Using A-patches to Tessellate Algebraic Curves and Surfaces

Technical Report CS-2009-21

Stephen Mann
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA N2L 3G1
smann@uwaterloo.ca

June 10, 2009

1 Abstract

This technical report expands on the A-patch tessellation work in the masters thesis of Curtis Luk [Luk08], improving on many of the techniques in his work and extending the A-patch constraints to allow a wider class of single sheeted Bernstein representations.

2 Introduction

An *implicit function* is defined to be the zero set of a scalar function F over space. Further, for points not on the implicit function, the sign of this scalar function tells us whether the point lies inside or outside the implicit function. The choice of whether positive values are inside or outside is arbitrary, since we may multiply an implicit function by -1 and change the notion of inside and outside.

An *algebraic* function is an implicit function that is a polynomial. The following example illustrates these ideas. Consider

$$F(P) = x^2 + y^2 - 1.$$

F represents an algebraic curve (a circle; Figure 1) in the plane, where P lies in the plane and (x, y) are the coordinates of this point relative to some basis. Further, making the arbitrary choice that positive values are outside the circle, we have

$$F(P) = \begin{cases} = 0 & P \text{ is on the curve/surface.} \\ < 0 & P \text{ is inside the curve/surface.} \\ > 0 & P \text{ is outside the curve/surface.} \end{cases}$$

If F were in 3-space, then the algebraic would represent a cylinder. In this paper, I will be concerned only with curves in 2-space and surfaces in 3-space.

Implicit and algebraic functions provide a nice inside-outside test, which usually also gives the distance to the curve/surface. There are also some nice modeling paradigms using implicit functions. However, one weakness of algebraics and implicits is that it is hard to tessellate them, i.e., it is hard to compute a piecewise linear approximation to the curve or surface.

Most approaches for tessellating implicit functions are variations of *marching cubes* [LC87, NY06], where the implicit function is sampled at a grid of locations. If the sign at adjacent grid cells differs, then the

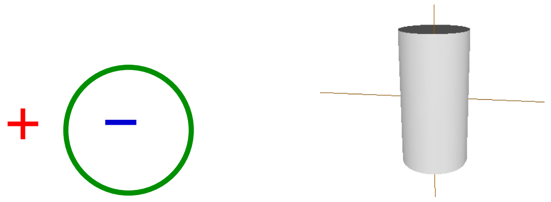

Figure 1: The algebraic $x^2 + y^2 - 1 = 0$ represents a circle in the plane and a cylinder in 3-space.

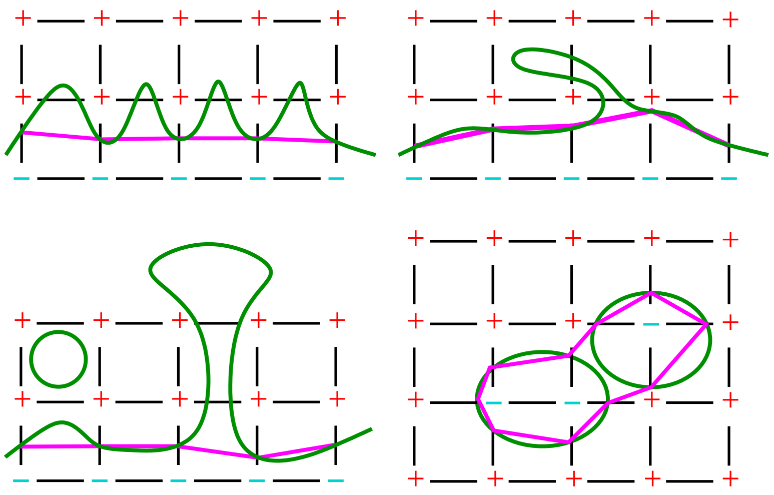

Figure 2: Problems with marching cube algorithms for curves.

implicit function passes between the two grid cells and its location can be found with simple root finding (either the bisection method or a form of modified regula falsi).

However, these marching cubes approaches have a variety of difficulties, such as those illustrated in Figure 2. On the upper left we see that if our grid spacing is too sparse, then we may under sample high frequency information about the curve. On the upper right, we see that we might miss small folds in the curve. On the lower left, we see that we may miss small, disconnected components. Further, in this figure we see that if the implicit bulges out between two grid cells, we may miss large portions of the curve, even if we attempt to expand our grid whenever adjacent boundary cells have opposite sign. And on the lower right, we see that if two disconnected components of the surface pass near one another, we may mistake them for a single component. These and other problems arise for surfaces, where the marching cubes algorithm also has some ambiguities in how to tessellate the grid cells [NY06].

A variety of methods have been proposed to address these issues. The methods closest to ours are *octree schemes* [WvG92, ACM05] and *marching tetrahedral schemes* [PT90, ST90]. In an octree scheme, space is partitioned into cubes. These cubes are analyzed, and each cube is subdivided into eight subcubes if the algorithm decides it needs details at a finer level. For example, in Paiva et al.'s scheme [PLLdF06], interval arithmetic is used to discard cubes that do not contain the surface, and derivative tests are used to decide if a cube should be subdivided. In our work, we examine the Bernstein coefficients both to decide that an octree cube does not contain the surface and to decide if a cube should be subdivided. This is similar to the approach of Alberti et al., who test the coefficients of the trivariate tensor product form of the algebraic to decide whether or not to subdivide. However, our algorithm partitions each cube into five tetrahedra and uses the triangular Bézier representation rather than the tensor product representation, making it a tetrahedral scheme. A further discussion of Alberti et al.'s scheme and a comparison to our scheme appears in Section 6.3.

For his masters thesis, Luk used A-patches to tessellate both algebraic curves and surfaces. The method is a form of *marching tetrahedron* [PT90, ST90]. An A-patch is a Bernstein representation of an algebraic function over a simplex (a triangle or tetrahedron in this paper). The advantage of the A-patch representation is that it guarantees that only a single sheet of the algebraic passes through a triangle/tetrahedral cell, thus avoiding most of the problems mentioned above.

Using A-patches is non-trivial. In particular, you need to partition the region of interest in the plane into a set of triangles (or tetrahedron when working in 3-space) and convert the algebraic to A-patch format for each triangle. However, the A-patch representation puts restrictions on the signs of the coefficients, so there is no guarantee that the Bernstein representation over any triangle will be in A-patch format. To address this issue, Luk performed 4-1 subdivision on any triangle not in A-patch format. Further, he showed that in regions of space not containing the curve, then all the Bernstein coefficients will be of one sign if you subdivide the region finely enough. This gave two stopping conditions for his recursive subdivision: either the curve does not pass through a cell or it is in A-patch format. Near singularities, you may never be in A-patch format, so one must also limit the depth of the subdivision.

While Luk demonstrated the feasibility of such an algorithm, there were several weaknesses in his actual implementation. In particular, his test for whether a patch was in A-patch format was incorrect; his root finding method could be more efficient; and for some algebraic functions (such as the "Clown Smile" of Lopez [LOdF02]), the algorithm is slow to converge in fairly benign regions of the curve.

The purpose of this technical report is to address these weaknesses, showing how to perform root finding more efficiently, how to test if a patch is in A-patch format, and extending beyond A-patches to a more general form of Bernstein representations that are single sheeted. With this last issue in mind, I will review A-patches and discuss why they are single sheeted so that we may better understand how to extend them to this larger class.

3 Bernstein Representations

For this work, we are interested in the univariate, bivariate, and trivariate Bernstein polynomials, how to evaluate them, and in one particular type of subdivision algorithm. I state here just the facts needed for this paper; for more details on Bernstein/Bézier representations, see [Man06].

