad = 1 \times \sum_{i_0=0}^{j_0}\sum_{i_1=0}^{j_1}\cdots\sum_{i_n=0}^{j_n} P_{i_0,i_1,\ldots,i_n} B^0_{i_0,j_0}(u_0)B^1_{i_1,j_1}(u_1)\cdots B^n_{i_n,j_n}(u_n)\\
&\quad = \left(\sum_{i_{n+1}=0}^{j_{n+1}} B^{n+1}_{i_{n+1},j_{n+1}}(u_{n+1})\right) \times \sum_{i_0=0}^{j_0}\sum_{i_1=0}^{j_1}\cdots\sum_{i_n=0}^{j_n} P_{i_0,i_1,\ldots,i_n} B^0_{i_0,j_0}(u_0)B^1_{i_1,j_1}(u_1)\cdots B^n_{i_n,j_n}(u_n)\\
&\quad = \sum_{i_0=0}^{j_0}\sum_{i_1=0}^{j_1}\cdots\sum_{i_{n+1}=0}^{j_{n+1}} P_{i_0,i_1,\ldots,i_n} B^0_{i_0,j_0}(u_0)B^1_{i_1,j_1}(u_1)\cdots B^{n+1}_{i_{n+1},j_{n+1}}(u_{n+1})\\
&\quad = F(u_0,u_1,\ldots,u_{n+1})
\end{aligned}
$$

The original slice introduced k dependencies into $n+1$ dimensions, leaving a subdomain of dimension $n-k+1$. After raising the dimension of the tensor product by one, the parameter space of the slice will also have increased in dimension by one, and we will be left with a formula which implicitly defines the $n+1$-dimensional control vertices of the new tensor product in terms of the control vertices of the original tensor product.

Let $L_{i_0,i_1,\ldots,i_{n+1}}$ (where $i_t = 0,\ldots,j_t$, $t = 0,\ldots,n+1$) be the control vertices of the new $n+2$-dimensional tensor product. Then

$$
\sum_{i_{n+1}=0}^{j_{n+1}} P_{i_0,i_1,\ldots,i_n} B^{n+1}_{i_{n+1},j_{n+1}}(u_{n+1}) = \sum_{i_{n+1}=0}^{j_{n+1}} L_{i_0,i_1,\ldots,i_{n+1}} B^{n+1}_{i_{n+1},j_{n+1}}(u_{n+1})
$$

which implies that

$$
P_{i_0,i_1,\ldots,i_n} = \sum_{i_{n+1}=0}^{j_{n+1}} L_{i_0,i_1,\ldots,i_{n+1}} B^{n+1}_{i_{n+1},j_{n+1}}(u_{n+1})
$$

These equations have the solution $L_{i_0,i_1,\ldots,i_{n+1}} = P_{i_0,i_1,\ldots,i_n}$, for $i_{n+1} = 0$ to j_{n+1}, because blending functions sum to one. If the blending functions form a basis, then this is the only solution.

5. Blending-Function Specific Inheritance

We will consider a few blending-function specific examples of inheritance: For Bézier tensor products: recursive evaluation, subdivision, degree elevation and blossoming; for B-splines: recursive evaluation, knot insertion and blossoming.

5.1. The de Casteljau Algorithm

Recursive evaluation is an inherited property. The *de Casteljau Algorithm*, [Farin88 p. 27] can be used directly on any Bézier tensor product. The following pseudo-code illustrates the algorithm for a degree j Bézier curve with control points P_0 through P_j at parameter u.

```
decasteljau ( P_0,...,P_j, u ) := {
    for i ← 0,...,j {
        P_i^0 ← P_i
    }
    for k ← 1,...,j {
        for i ← 0,...,j-k {
            P_i^k ← (1-u)P_i^{k-1} + uP_{i+1}^{k-1}
        }
    }
    return (P_0^j)
}
```

The value returned is the point on the curve at parameter u. In the multi-dimensional case, we treat the $n+1$-dimensional tensor product as a 'curve' over n-dimensional tensor products and apply the de Casteljau algorithm to this curve.

Let $\mathbf{P}_k = \sum_{i_1=0}^{j_1} \cdots \sum_{i_n=0}^{j_n} P_{k,i_1,\ldots,i_n} B_{i_1,j_1}(u_1) \cdots B_{i_n,j_n}(u_n)$ for $k = 0,\ldots,j_0$. Then one stage of the algorithm appears as

$$
\begin{aligned}
&(1-u)\,\mathbf{P}_k + u\,\mathbf{P}_{k+1} \\
&= (1-u) \sum_{i_1=0}^{j_1} \cdots \sum_{i_n=0}^{j_n} P_{k,i_1,\ldots,i_n} B_{i_1,j_1}(u_1) \cdots B_{i_n,j_n}(u_n) + u \sum_{i_1=0}^{j_1} \cdots \sum_{i_n=0}^{j_n} P_{k+1,i_1,\ldots,i_n} B_{i_1,j_1}(u_1) \cdots B_{i_n,j_n}(u_n) \\
&= \sum_{i_1=0}^{j_1} \cdots \sum_{i_n=0}^{j_n} \left((1-u)\,P_{k,i_1,\ldots,i_n} + u\,P_{k+1,i_1,\ldots,i_n} \right) B_{i_1,j_1}(u_1) \cdots B_{i_n,j_n}(u_n)
\end{aligned}
$$

Thus the de Casteljau algorithm can be applied dimension-by-dimension to the original control vertices.

The value computed is independent of the order in which dimensions are processed. However, the total number of affine combinations required by each different ordering of dimensions is not necessarily the same. We will show how to determine the most cost efficient ordering in Section 5.5.

5.2. Bézier Subdivision

Subdivision is also an inherited property. For Bézier tensor products, subdivision is a procedure for finding new Bézier control vertices for rectilinear portions of the original tensor product. In the course of finding a point on a curve using the de Casteljau algorithm, the intermediate points $P_0^0\ P_0^1\ \cdots\ P_0^j$ and $P_0^j\ P_1^{j-1}\ \cdots\ P_j^0$ are generated. These points are the Bézier control vertices for the portions of the original curve from $[0,u]$ and $[u,1]$, respectively. That is, $\sum_{i=0}^{j} P_0^i\ B_{i,j}(uv)$, for $v \in [0,1]$, is the same curve as $\sum_{i=0}^{j} P_i\ B_{i,j}(v)$, for $v \in [0,u]$, and $\sum_{i=0}^{j} P_i^{j-i}\ B_{i,j}(\,(1-v)u+v\,)$, for $v \in [0,1]$, is the same curve as $\sum_{i=0}^{j} P_i\ B_{i,j}(v)$, for $v \in [u,1]$ [Lane80].

This technique extends easily to multi-dimensional tensor products. The same algorithm can be used to subdivide Bézier tensor product slices, since the algorithm depends only on the values of the control vertices of the original tensor product.

5.3. Bézier Degree Raising

The inheritance of degree raising is only slightly less immediate. The degree of a Bézier tensor product can be increased one dimension at a time [Forrest72]. Consider raising the degree in dimension l, of

$$\sum_{i_0=0}^{j_0}\sum_{i_1=0}^{j_1}\cdots\sum_{i_n=0}^{j_n} P_{i_0,i_1,\ldots,i_n}\,B_{i_0,j_0}(u_0)B_{i_1,j_1}(u_1)\cdots B_{i_n,j_n}(u_n)$$

Rewriting so that the sum in the l th dimension is outermost, we have

$$\sum_{i_l=0}^{j_l}\mathbf{P}_{i_l}\,B_{i_l,j_l}(u_l)=\sum_{i_l=0}^{j_l+1}\mathbf{L}_{i_l}\,B_{i_l,j_l+1}(u_l)$$

where the $\mathbf{P}_{i_l}$'s represent summation in the other n dimensions. The L's and the P's are related as follows:

$$\mathbf{L}_0=\mathbf{P}_0$$

$$\mathbf{L}_i=\left(\frac{i}{j_l+1}\right)\mathbf{P}_{i-1}+\left(1-\frac{i}{j_l+1}\right)\mathbf{P}_i \text{ for } i=1,\ldots,j_l$$

$$\mathbf{L}_{j_l+1}=\mathbf{P}_{j_l}$$

This yields

$$L_{i_0,i_1,\ldots,i_l=0,\ldots,i_n}=P_{i_0,i_1,\ldots,i_l=0,\ldots,i_n}$$

$$L_{i_0,i_1,\ldots,i,\ldots,i_n}=\left(\frac{i}{j_l+1}\right)P_{i_0,i_1,\ldots,i-1,\ldots,i_n}+\left(1-\frac{i}{j_l+1}\right)P_{i_0,i_1,\ldots,i,\ldots,i_n} \text{ for } i=1,\ldots,j_l$$

$$L_{i_0,i_1,\ldots,j_l+1,\ldots,i_n}=P_{i_0,i_1,\ldots,j_l,\ldots,i_n}$$

The subsequent restriction to an $n-k$-dimensional slice results in

$$\sum_{i_0=0}^{j_0}\sum_{i_1=0}^{j_1}\cdots\sum_{i_l=0}^{j_l+1}\cdots\sum_{i_n=0}^{j_n} L_{i_0,i_1,\ldots,i_n}\,B_{i_0,j_0}(u_0)B_{i_1,j_1}(u_1)\cdots B_{i_l,j_l+1}(u_l)\cdots B_{i_n,j_n}(u_n)$$

where $u_m=\gamma_m(u_0,\ldots,u_{n-k})$, for $m=n-k+1,\ldots,n$.

5.4. Blossoming a Bézier Tensor Product Slice

Blossoms were introduced by Ramshaw [Ramshaw87]. We review his terminology briefly and extend it to suit our needs.