The univariate Bernstein polynomials relative to a domain interval $[0, 1]$ are

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i,$$

where t is a scalar value; usually, we are most interested in $t \in [0, 1]$. The degree n Bernstein polynomials form a basis for degree n polynomials, and thus any degree n polynomial P may be expressed as

$$P(t) = \sum P_i B_i^n(t),$$

where P_i are *control points* in the range of the space of the polynomial and t is a scalar value in the domain. The resulting polynomial is a *Bézier curve* (see Figure 3, left, for an example curve).

A Bézier curve can be evaluated using *de Casteljau's algorithm*, which uses repeated linear interpolation (Figure 3, middle). In the figure, the curve is evaluated at $t = 0.5$ by computing the light gray points as

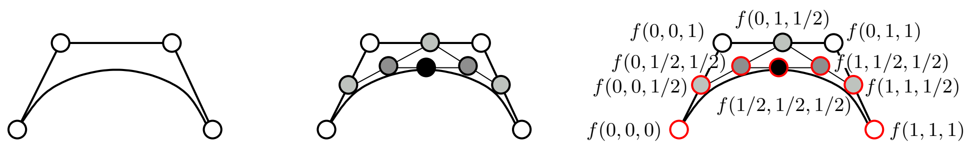

Figure 3: Left: A Bézier curve and its control points. Middle: The de Casteljau evaluation of a Bézier curve. Right: The de Casteljau evaluation with blossom labels; red points are subdivision points.

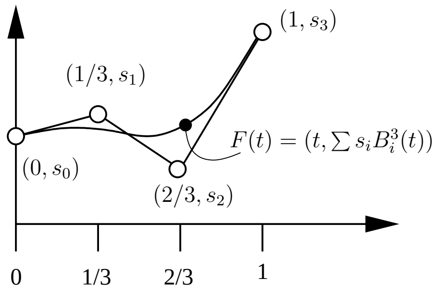

Figure 4: A scalar valued Bézier function represented as a 2D curve.

averages of the adjacent white ones, computing the dark gray points as averages of the adjacent light gray points, and computing the black point (which is $F(0.5)$) as an average of the dark gray points.

Since an A-patch represents an algebraic function, the Bernstein coefficients of an A-patch will be scalars. However, at times it is convenient to treat the univariate algebraic as a 2D curve. If our parameter value is t, we can re-express the Bernstein algebraic $\sum s_i B_i^n(t)$ as a 2D curve

$$\sum (i/n, s_i) B_i^n(t).$$

As Bernsteins have linear precision, the resulting function can be re-expressed as

$$\left(t, \sum s_i B_i^n(t)\right).$$

See Figure 4.

Another property of the Bernstein polynomials is that over the interval $[0,1]$, we have $0 \le B_i^n(t) \le 1$. Further, $\sum B_i^n(t) = 1$ for all t. These properties result in a weighted combination with Bernstein polynomials being a *convex combination* over $[0,1]$. For Bézier curves, this means that the curve lies in the convex hull of its control points. For scalar valued Bernstein functions, this means that over $[0,1]$, the value of the algebraic is no less than the smallest Bernstein coefficient and no greater than the largest Bernstein coefficient.

For our purposes, at times it is more convenient to represent the Bernstein polynomials in a more symmetric form as

$$B_{\vec{i}}^n(P) = \binom{n}{\vec{i}} p_0^{i_0} p_1^{i_1},$$

where p_0, p_1 are the Barycentric coordinates of P relative to a univariate domain interval $[A, B]$ with $p_0 + p_1 = 1$, and where $\vec{i} = (i_0, i_1)$ with $i_0, i_1 \ge 0$ and $i_0 + i_1 = n$.

A final property of Bernstein/Bézier representations that we will use in this paper is the *variation diminishing* property. In short, the property says that if we intersect a 2D Bézier curve and its control

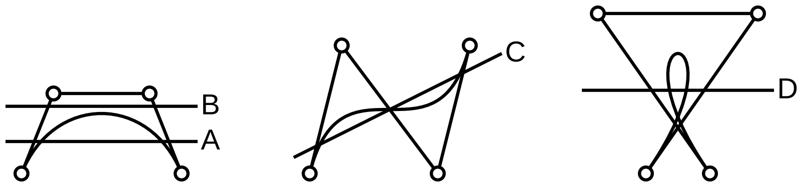

Figure 5: Variation diminishing. The lines intersect the curve no more times than they intersect the Bézier control polygon.

polygon with a line, then the line intersects the curve no more times than it intersects the control polygon; see Figure 5 and see Gallier's book for a proof [Gal00].

3.1 Blossoms

Blossoming is a way to label the control points of Bézier curves so that many of their properties are self-evident, as is the correctness of many algorithms. In brief, the blossom of a degree n polynomial F is the unique n-variate function f such that

- f is symmetric (i.e., we can change the order of its arguments without changing the value of the function);
- f is *multiaffine* (i.e., if we hold all but one argument fixed, then f is affine in the unfixed argument).
- f has the *diagonal property*; i.e, if we set all of f's arguments to the same value, then f evaluates to the same value as F at that argument: $f(u, u, ..., u) = F(u)$.

Notationally, if we want to repeat an argument k times, we will write $u^{\langle k\rangle}$. With this notation, the last property above should be written as $f(u^{\langle n\rangle}) = F(u)$.

The blossom values of the control points of a Bézier curve evaluated over $[0, 1]$ are $f(0^{\langle n\rangle})$, $f(0^{\langle n-1\rangle}, 1)$, $\ldots$, $f(1^{\langle n\rangle})$, i.e., when the arguments are all 0s and 1s. If we are working with the Bézier curve over an arbitrary interval $[a, b]$, then its control points are given by the blossom values $f(a^{\langle n-i\rangle}, b^{\langle i\rangle})$ for all $i \in \{0, \ldots, n\}$.

With this notation, by labeling the de Casteljau evaluation points with blossom values, we can immediately see the mathematics for the individual steps, and that de Casteljau's algorithm evaluates the curve (Figure 3, right). For example, since $t = 0.5 \cdot 0 + 0.5 \cdot 1$, we have

$$0.5f(0, 0, 0) + 0.5f(0, 0, 1) = f(0.5 \cdot 0 + 0.5 \cdot 1, 0, 0) = f(1/2, 0, 0).$$

The remaining points computed in the de Casteljau evaluation can be computed in a similar way, and the last point is a point on the curve: $f(1/2, 1/2, 1/2) = F(1/2)$. Note also that de Casteljau's algorithm subdivides the curve into two intervals $[0, 1/2]$ and $[1/2, 1]$ as seen by the labeling of the red circle points in Figure 3, right.

3.2 Higher Dimensions

The formula for univariate Bernstein polynomials can be generalized to higher dimensions. The bivariate Bernstein polynomials are expressed as

$$B^n_{\vec{i}}(P) = \binom{n}{\vec{i}} p_0^{i_0} p_1^{i_1} p_2^{i_2},$$

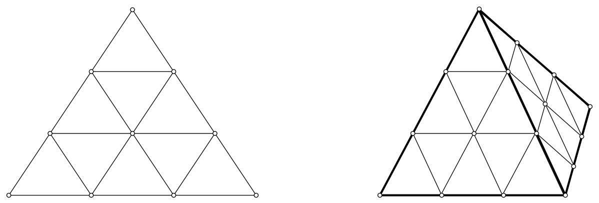

Figure 6: Bivariate and trivariate arrays.

where $\vec{i} = (i_0, i_1, i_2)$ with $i_0, i_1, i_2 \leq n$ and $i_0 + i_1 + i_2 = n$ and (p_0, p_1, p_2) are the Barycentric coordinates of P relative to a triangle $\triangle ABC$.