Definition: Let $d_i>0$ for $i=0,\ldots,n$, and let $\boldsymbol{D}=U_0^{d_0}\times U_1^{d_1}\times\cdots\times U_n^{d_n}$, where $U_i^{d_i}$ is $U_i\times U_i\times\cdots\times U_i$ d_i times, and U_i is some real interval $[a_i,b_i]$. Let $F:\boldsymbol{D}\rightarrow\mathbf{R}$, and let $(u_{0,0},u_{0,1},\ldots,u_{0,d_0-1},u_{1,0},u_{1,1},\ldots,u_{1,d_1-1},\ldots,u_{n,0},u_{n,1},\ldots,u_{n,d_n-1})\in\boldsymbol{D}$. Then F is *symmetric in dimension i*, if, for any permutation $\sigma:\{0,\ldots,d_i-1\}\rightarrow\{0,\ldots,d_i-1\}$,

$$F(u_{0,0},\ldots,u_{i,0},u_{i,1},\ldots,u_{i,d_i-1},\ldots,u_{n,d_n-1})=F(u_{0,0},\ldots,u_{i,\sigma(0)},u_{i,\sigma(1)},\ldots,u_{i,\sigma(d_i-1)},\ldots,u_{n,d_n-1})\,.$$

$F : D \rightarrow \mathbf{R}$ is *dimensionally symmetric* if it is symmetric in every dimension.

Let j_l be a set of positive integers, for $l \in \{0,...,n\}$, and let i_l be an integer in the range $0,...,j_l$. Furthermore, let $B_{i_l,j_l}(u_l)$ be polynomials of degree j_l in u_l, and let $F(u_0,u_1,...,u_n)$ be a tensor product over these blending functions, and over a set of control vertices $P_{i_0,i_1,...,i_n}$ from $\mathbf{R}^m$, for $i_l = 0,...,j_l$.

Definition: The *blossom* of F is the unique polynomial

$$f_F(u_{0,0},...,u_{0,j_0-1}\ ;\ \cdots\ ;\ u_{n,0},...,u_{n,j_n-1})$$

which is

- affine in each parameter u_{i_l,j_l},
- symmetric in each dimension l; that is, for each fixed $l \in \{0,...,n\}$, f_F is symmetric over the collection of parameters $u_{l,i}$ whose first index is l, and
- F is the *diagonal* of f_F; that is:

$$F(u_0,...,u_n) = f_F(u_0,...,u_0\ ;\ \cdots\ ;\ u_n,...,u_n)$$

(semi-colons are used to separate variables in different dimensions).

A tensor product is transformed by the blossoming operation into a multi-affine polynomial which is "essentially equivalent" to the original polynomial [Ramshaw88 p. 3]. Ramshaw demonstrates that the blossom of a tensor product (over polynomial blending functions) exists and is unique, and gives methods for finding the blossom explicitly.

Given the blossom of a tensor product, it is easy to determine the Bézier control vertices of the tensor product. Let $F(u_0,...,u_n)$ be an n-dimensional Bézier tensor product over the domain $[0,1]^{n+1}$ with control vertices $P_{i_0,i_1,...,i_n}$, and having degree j_l in dimension l. Then

$$f_F(1,1,...,1,0,0,...,0\ ;\ \cdots\ ;\ 1,1,...,1,0,0,...,0) = P_{i_0,i_1,...,i_n}$$

when, in dimension l of the blossom, the number of variables which have been set to one is i_l and the number set to zero is $j_l - i_l$ [Ramshaw88]. (Because of dimensional symmetry, the actual order in which ones and zeros are substituted into the parameters of the blossom for a given dimension is immaterial.)

A slice of a tensor product F is simply F constrained by a set of linear restrictions

$$u_m = \gamma_m(u_0,...,u_{n-k}) = c_{m,0}u_0 + \cdots + c_{m,n-k}u_{n-k} + c_{m,n-k+1} \quad \text{for } m = n-k+1,...,n\ .$$

In order to simplify upcoming discussions, we will adopt the convention that when $c_{m,i} = 0$, the variable u_i will not be given as a parameter of γ_m.

Let F_C be the polynomial which results when substituting the constraints of the slice into the tensor product F. There must be a blossom corresponding to it, which we will call f_{F_C}. We want to find f_{F_C} in terms of the blossom f_F and the set of constraints C.