The trivariate Bernstein polynomials are expressed as

$$B^n_{\vec{i}}(P) = \binom{n}{\vec{i}} p_0^{i_0} p_1^{i_1} p_2^{i_2} p_3^{i_3},$$

where $\vec{i} = (i_0, i_1, i_2, i_3)$ with $i_0, i_1, i_2, i_3 \leq n$ and $i_0 + i_1 + i_2 + i_3 = n$ and (p_0, p_1, p_2, p_3) are Barycentric coordinates of P relative to a tetrahedron $\triangle ABCD$.

The Bernstein/Bézier representation of a k-dimensional, degree n polynomial is

$$\sum_{\vec{i}} P_{\vec{i}} B^n_{\vec{i}}(P) = \sum_{\vec{i}} f(D_0^{\langle i_0 \rangle}, \ldots, D_k^{\langle i_k \rangle}) B^n_{\vec{i}}(P)$$

where $\triangle D_0 \ldots D_k$ is the domain simplex.

Both the bivariate and trivariate Bernstein polynomials form bases for their respective polynomial spaces. Further, they retain many of the nice properties of univariate Bernstein polynomials, and polynomials represented in these bases retain many of the properties of univariate Bézier curves. In particular, for points inside the domain simplex, the Bernstein polynomials are non-zero and sum to one, and thus form a convex combination of their control points (or weights). However, there is no known generalization of variation diminishing for higher dimensional Bernstein representations.

The control points for the bivariate and trivariate representations can be organized in a triangular array and a tetrahedral array, respectively; see Figure 6. Note that the boundaries of a triangular Bézier patch are Bézier curves, and that the control points of these curves are the boundaries of the triangular array of control points. A similar property holds for Bézier volumes.

We can label the control points of higher dimensional Bézier arrays with blossom values. For a triangular domain $\triangle ABC$, the control points of the blossom corresponding to a degree n function F are $f(A^{\langle a \rangle}, B^{\langle b \rangle}, C^{\langle c \rangle})$ for all $a, b, c \geq 0$ with $a + b + c = n$ (Figure 7, left). For a tetrahedral domain $\triangle ABCD$, the control points of the blossom corresponding to a degree n function F are $f(A^{\langle a \rangle}, B^{\langle b \rangle}, C^{\langle c \rangle}, D^{\langle d \rangle})$ for all $a, b, c, d \geq 0$ with $a + b + c + d = n$.

There is a de Casteljau-style evaluation algorithm for both bivariate and trivariate Bernstein representations (see Figure 7, right, for the evaluation of a quadratic, bivariate Bézier patch). This algorithm also performs a 3-1 subdivision of the bivariate patch and a 4-1 subdivision of the trivariate volume (Figure 8). However, of more interest to us for this work, there is a 2-1 subdivision algorithm for bivariate patches [Pet94], and a 3-1 subdivision for trivariate volumes, where the subdivision point lies on one edge/face. Further, the 2-1 subdivision of a bivariate patch can be computed by performing curve subdivision on each row of the patch. Likewise, the 3-1 subdivision can be performed by doing a bivariate de Casteljau evaluation of each layer of the tetrahedral grid of control points. See Figure 9.

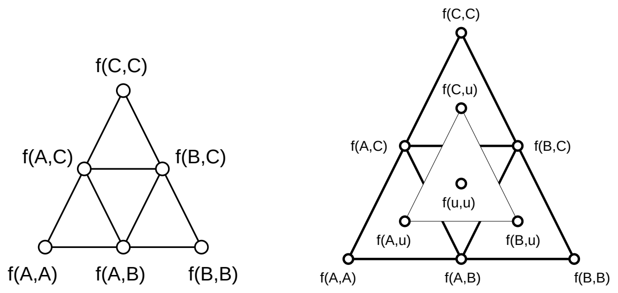

Figure 7: Blossom labels of quadratic, bivariate patch with domain $\triangle ABC$ and the de Casteljau evaluation of this patch.

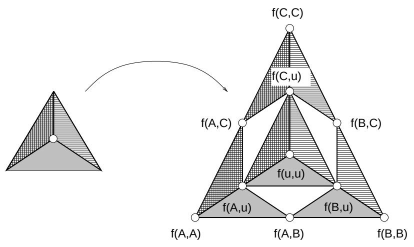

Figure 8: 3-1 subdivision of a quadratic patch.

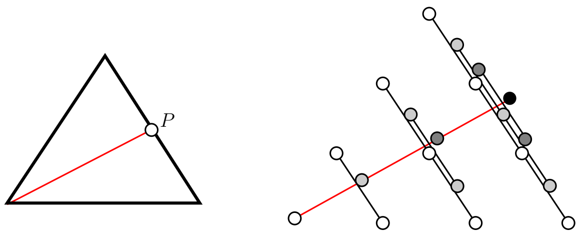

Figure 9: 2-1 subdivision via curve subdivision. The domain on the left is subdivided at P; the triangular patch can be subdivided by multiple curve evaluations, with the control points of the split boundary connected by the red line; the white points are the control points of the patch, and the light gray, dark gray and black points are the successive levels of de Casteljau evaluations of the "curves" of white control points.

3.3 Tensor Product Patches

The following are some bare minimum facts about tensor product Bézier patches that are needed in Section 4.1.

A tensor product Bézier patch over $[0,1] \times [0,1]$ has the form

$$T(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} B_i^n(u) B_j^m(v),$$

where the $P_{i,j}$ are the control points of the patch and the B_i^n and B_j^m are the Bernstein polynomials. The blossom of T is a function

$$t(u_1, \ldots, u_n; v_1, \ldots, v_m),$$

which is symmetric in the u_i and symmetric in the v_j, but interchanging a u_i with a v_j changes the value of the function. t is also multiaffine in each of its variables, and has the diagonal property.

Generalizing to an arbitrary rectilinear domain $[A,B] \times [C,D]$, the control points of the patch are related to the blossom values by

$$P_{i,j} = t(A^{\langle n-i \rangle}, B^{\langle i \rangle}; C^{\langle m-j \rangle}, D^{\langle j \rangle}).$$

4 A-patches

An A-patch is an algebraic function represented relative to the Bernstein basis, where the scalar coefficients meet some restrictions on their sign. For algebraic curves, we use the bivariate Bernsteins over a triangle. For now, assume that the algebraic function is non-zero at each of the corners. Then clearly, either all three corners have the same sign, or two of the corners will have one sign and the third corner has the opposite sign. The former case is not in A-patch format. In the latter case, we will consider the triangular array of Bernstein scalars in layers parallel to the edge having two corners of the same sign. For the patch to be in A-patch format, the following must hold:

- All the scalars on the edge containing two vertices of one sign must all be of the same sign. Some but not all of these scalars may be zero.
- There must be a separating layer, where all the scalars on one side have one sign, and all the scalars on the other side have the other sign. The separating layer itself may have mixed sign.

Mathematically, for triangle $\triangle V_0 V_1 V_2$ where $F(V_0) > 0$, $F(V_1) < 0$ and $F(V_2) < 0$, the conditions on the coefficients of a degree n Bernstein representation to be in A-patch format are that there exists a k with $0 < k < n$ such that

$$\begin{aligned} f(V_0^{\langle i_0 \rangle}, V_1^{\langle i_1 \rangle}, V_2^{\langle i_2 \rangle}) &\le 0, \quad 0 \le i_0 < k \\ f(V_0^{\langle i_0 \rangle}, V_1^{\langle i_1 \rangle}, V_2^{\langle i_2 \rangle}) &\ge 0, \quad k < i_0 \le n \end{aligned}$$

where the remaining coefficients (i.e., for $i_0 = k$) may be of any sign. Similar conditions hold for when one of the other two corners is the corner with the different sign. Note that we already have the additional conditions that the values at the corners are not zero; see Section 4.2 for a discussion of zeros at the corners.

See Figure 10, left, for an example of a triangular array with a separating layer. The advantage of having a separating layer is that (as proven in the next section) any line between the vertex of one sign to any point on the edge of one sign has exactly one zero on it. This makes it easy to find a piecewise linear approximation to the algebraic curve within this triangle (Figure 10, right): construct a set of samples on the edge of one sign, and for each sample, do root finding between the sample and the corner of opposite sign. As there

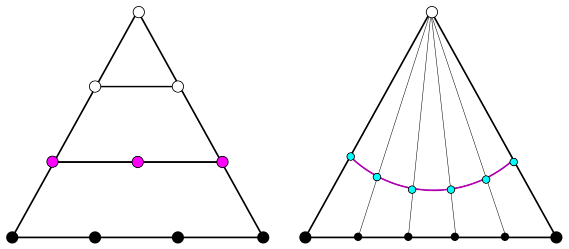

Figure 10: Left: The separating layer, shown in magenta, may have mixed sign. The white points all have to have the same sign, which must be opposite to the sign of the black points. Right: if we have a separating layer, then a single sheet of the algebraic passes through the triangle, making it easy to tessellate.

is exactly one zero along each segment, the root finding is relatively simple. Further, the A-patch format guarantees that no other portion of the algebraic curve is within this triangle.

The trivariate case is similar, although more involved. First, there are two types of trivariate A-patches: three sided and four sided. Three sided A-patches must meet the following criteria:

- One corner must have one sign, and all the coefficients on the opposite face must have the opposite sign. Some but not all of these face coefficients may be zero.
- Layering the coefficients in layers parallel to the face whose coefficients all have one sign, there must be a separating layer where the coefficients on one side must all be of one sign and those on the other side must be the other sign. The coefficients on the separating layer may be of any sign, and any of the non-corner coefficients may be zero.

Mathematically the conditions are a straightforward generalization of those for triangular patches. For tetrahedron $\triangle V_0V_1V_2V_3$ where $F(V_0) > 0$, $F(V_1) < 0$, $F(V_2) < 0$, and $F(V_3) < 0$, the conditions on the coefficients of a degree n Bernstein representation to be in A-patch format are that there exists a k with $0 < k < n$ such that

$$\begin{aligned} f(V_0^{\langle i_0\rangle}, V_1^{\langle i_1\rangle}, V_2^{\langle i_2\rangle}, V_3^{\langle i_3\rangle}) \leq 0, &\quad 0 \leq i_0 < k \\ f(V_0^{\langle i_0\rangle}, V_1^{\langle i_1\rangle}, V_2^{\langle i_2\rangle}, V_3^{\langle i_3\rangle}) \geq 0, &\quad k < i_0 \leq n \end{aligned}$$

where the remaining coefficients (i.e., for $i_0 = k$) may be of any sign. Similar conditions hold for when one of the other two corners is the corner with the different sign. As stated, we can not have zeros at the corners of the patch; see Section 4.2 for a discussion of zeros at the corners.

A four sided A-patch must meet the following criteria:

- All the coefficients of one edge of the tetrahedron must be of one sign, while those of the opposite edge must be of the opposite sign. Some but not all of these edge coefficients may be zero.
- Layering the coefficients in layers parallel to the two edges of one sign, there must be a separating layer where the coefficients on one side must all be of one sign and those on the other side must be the other sign. The coefficients on the separating layer may be of any sign, and any of the non-corner coefficients may be zero.

Mathematically, for tetrahedron $\triangle V_0V_1V_2V_3$ where $F(V_0) > 0$, $F(V_1) > 0$, $F(V_2) < 0$, and $F(V_3) < 0$, the conditions on the coefficients of a degree n Bernstein representation to be in A-patch format are that there

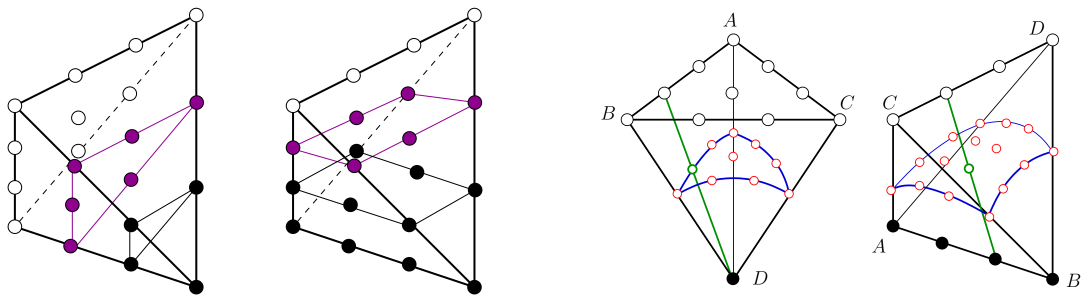

Figure 11: Left: Separating layers for 3- and 4-sided A-patches. Right: Tessellation patterns for 3- and 4-sided A-patches.

exists a k with $0 < k < n$ such that

$$f(V_0^{\langle i_0\rangle}, V_1^{\langle i_1\rangle}, V_2^{\langle i_2\rangle}, V_3^{\langle i_3\rangle}) \leq 0, \quad 0 \leq i_0 + i_1 < k$$
$$f(V_0^{\langle i_0\rangle}, V_1^{\langle i_1\rangle}, V_2^{\langle i_2\rangle}, V_3^{\langle i_3\rangle}) \geq 0, \quad k < i_0 + i_1 \leq n$$

where the remaining coefficients (i.e., for $i_0 + i_1 = k$) may be of any sign. Similar conditions hold for when one of the other two corners is the corner with the different sign. As stated, we can not have zeros at the corners of the patch; see Section 4.2 for a discussion of zeros at the corners.

See Figure 11 for examples of the separating layers.

Note that the conditions given here are slightly more restrictive than those given by Bajaj [BCX95], whose conditions only guarantee that a line segment from a vertex to the opposite side will intersect the algebraic no more than once; the stronger conditions here guarantee that the line segment will intersect exactly once, which is a more useful condition for tessellating algebraic curves and surfaces.

4.1 Proof of single sheetedness

Understanding why the A-patch conditions result in a single sheet passing through the simplex will enable us to generalize these conditions to improve our algebraic curve/surface tessellator. Thus, we sketch here a proof of single sheetedness.

Given a degree n, bivariate A-patch over triangle $\triangle V_0V_1V_2$, the following shows that it is single sheeted. Assume $F(V_0) > 0$ and $F(V_1) < 0$ and $F(V_2) < 0$. Let $s_{i,j,k}$ be the scalar values for layer i, with $i = 0..n$. Since we are in A-patch format, there exists $0 < \ell < n$ such that $s_{m<\ell,j,k} < 0$, $s_{m>\ell,j,k} > 0$, and where $s_{\ell,j,k}$ may have any value.

If we perform 2-1 subdivision of this patch from V_0 to any point V on the opposite side, and extract the Bézier coefficients b_i of the split edge, we see that $b_i < 0$ for $i < \ell$ and $b_i > 0$ for $i > \ell$ since each b_i is computed as a convex combination of the scalar values for its layer, and where b_ℓ may have any value. Representing this scalar valued function as a 2D curve, we see that we are in the configuration illustrated in Figure 12. By the variation diminishing property, the curve crosses the horizontal zero line no more than once since the control polygon crosses this line exactly once, and since $b_0 < 0$ and $b_n > 0$, the curve crosses the horizontal zero line. Thus, the curve crosses the horizontal zero line exactly once.

This proof immediately generalizes to 3-sided surface A-patches, using 3-1 subdivision of the tetrahedron rather than 2-1 subdivision of the triangle.