Let h_l be the maximum degree of F_C in dimension l (where $l \in \{0,...,n-k\}$). Let u_l be the variable for dimension l. The degree of F in dimension l was j_l. If u_m, where $m \in \{n-k+1,...,n\}$, is constrained to depend on u_l by an equation in C, then the degree of u_l in F_C is the degree of u_l in F, plus the degree of each u_m that depends on u_l. By examining γ_m, we know that u_m depends on u_l when $c_{m,l} \neq 0$. We can state this formally by defining a function ε as follows:

$$\varepsilon(m,l)=\begin{cases}1 \text{ when } c_{m,l}\neq 0\\ 0 \text{ otherwise}\end{cases}$$

Then the maximum degree of u_l in F_C is

$$h_l = j_l + \sum_{m=n-k+1}^{n} j_m \varepsilon(m,l)$$

The blossom of the slice must have the form

$$f_{F_C}(u_{0,0}\ldots u_{0,h_0-1};\cdots;u_{n-k,0}\ldots u_{n-k,h_{n-k}-1}) \tag{1}$$

where f_{F_C} is dimensionally symmetric, multi-affine, and the diagonal of f_{F_C} is F_C.

We create a candidate for the blossom of F_C by substituting the constraint equations into the blossom of F:

$$f_F(<0>;<1>;\ldots;<n-k>;<\gamma_{n-k+1}>;<\gamma_{n-k+2}>;\ldots;<\gamma_n>)$$

where $<l>$ stands for the parameters of the independent dimension l, and $<\gamma_m>$ stands for the substitution of one instance of the expression for γ_m for each of the j_m parameters f_F has for dimension m.

There are precisely as many parameters of the independent dimensions and all the instances of the constraint equations substituted into the blossom of F, as there are variables in Equation 1. Let $l=0,\ldots,n-k$, $m=n-k+1,\ldots,n$, and $i=0,\ldots,j_m-1$, and let H be a function which computes index values defined as follows:

$$H(m,l,i)=i+j_l+\sum_{t=n-k+1}^{m-1}\varepsilon(t,l)j_t$$

(The summation is zero if $n-k+1>m-1$.)

The indices given above assign unique variables of f_{F_C} to each parameter of the modified blossom of F, and H provides precisely these indices. Furthermore, the greatest index H provides for dimension l is h_l-1. Therefore, H gives us unique subscripts for each independent dimension l.

Consider

$$g(u_{0,0},\ldots,u_{0,h_0-1};\cdots;u_{n-k,0},\ldots,u_{n-k,h_{n-k}-1})=$$

$$f_F\Big(u_{0,0},\ldots,u_{0,j_0-1};\cdots;u_{n-k,0},\ldots,u_{n-k,j_{n-k}-1};$$

$$\gamma_{n-k+1}(u_{0,H(n-k+1,0,0)},\ldots,u_{n-k,H(n-k+1,n-k,0)}),\ldots,\gamma_{n-k+1}(u_{0,H(n-k+1,0,j_{n-k}-1)},\ldots,u_{n-k,H(n-k+1,n-k,j_{n-k}-1)});$$

$$\cdots;\gamma_n(u_{0,H(n,0,0)},\ldots,u_{n-k,H(n,n-k,0)}),\ldots,\gamma_n(u_{0,H(n,0,j_n-1)},\ldots,u_{n-k,H(n,n-k,j_n-1)})\Big)$$

where $u_{l,i}$ appears as a parameter of γ_m only if u_m depends on u_l in F_C. The function γ_m is substituted once into every variable of dimension m, that is, j_m times. Each substituted γ_m has different variables from every other γ_m, and from any γ_p when $m\neq p$, and $m,p\in\{n-k+1,\ldots,n\}$. This follows from the construction of the function H.

The function g is affine in each of its variables $u_{i,j}$, because f_F is multi-affine and each γ_m is multi-affine. However, g is not dimensionally symmetric. Let $\sigma_{n+1}:\{0,\ldots,n\}\rightarrow\{0,\ldots,n\}$ denote a permutation of the first $n+1$ natural numbers. To introduce dimensional symmetry we form:

$$\frac{\sum_{\forall\sigma_{h_0}}\sum_{\forall\sigma_{h_1}}\cdots\sum_{\forall\sigma_{h_{n-k}}} g\left(u_{0,\sigma_{h_0}(0)},\ldots,u_{0,\sigma_{h_0}(h_0-1)};\cdots;u_{n-k,\sigma_{h_{n-k}}(0)},\ldots,u_{n-k,\sigma_{h_{n-k}}(h_{n-k}-1)}\right)}{h_0!h_1!\cdots h_{n-k}!}$$

This expression is dimensionally symmetric; this follows as a direct result of the way we constructed the summation. Because g is multi-affine, this expression is multi-affine. Moreover, its diagonal is F_C. We can see this by substituting u_i for each $u_{i,j}$ where $j = 0,\ldots,h_i-1$. The above expression reduces to

$$f_F\Big(u_0,\ldots,u_0;\cdots;u_{n-k},\ldots,u_{n-k};$$

$$\gamma_{n-k+1}(u_0,u_1,\ldots,u_{n-k}),\ldots,\gamma_{n-k+1}(u_0,u_1,\ldots,u_{n-k});\cdots;\gamma_n(u_0,u_1,\ldots,u_{n-k}),\ldots,\gamma_n(u_0,u_1,\ldots,u_{n-k})\Big)$$

But because the diagonal of f_F is just F, this must be

$$F\left(u_0,u_1,\ldots,u_{n-k},\gamma_{n-k+1}(u_0,u_1,\ldots,u_{n-k}),\ldots,\gamma_n(u_0,u_1,\ldots,u_{n-k})\right)$$

which is just F evaluated at a point on the slice. Therefore F_C is the diagonal of the above expression.