The proof for four sided patches is similar. Given a four sided patch with $F(V_0) > 0$, $F(V_1) > 0$, $F(V_2) < 0$ and $F(V_3) < 0$ where F is degree n, we want to consider the function F on the line segment from

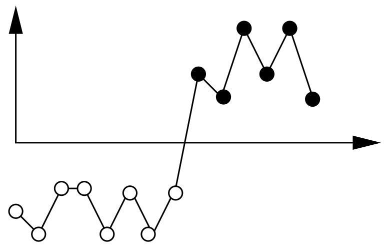

Figure 12: With a sequence of negative coefficients followed by positive coefficients, there is one crossing of the zero line.

P to Q where P lies on the segment V_0V_1 and Q lies on the line segment from V_2V_3. We know there exists Bézier control points $b_0, \ldots, b_n$ that represent the value of F on the line segment PQ. Further, we know that the blossom value of b_i is $f(P^{\langle n-i \rangle}, Q^{\langle i \rangle})$.

We now make the observation that the labeling of each layer of control points indicates that P_i can be computed by a tensor product evaluation of one of these layers (see Figure 13). Since layers are separated by sign, so are the control points of this curve and we have our result.

4.2 Zeros at the Corners

The A-patch conditions given in Section 4 do not allow for the algebraic function to be zero at the corners of the triangle or tetrahedron. While technically, this occurs only on a set of measure zero, in practice people are likely to create algebraic functions and tessellations that will result in zeros at the corners. We can extend our definition to allow zeros at the corners although the restrictions on the remaining coefficients are stronger. Effectively, having one or more zeros at the corners makes one edge (in the triangular case or 4-sided case) or one face (in the 3-sided case) into the "separating layer", albeit with additional restrictions.

In particular, the single sheeted A-patch conditions for curves (triangular patches) for triangle $\triangle V_0V_1V_2$ with $F(V_0) > 0$, $F(V_1) \le 0$, and $F(V_2) \le 0$ are

$$\begin{aligned} f(V_1^{\langle n \rangle}) \le 0, \quad f(V_2^{\langle n \rangle}) \le 0 & \\ f(V_1^{\langle i_1 \rangle}, V_2^{\langle i_2 \rangle}) < 0, &\quad i_1 > 0, i_2 > 0 \\ f(V_0^{\langle i_0 \rangle}, V_1^{\langle i_1 \rangle}, V_2^{\langle i_2 \rangle}) \ge 0, &\quad 0 < i_0 \le n \end{aligned}$$

For a 3-sided patch with tetrahedron $\triangle V_0V_1V_2V_3$ where $F(V_0) > 0$, $F(V_1) \le 0$, $F(V_2) \le 0$, and $F(V_3) \le 0$ then the conditions on the coefficients of a degree n Bernstein representation to be in A-patch format are that there exists a k with $0 < k < n$ such that

$$\begin{aligned} f(V_1^{\langle n \rangle}) \le 0, \quad f(V_2^{\langle n \rangle}) \le 0, \quad f(V_3^{\langle n \rangle}) \le 0 & \\ f(V_1^{\langle i_1 \rangle}, V_2^{\langle i_2 \rangle}, V_3^{\langle i_3 \rangle}) < 0, &\quad i_1, i_2, i_3 > 0 \\ f(V_0^{\langle i \rangle}, V_1^{\langle i_1 \rangle}, V_2^{\langle i_2 \rangle}, V_3^{\langle i_3 \rangle}) \ge 0, &\quad 0 < i_0 \le n \end{aligned}$$

For a 4-sided patch with tetrahedron $\triangle V_0V_1V_2V_3$ where $F(V_0) > 0$, $F(V_1) > 0$, $F(V_2) \le 0$, and $F(V_3) \le 0$ then the conditions on the coefficients of a degree n Bernstein representation to be in A-patch format are that there exists a k with $0 < k < n$ such that

$$f(V_2\langle n \rangle) \le 0, \quad f(V_3^{\langle n \rangle}) \le 0$$

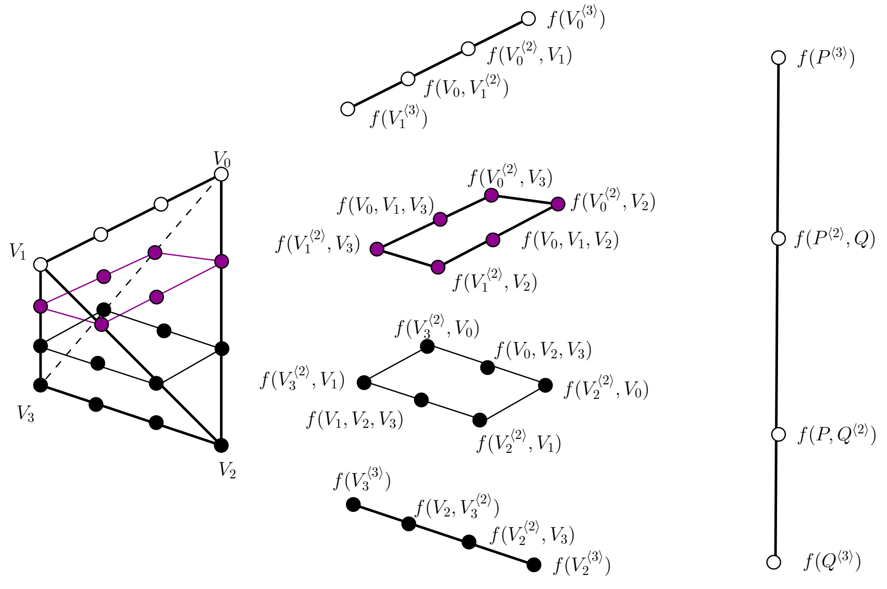
V_0
V_1
V_3
V_2
$f(V_0^{\langle 3\rangle})$
$f(V_0^{\langle 2\rangle}, V_1)$
$f(V_0, V_1^{\langle 2\rangle})$
$f(V_1^{\langle 3\rangle})$
$f(V_0^{\langle 2\rangle}, V_3)$
$f(V_0, V_1, V_3)$
$f(V_0^{\langle 2\rangle}, V_2)$
$f(V_1^{\langle 2\rangle}, V_3)$
$f(V_0, V_1, V_2)$
$f(V_1^{\langle 2\rangle}, V_2)$
$f(V_3^{\langle 2\rangle}, V_0)$
$f(V_0, V_2, V_3)$
$f(V_3^{\langle 2\rangle}, V_1)$
$f(V_2^{\langle 2\rangle}, V_0)$
$f(V_1, V_2, V_3)$
$f(V_2^{\langle 2\rangle}, V_1)$
$f(V_3^{\langle 3\rangle})$
$f(V_2, V_3^{\langle 2\rangle})$
$f(V_2^{\langle 2\rangle}, V_3)$
$f(V_2^{\langle 3\rangle})$
$f(P^{\langle 3\rangle})$
$f(P^{\langle 2\rangle}, Q)$
$f(P, Q^{\langle 2\rangle})$
$f(Q^{\langle 3\rangle})$

Figure 13: Layers of a four sided patch can be evaluated as tensor products. In this figure, P lies on the segment V_0V_1 and Q lies on the segment V_2V_3.

$$\begin{aligned} f(V_2^{\langle k\rangle}, V_3^{\langle \ell\rangle}) &< 0, \qquad i_2, i_3 > 0 \\ f(V_0^{\langle i\rangle}, V_1^{\langle j\rangle}, V_2^{\langle k\rangle}, V_3^{\langle \ell\rangle}) &\geq 0, \quad 0 < i_0 + i_1 \leq n \end{aligned}$$

5 Tessellating Algebraic Curves

Given what we know about A-patches, we can now devise an algorithm to adaptively tessellate an algebraic curve:

1. Subdivide a region of interest into triangles.
2. Convert to the Bernstein representation over each triangle.
3. If the Bernstein representation over any triangle is in A-patch format, then tessellate it.

   Otherwise, split the triangle 4-1 and repeat on the new triangles.

Many key details are missing about the algorithm: how do we select the region of interest? How do we convert to Bernstein representation? And do we know the subdivision will stop, or do we need to limit the depth of the subdivision?

The selection of the region of interest is an issue with all algorithms that tessellate implicit functions. We will just assume that we are given a square region, which we will split into two triangles and then apply the algorithm.

Conversion from monomial to Bernstein basis can be done using a change of basis matrix; alternatively, you can apply de Casteljau's algorithm three times to effect the change [BG93]. In our implementation, we chose the former method. Note that the change of basis matrix only has to be computed once for all triangles; however, it must be applied once per triangle.

The question of "will the recursive subdivision stop?" is more tricky. Basically, there are two stopping conditions: the Bernstein representation is in A-patch format, in which case we can tessellate it, or all the coefficients in the Bernstein representation are of one sign, in which case because of the convex combination property we know that the algebraic function has no zeros within the triangle and we need no longer consider it (this latter condition should be added to the algorithm above in Step 3).

In Luk's thesis, a further observation is made: if the algebraic function is of one sign over a triangle, then if you subdivide deep enough (with the exact depth being dependent on a bound away from zero for the algebraic function over the triangle), then all the Bernstein coefficients will be of one sign. Essentially, this guarantees that the algorithm will stop in regions not containing the algebraic curve.

However, we do not have any guarantees about the algorithm terminating around the curve. In fact, in regions containing a singularity, we know the opposite: the algorithm will not terminate, unless the singularity ends up on a vertex of a triangle, in which case those triangles might be in A-patch format. This means that we must limit the depth of the recursion, otherwise we are guaranteed that the algorithm will not terminate.

5.1 Examples

Figure 14 shows some examples of our algorithm on functions used in [LOdF02]. Note that two of the examples in that paper referred to webpages that no longer existed and thus are not included here. In all examples, our algorithm started with two triangles that split a square in half diagonally. The formulas for these examples can be found in Appendix A.

A couple of notes about these examples. First, note the deep subdivision that occurred for the Clown Smile example. The actual result is worse than the figure indicates: the recursion is much deeper than it appears, and the Bernstein representation never converged to A-patch format in some regions. This is a serious problem, and one that I address in Section 5.2.

The second note is that only the Pear example has a singularity in it. To better demonstrate how our algorithm localizes singularities, we ran another example (Figure 15). Where the curve crosses itself, the curve is not a manifold. Unless this point lands on a triangle vertex, our algorithm will never converge. However, it will localize the singularity, and the regions in which it did not converge can be processed later to determine what to do in the region.

5.2 Improving the Curve Tessellation Algorithm

The Clown Smile example indicates that there is a problem with the algorithm: in some fairly simple cases, it is slow to converge (at best). Looking at the patches that the algorithm decides to subdivide deeply, we find that their coefficients are similar to those shown in Figure 16.

From this example, we see that the patch is almost in A-patch format, but it has two columns of mixed coefficients rather than one. Further, both of these columns consist of a sequence of positive values followed by a sequence of negative values. The A-patch conditions are sufficient but not necessary conditions to have a single sheet passing through the triangle. The example we see here suggests another condition: if we have multiple columns of mixed coefficients of the form positive followed by negative (which I will refer to as having *one change*), we will have a single sheet if the zeros of the columns are in the correct order.

To state the conditions, we will treat each layer of control points as a Bézier curve parameterized over $[0, 1]$. Since we are in A-patch format, our patch has $n + 1$ layers and the new conditions to have a single sheet pass through the patch are that the layers are structured as follows:

- Coefficients in layers 0 to ℓ are strictly less than 0;
- Coefficients in layers m to n are strictly greater than 0, where $m - \ell > 2$;

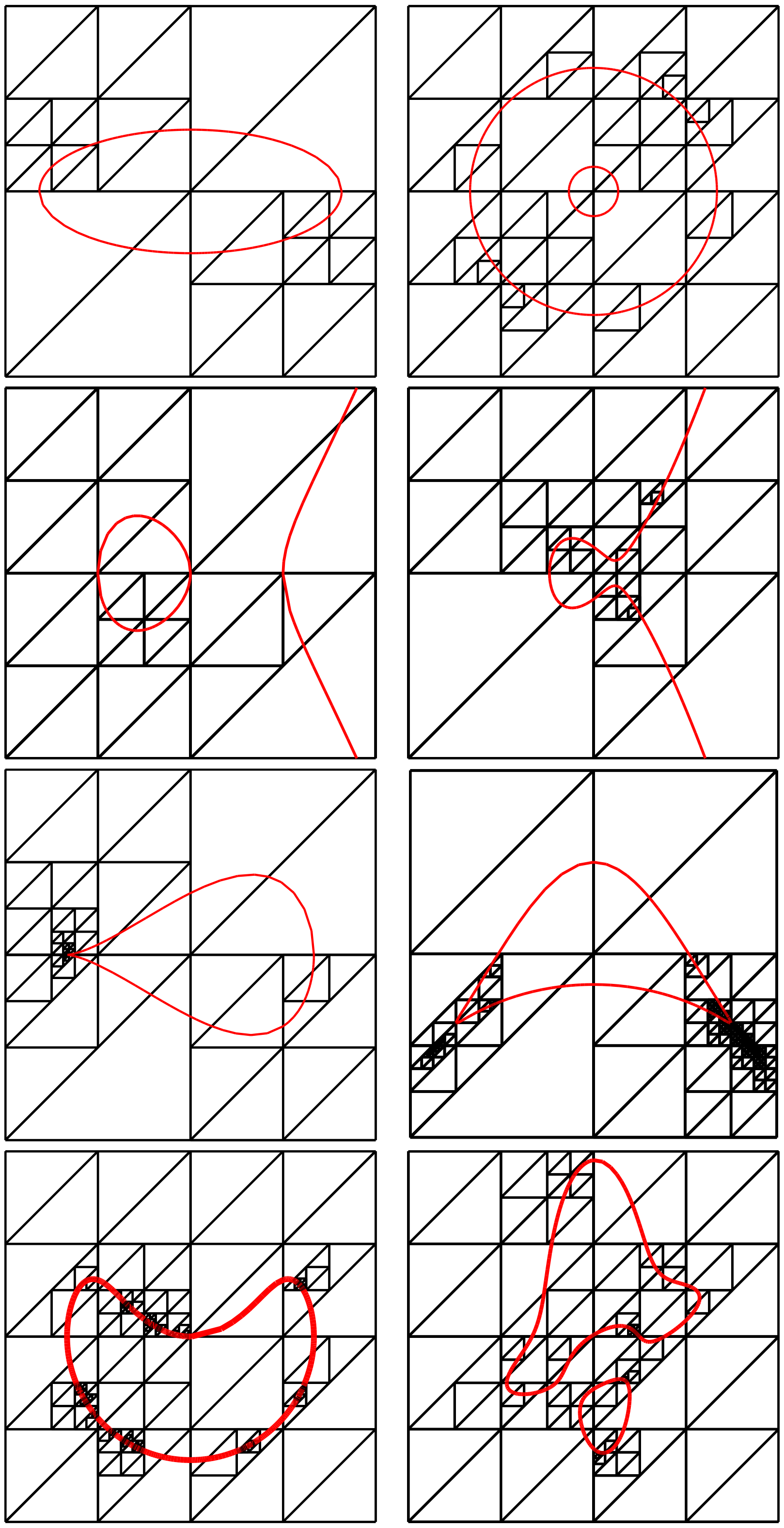
Figure 14: Examples from [LOdF02].