Finally, the degree of F_C in u_l is h_l, which matches precisely the number of variables in dimension l, for each $l \in 0,\ldots,n-k$. Therefore, this expression must be f_{F_C}, the blossom of F_C.

Notice that g already has *some* dimensional symmetry, but we did not take advantage of this symmetry when forming the blossom of F_C.

The same blossoming procedure can be performed for polynomial slices instead of linear slices. In this case, we use the blossom of each γ_m, instead of γ_m, in the construction, because a general polynomial γ_m is not multi-affine, whereas the blossom of γ_m is.

Example 3: Consider a 4-dimensional Bézier tensor product $F(u,v,w,x)$, linear in u,v and x and quadratic in w, with the blossom $f(u_0;v_0;w_0,w_1;x_0)$, sliced by the constraints $w = 3+u$ and $x = 2-v$. Then the blossom of the slice will have the form:

$$f_{F_C}(u_0,u_1,u_2;v_0,v_1)$$

$$= \frac{1}{3!2!}\Big(2f(u_0;v_0;(3+u_1),(3+u_2);(2-v_1)) + 2f(u_1;v_0;(3+u_2),(3+u_0);(2-v_1))$$

$$+ 2f(u_2;v_0;(3+u_0),(3+u_1);(2-v_1)) + 2f(u_0;v_1;(3+u_1),(3+u_2);(2-v_0))$$

$$+ 2f(u_1;v_1;(3+u_2),(3+u_0);(2-v_0)) + 2f(u_2;v_1;(3+u_0),(3+u_1);(2-v_0))\Big)$$

To find the value of the control vertex $f_{F_C}(0,1,1;0,1)$ we compute:

$$\frac{1}{6}\Big(f(0;0;4,4;1) + f(0;1;4,4;2) + 2f(1;0;3,4;1) + 2f(1;1;3,4;2)\Big)$$

Each specific invocation of the blossom of F can be written exclusively in terms of the Bézier control vertices of F, by making use of the multi-affine and symmetry properties of the blossom. For example,

$$f(0;0;4,4;1) = f(0;0;\ 4\times1 + (-3)\times0\ ,4;1)$$

$$= 4f(0;0;1,4;1) - 3f(0;0;0,4;1)$$

$$= 4(4f(0;0;1,1;1) - 3f(0;0;1,0;1)) - 3(4f(0;0;1,0;1) - 3f(0;0;0,0;1))$$

$$= 16f(0;0;1,1;1) - 24f(0;0;1,0;1) + 9f(0;0;0,0;1)$$

We can solve for each of the other instances of the blossom of F in terms of control vertices of F and find that

$$f_{F_C}(0,1,1;0,1) = \frac{1}{6}\Big(9f(0;0;0,0;1) - 24f(0;0;1,0;1) + 16f(0;0;1,1;1) - 9f(0;1;0,0;0)$$
$$+ 18f(0;1;0,0;1) + 24f(0;1;1,0;0) - 48f(0;1;1,0;1) - 16f(0;1;1,1;0)$$
$$+ 32f(0;1;1,1;1) + 12f(1;0;0,0;1) - 34f(1;0;1,0;1) + 24f(1;0;1,1;1)$$
$$- 12f(1;1;0,0;0) + 24f(1;1;0,0;1) + 34f(1;1;1,0;0) - 68f(1;1;1,0;1)$$
$$- 24f(1;1;1,1;0) + 48f(1;1;1,1;1)\Big)$$

Example 3 illustrates that an explicit expression for the blossom of a polynomial function is not required in order to evaluate it. [Ramshaw88a] shows that, in general, all points on a blossom are computable by forming affine combinations of control vertices.

Example 4: Consider the tensor product $F(u,v) = \sum_{i=0}^{1}\sum_{j=0}^{1} P_{i,j}\, B_{i,1}(u)B_{j,1}(v)$, where $B_{i,1}(u)$ and $B_{j,1}(v)$ are the linear Bernstein polynomials. Let $v = \gamma(u) = 1-u$. This results in the 1-dimensional slice (a quadratic curve), given by

$$P_{0,0}(1-u)(1-v) + P_{0,1}(1-u)v + P_{1,0}u(1-v) + P_{1,1}\,u\,v$$
$$= P_{0,0}u(1-u) + P_{0,1}(1-u)^2 + P_{1,0}u^2 + P_{1,1}u(1-u)$$
$$= P_{0,1}(1-u)^2 + (P_{0,0}+P_{1,1})u(1-u) + P_{1,0}u^2$$

This slice can be re-represented in Bézier form, using the quadratic Bernstein polynomials, as $\sum_{i=0}^{2} L_i\, B_{i,2}(u)$, where $L_0 = P_{0,1}$, $L_1 = \frac{P_{0,0}+P_{1,1}}{2}$, and $L_2 = P_{1,0}$.

Now examine the slice of F using blossoming. Note that since F is bi-linear, its blossom f will be 2-dimensional, having a group of variables for each dimension. Since the original tensor product was linear in each dimension, its blossom will have only one variable for each dimension, and, by definition, $F(u,v) = f_F(u;v)$.

Since the slice constrains the second dimension of the tensor product in terms of the first, the slice is one dimensional, but is of degree two. Therefore, the blossom of the slice will be of the form $f_{F_C}(u_0,u_1)$ and be symmetric over both parameters (unlike the blossom of the original tensor product), giving:

$$f_{F_C}(u_0,u_1) = \frac{1}{2}\Big[f_F(u_0;\gamma(u_1)) + f_F(u_1;\gamma(u_0))\Big]$$

Since the slice blossom is the blossom of an equivalent ordinary tensor product, the control vertices of the equivalent tensor product must be $f_{F_C}(0,0)$, $f_{F_C}(1,0)$ and $f_{F_C}(1,1)$. Performing the appropriate substitutions, we obtain:

$$L_0 = f_{F_C}(0,0) = \frac{1}{2}\left[f_F(0;\gamma(0)) + f_F(0;\gamma(0)) \right] = f_F(0;1) = P_{0,1}$$

$$L_1 = f_{F_C}(1,0) = \frac{1}{2}\left[f_F(1;\gamma(0)) + f_F(0;\gamma(1)) \right] = \frac{1}{2}\left(P_{1,1} + P_{0,0} \right)$$

$$L_2 = f_{F_C}(1,1) = \frac{1}{2}\left[f_F(1;\gamma(1)) + f_F(1;\gamma(1)) \right] = f_F(1;0) = P_{1,0}$$

5.5. Evaluation of a Bézier Tensor Product Slice

We will consider the computational cost of evaluating the tensor product slice as it stands against that of evaluating its equivalent form with the constraints already substituted and assimilated. The two costs are not, in general, the same. Moreover, for any collection of degree and dimension, the evaluation cost is sometimes more in the tensor product form and sometimes more in the equivalent form.

For example, in two dimensions, using cubic polynomials and constraining one dimension in terms of the other (giving a curve), the equivalent tensor product requires 21 affine combinations to compute a curve point, while the original tensor product requires 30. Whereas in three dimensions, using cubic polynomials and constraining one dimension in terms of the other two (giving a surface), the equivalent tensor product requires 168 affine combinations to compute a surface point, while the original tensor product requires 126. However, given a chosen form, there is always an optimal ordering of dimensions.

Consider a cost function $\Delta: \mathbf{Z}^+ \rightarrow \mathbf{Z}^+$, where $\Delta(d)$ represents the number of affine combinations required to compute the value of a point on a curve of degree d, and where $\Delta(d)$ grows quickly enough, that is $\frac{\Delta(d+k)}{\Delta(d)} \geq \frac{d+k}{d}$ for $k>0$. Note that $\frac{\Delta(t+k)}{\Delta(t)} \geq \frac{t+k}{t} \Rightarrow (t+k)\Delta(t) - t\Delta(t+k) \leq 0$, for $k>0$. Also observe that the de Casteljau cost function has this property:

$$\frac{\Delta(d+k)}{\Delta(d)} = \frac{(d+k)(d+k+1)}{d(d+1)} = \frac{(d+k)}{d}\,\frac{(d+k+1)}{(d+1)} \geq \frac{(d+k)}{d}$$

Given that the dimension ordering is $n, n-1, \ldots, 1, 0$ and given that the maximum degree in dimension k of a blending function is j_k, the cost of finding a surface point is:

$$\Delta(j_0) + \Delta(j_1)(j_0+1) + \Delta(j_2)(j_1+1)(j_0+1) + \cdots + \Delta(j_n)(j_{n-1}+1)\ldots(j_1+1)(j_0+1)$$

We shall show that this expression will be minimized when 0 is the dimension of highest degree, 1 the dimension of the next highest degree, and so on.

Define $\mu_n : \mathbf{Z}^{+n} \rightarrow \mathbf{Z}^+$ as follows:

$$\mu_1(x) = \Delta(x)$$

$$\mu_{n+1}(x_0, \ldots, x_n) = \Delta(x_0) + \Delta(x_1)(x_0+1) + \Delta(x_2)(x_1+1)(x_0+1) + \cdots +$$

$$\Delta(x_n)(x_{n-1}+1)...(x_0+1)$$

Then, for $n \geq 1$

$$\mu_{n+1}(x_0,...,x_n) = \Delta(x_0) + (x_0+1)\mu_n(x_1,...,x_n)$$

Furthermore

$$\mu_{n+1}(x_0,...,x_n) = \mu_n(x_0,...,x_{n-1}) + \Delta(x_n)(x_{n-1}+1)...(x_0+1)$$

Lemma: Let $\sigma : \{0,...,n\} \rightarrow \{0,...,n\}$ be some permutation of the first $n+1$ integers and let $x_0,...,x_n$ be a sequence of positive integers given in non-increasing order ($x_i \geq x_j$, when $i < j$). Then

$$\mu_{n+1}(x_0,...,x_n) \leq \mu_{n+1}(x_{\sigma(0)},...,x_{\sigma(n)})$$

That is, the value of μ_n is minimized when its arguments are given in non-increasing order.