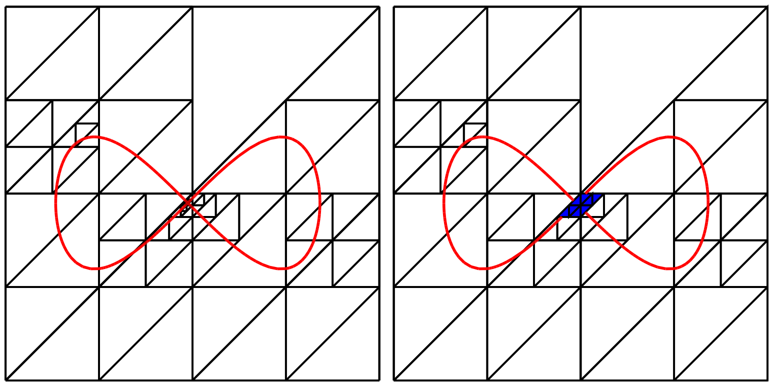

Figure 15: A curve with a singularity.

```
61.6
36.4 24.0
21.9 13.6 9.38
13.1 7.75 4.95   3.44
7.80 4.22 2.43   1.47   0.951
4.64 2.19 1.01   0.395  0.056 -0.118
3.01 1.20 0.327 -0.131 -0.381 -0.517 -0.580
2.45 0.93 0.161 -0.271 -0.521 -0.665 -0.744 -0.778
2.59 1.16 0.360 -0.134 -0.447 -0.643 -0.762 -0.831 -0.863
```

Figure 16: Clown Smile coefficients.

- Coefficients in layers $\ell+1$ to $m-1$ are one-change, and when treated as Bézier curves have zeros at $t_{\ell+1}, \ldots, t_{m-1}$, and one of the following conditions is met:
  - Each layer starts with a positive value and $t_{\ell+1} \geq \ldots \geq t_{m-1}$.
  - Each layer starts with a negative value, and $t_{\ell+1} \leq \ldots \leq t_{m-1}$.

Similar conditions hold when the initial layers are negative and the final layers are positive.

The point here is that when treating the layers as scalar valued Bézier curves, for any t value from 0 to 1, the layers evaluate to a sequence of positive values followed by a sequence of negative values, and they never alternate from positive to negative and back to positive again. See Figure 17 for an illustration; in this figure, the vertical axis represents the t value at which we evaluate each the curve represented by each layer of control points. The horizontal axis represents the layers of control points. The shades region of the figure indicates whether each layer evaluates to a positive or negative value. Note that the left column represents a layer in the triangular patch that is just the corner control point; i.e., it is the constant function, and thus is always of one sign.

In our example, the fourth column (when treated as a Bézier curve with 3.4 as the first coefficient) has a zero at 0.66021, while the fifth column has a zero at 0.35435. This meets our conditions on the zeros, and a single sheet of the curve passes through this triangle.

We can implement this new condition in two steps. First, we modify our A-patch test to allow for one change columns (or row, or diagonals, depending on which two corners have the same sign). Then we test

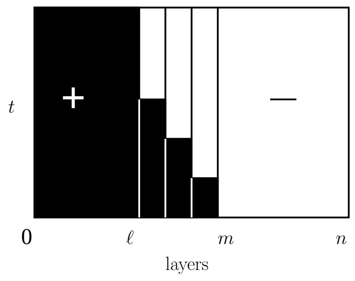

Figure 17: The zeros of the one-change layers must be ordered so that horizontal lines also have exactly one sign change.

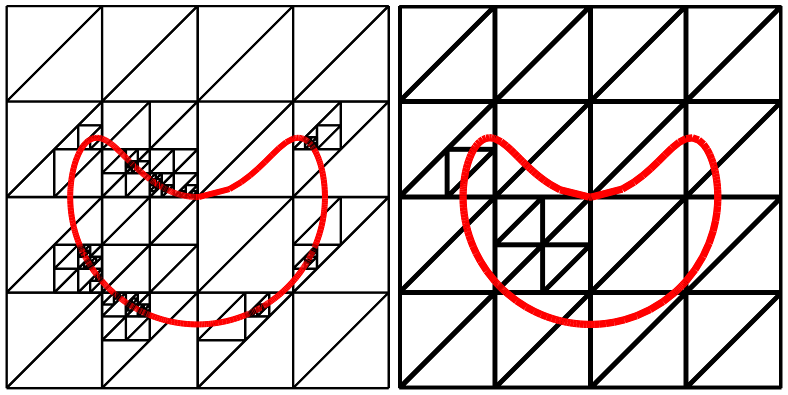

Figure 18: Clown Smile using: just A-patch condition; sign test and zeros test.

the one change columns to see that their zeros are in the correct order. If the zeros are in the correct order, then a single sheet passes through the triangle and we can tessellate it.

Figure 18 shows two tessellations of the Clown Smile. The one on the left just tests the A-patch condition, and has sections that never converge to the depth at which we tested. The figure on the right shows the result that incorporates the new test for single sheetedness. For this particular example, the reduction in the depth of the subdivision is high, resulting in a reduction of cells visited from 3414 cells down to 54 cells. Further, with the new single sheet test, our algorithm converged in all regions.

Table 1 in the Appendix A shows the number of cells visited by our A-patch algorithm, our modified algorithm, and for comparison, the method of Lopes et al. (although a direct comparison of the number of cells between the Lopes algorithm and ours is a bit difficult, since they have square cells while ours are triangular).

Figure 19 shows the subdivision of the modified A-patch algorithm on all the Lopes examples for which there was a different depth of subdivision. Note that with the new test, our algorithm converged everywhere for all examples except the Bicorn, where it failed to converge in 19 cells.