Proof: We use induction on n.

n = 2: Let $a \leq b$. We want to show that $\mu_2(b,a) \leq \mu_2(a,b)$. If $a = b$, then this is certainly true. Otherwise, $a < b$. Then

$$\mu_2(b,a) - \mu_2(a,b) = \Delta(b)+\Delta(a)(b+1) - \Delta(a)-\Delta(b)(a+1)$$

$$= b\,\Delta(a)-a\,\Delta(b)$$

But we were given as a property of Δ that when $a < b$, this difference is non-positive, implying that $\mu_2(b,a) \leq \mu_2(a,b)$. Therefore, when $a \leq b$, $\mu_2(b,a) \leq \mu_2(a,b)$.

n+1, given n: Let τ be some permutation of the first $n+1$ integers such that, for any other permutation σ of the first $n+1$ integers,

$$\mu_{n+1}(x_{\tau(0)},...,x_{\tau(n)}) \leq \mu_{n+1}(x_{\sigma(0)},...,x_{\sigma(n)})$$

That is, τ is a permutation which minimizes μ_{n+1} over all different orderings of the arguments $x_0,...,x_n$. Recall that

$$\mu_{n+1}(x_{\tau(0)},...,x_{\tau(n)}) = \Delta(x_{\tau(0)}) + (x_{\tau(0)}+1)\mu_n(x_{\tau(1)},...,x_{\tau(n)})$$

Since τ is an ordering of arguments which minimizes the value of μ_{n+1}, $\mu_n(x_{\tau(1)},...,x_{\tau(n)})$ must also be minimal. By the induction hypothesis, the sequence $x_{\tau(1)},...,x_{\tau(n)}$ must be in non-increasing order.

Also recall that

$$\mu_{n+1}(x_{\tau(0)},...,x_{\tau(n)}) = \mu_n(x_{\tau(0)},...,x_{\tau(n-1)}) + \Delta(x_{\tau(n)})(x_{\tau(n-1)}+1)...(x_{\tau(0)}+1)$$

Then, $\mu_n(x_{\tau(0)},...,x_{\tau(n-1)})$ must be minimal. By the induction hypothesis, the sequence $x_{\tau(0)},...,x_{\tau(n-1)}$ must be in non-increasing order.

The fact that these two sequences are in non-increasing order implies that the sequence $x_{\tau(0)},...,x_{\tau(n)}$ is also in non-increasing order. **Q.E.D.**

The minimum number of affine combinations required for the de Casteljau algorithm (in dimension-by-dimension evaluation) is $\mu_{n+1}(j_{\sigma(0)},...,j_{\sigma(n)})$, where the permutation σ sorts the sequence of j_l's into non-increasing order. Knowing this fact, and applying the definition of μ_n, we can compare the minimal cost for the tensor product and for the equivalent tensor product and use the least expensive version for evaluating points on the surface.

5.6. The de Boor Algorithm

The *de Boor* algorithm [Boor87] is an evaluation scheme for B-splines, which is very similar to the de Casteljau algorithm. The de Boor algorithm has the same form as the de Casteljau algorithm, but uses different multipliers in each intermediate point calculation.

This algorithm can be applied to B-spline tensor products in precisely the same dimension-by-dimension fashion as the de Casteljau algorithm is applied to Bézier tensor products. It is also inherited in the same fashion from B-spline tensor products by B-spline tensor product slices.

Since the algorithm is structurally identical to the de Casteljau algorithm, the cost analysis given for that algorithm can be applied to B-spline evaluation as well.

5.7. B-spline Knot Insertion

The *Oslo Algorithm* and *Böhm's Algorithm* for knot insertion are described in [Cohen80, Prautzsch84, Böhm80, Goldman89, Bartels87]. Both the Oslo algorithm and Böhm's algorithm can be extended to multi-dimensional B-splines tensor products in the same manner as the de Boor algorithm.

A tensor product that has had knots inserted represents the same mapping from domain to range as the original tensor product. Therefore, a slice of a knot-inserted tensor product will represent the same set of points as the same slice of the original tensor product. Thus knot insertion is a valid operation for B-spline tensor product slices.

5.8. Blossoming a B-spline Tensor Product Slice

We have already developed at the blending function level most of the results we require for blossoming B-spline slices. A B-spline tensor product is made up of numerous blending functions domains. We can associate a blossom with each such domain. We already know how to evaluate these blossoms if we are given the Bézier control vertices. [Ramshaw88a] shows that the blossom can also be determined from the B-spline control vertices for the blending function domain and that B-spline control vertices may be determined from a blossom by evaluating it at sets of adjacent knots.

The blossom of a B-spline tensor product slice will also exist, in the same piecewise fashion that the blossom of the original tensor product existed. The Bézier blossoming development tells us what the blossom of the slice is over each blending function domain in terms of the blossom of the B-spline and the constraints over that domain.

6. Conclusions and Open Questions

It is not necessary to abandon the tensor product for surface representation in order to obtain useful multi-sided patches. We have shown that tensor product slices can be used to create multi-sided surface patches with arbitrary numbers of sides. The definition of a tensor product slice is firmly rooted in the concept of a multi-dimensional tensor product. Therefore, tensor product slices inherit most of the useful properties of general tensor products.

Many important properties of specific tensor product schemes are also inherited in the same fashion. Bézier tensor product slices can use the same evaluation, subdivision and degree-elevation algorithms that ordinary Bézier tensor products use. The notion of blossoming also gave us a method for computing an alternate representation for Bézier tensor product slices, which can reduce the computational costs of evaluation.

B-spline tensor product slices also inherit the same evaluation and knot insertion algorithms available to ordinary B-spline tensor products. Blossoming, too, makes sense for B-spline tensor product slices.

There remain many avenues for further exploration. Though we have not tried to do so, it is conceivable that tensor product slices could be used for visualizing 3-dimensional data, by looking at cross sections. The conditions under which a 2-dimensional tensor product slice can be constructed to fill an n-sided opening in a surface, while maintaining some degree of continuity, are not known. Even if such conditions were known, a method for constructing the desired patch is still required. The effects of introducing dependencies among blending functions upon the ease of manipulation of tensor product slices have yet to be explored. Lastly, the properties of tensor products sliced by more general constraint equations have not been investigated here.

7. References

Bartels87

R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, Palo Alto, California (1987).

Boor87

C. de Boor and K. Höllig, "B-splines without Divided Differences," in *Geometric Modeling: Algorithms and New Trends*, ed. G. Farin, SIAM Press, Philadelphia, Pennsylvania (1987).

Böhm80

W. Böhm, "Inserting New Knots into B-spline Curves," *Computer-Aided Design* **12**(4)(1980).

Böhm84

W. Böhm, G. Farin, and J. Kahmann, "A Survey of Curve and Surface Methods in CAGD," *Computer Aided Geometric Design* **1**(1)(1984).

Böhm84a

W. Böhm, "Calculating with Box Splines," *Computer Aided Geometric Design* **1**(2)(1984).

Charrot84

P. Charrot and J. A. Gregory, "A Pentagonal Surface Patch for Computer Aided Geometric Design," *Computer Aided Geometric Design* **1**(2)(1984).

Cohen80

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* **14**(2)(1980).

Dahmen83

W. Dahmen and C. A. Micchelli, "Multivariate Splines — A New Constructive Approach," in *Surfaces in CAGD*, ed. R. E. Barnhill and W. Böhm, North-Holland, Amsterdam (1983).

Dahmen84

W. Dahmen and C. Micchelli, "Subdivision Algorithms for the Generation of Box Spline Surfaces," *Computer Aided Geometric Design* **1**(2)(1984).

DeRose89

T. D. DeRose and C. T. Loop, "S-patches: A Class of Representations for Multi-Sided Surface Patches," *ACM Transactions on Graphics* **8**(3)(1989).

Farin88

G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, Boston (1988).

Forrest72

A. R. Forrest, "Interactive Interpolation and Approximation by Bézier Polynomials," *Computer Journal* **15**(1)(1972).

Goldman89
R. N. Goldman, *Blossoming and Knot Insertion Algorithms for B-spline Curves.* SUBMITTED FOR PUBLICATION.

Hosaka84
M. Hosaka and F. Kimura, "Non-four-sided Patch Expressions with Control Points," *Computer Aided Geometric Design* 1(1)(1984).

Lane80
J. M. Lane and R. F. Riesenfeld, "A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces," *IEEE Transactions on Pattern Matching and Machine Intelligence* PAMI-2(1)(1980).

Prautzsch84
H. Prautzsch, "A Short Proof of the Oslo Algorithm," *Computer Aided Geometric Design* 1(1)(1984).

Ramshaw87
L. Ramshaw, *Blossoming: A Connect-the-Dots Approach to Splines, Systems Research Technical Report #19,* Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, California, 94301, USA (1987).

Ramshaw88
L. Ramshaw, *Béziers and B-splines as Multiaffine Maps,* Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, California, 94301, USA (1988).

Ramshaw88a
L. Ramshaw, *Blossoms are Polar Forms, Systems Research Technical Report #34,* Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, California, 94301, USA (1989).