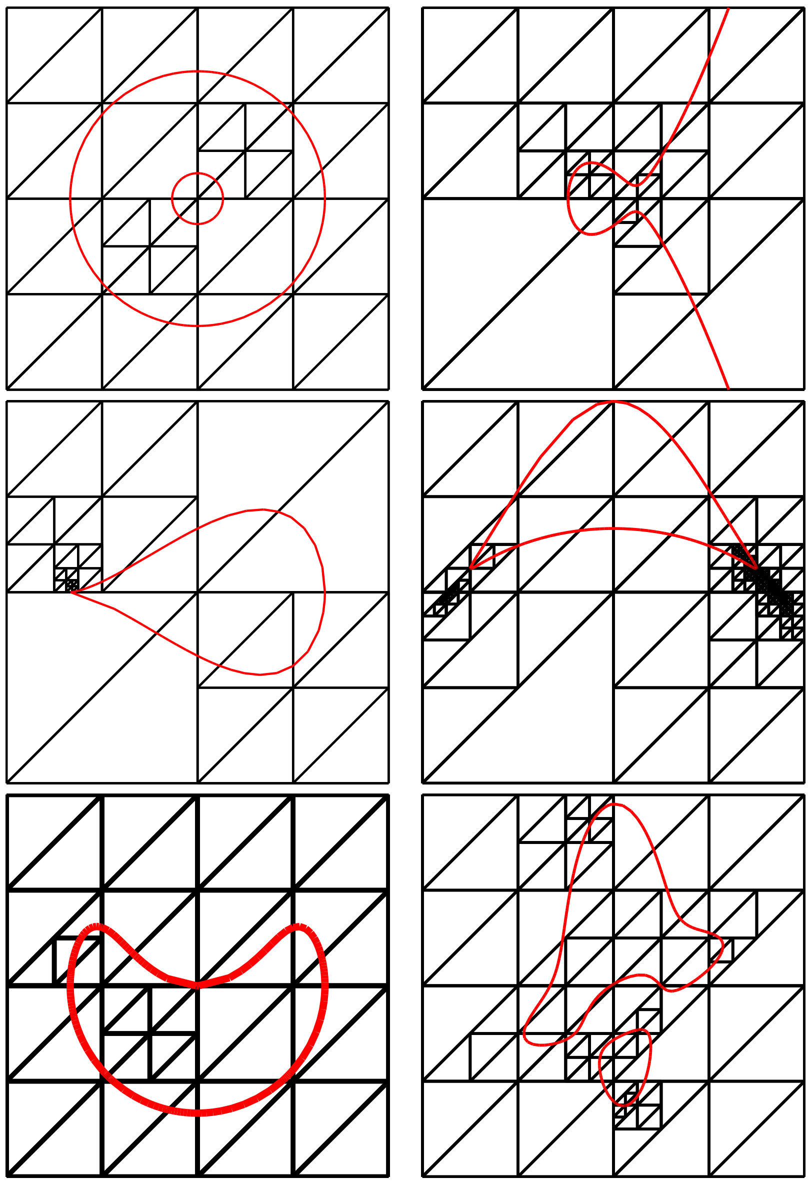

Figure 19: Examples from [LOdF02] using modified A-patch conditions.

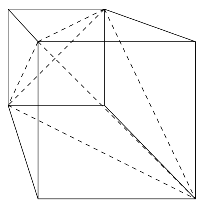

Figure 20: Splitting a cube into five tetrahedra.

6 Tessellating Algebraic Surfaces

Surfaces are more problematic to tessellate than curves for several reasons. First, there is no nice subdivision for tetrahedron that preserves the shape of the tetrahedron. Second, there are more cases to test if the Bernstein representation over a tetrahedron is in A-patch format than there are for triangles (seven for tetrahedron versus three for triangles). Third, the single sheet generalization of A-patches that is needed for curves (see Section 5.2) does not generalize to surfaces in a way that is computationally efficient. And fourth, the number of coefficients in the polynomial representation over a tetrahedron significantly increases the computational costs over triangles. Additional difficulties arise because of stitching issues between levels of different subdivision, although those concern us less in this paper as they are problems common to such methods and not specific to our A-patch method. The method still works for surfaces, but it is slower and there are more likely to be single sheeted regions in which the algorithm does not converge.

To deal with the lack of a subdivision algorithm for tetrahedron, Luk used a cubical grid, where each cube was subdivided into five tetrahedra (Figure 20). The algorithm then becomes

1. Convert to the Bernstein representation for the algebraic in each of the five tetrahedra for a cube.
2. Test the Bernstein representations in all five tetrahedra:
    - If all five have strictly positive (negative) coefficients, then the surface does not pass through this cube and we no longer need to consider it.
    - If some of the tetrahedron are in A-patch format and each of the other tetrahedron have coefficients of one sign, then tessellate the A-patches.
    - If any of the Bernstein representations has mixed sign coefficients that are not in A-patch format, then
        - If cube size is above minimum, then discard the Bernstein representations, subdivide the cube into eight subcubes, and repeat.
        - Else mark cube as unresolved.

6.1 Examples

Figure 21 gives some examples of simple algebraic surfaces tessellated by this algorithm. On the left is the surface, on the right is the grid of cubes and tetrahedron used to partition space. In the case of the cone, the algorithm can not converge at the singularity at the tip of the cone; however, it does localize this singularity. The yellow cubes indicate cubes for which some of the tetrahedra were of mixed sign, but not in A-patch format. In the examples in Figure 21, the algorithm started with a single cube of size $2 \times 2 \times 2$, and used a minimum cube size of 0.02.

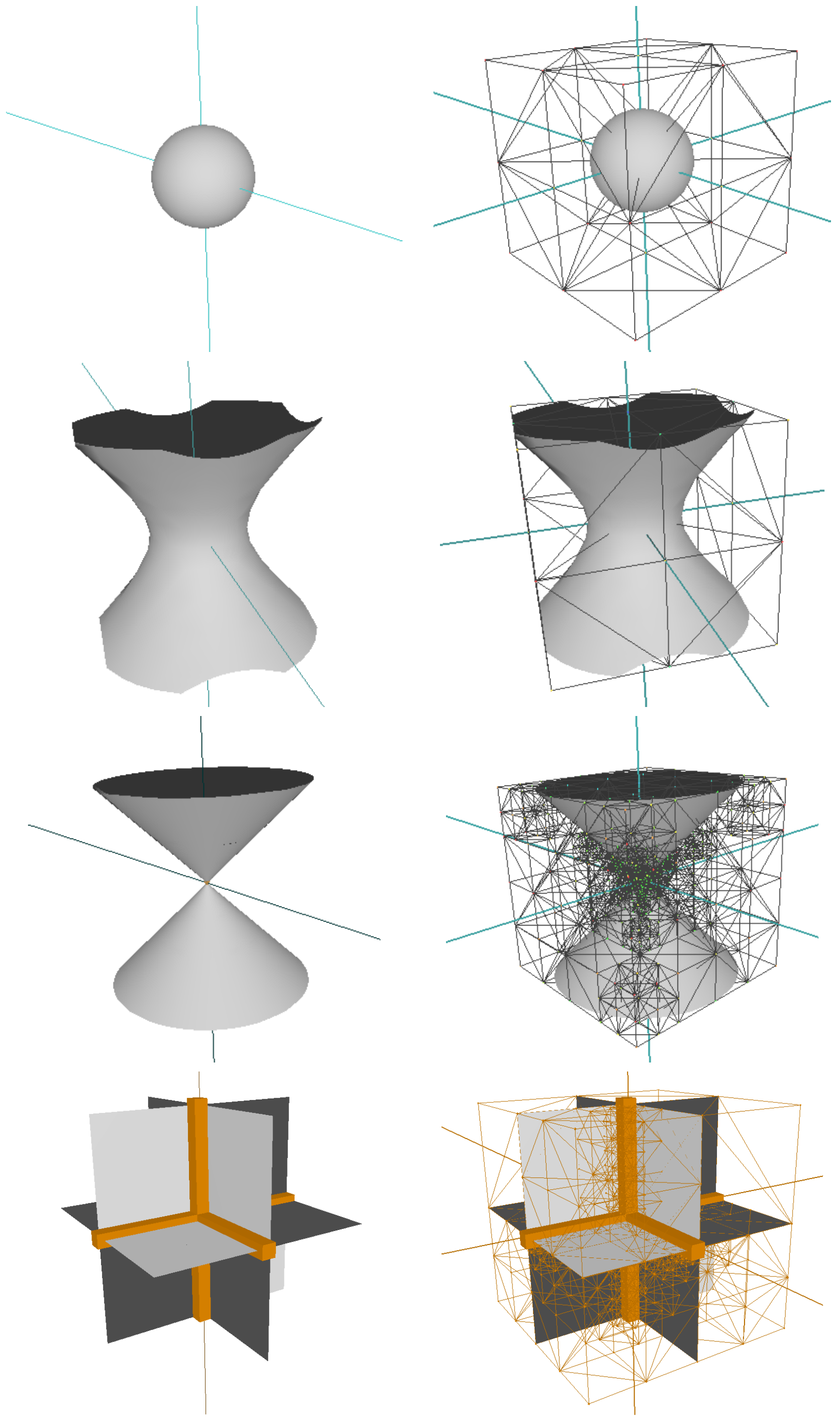

Figure 21: Examples of simple algebraic surfaces: sphere, hyperboloid, cone, $xyz = 0$.

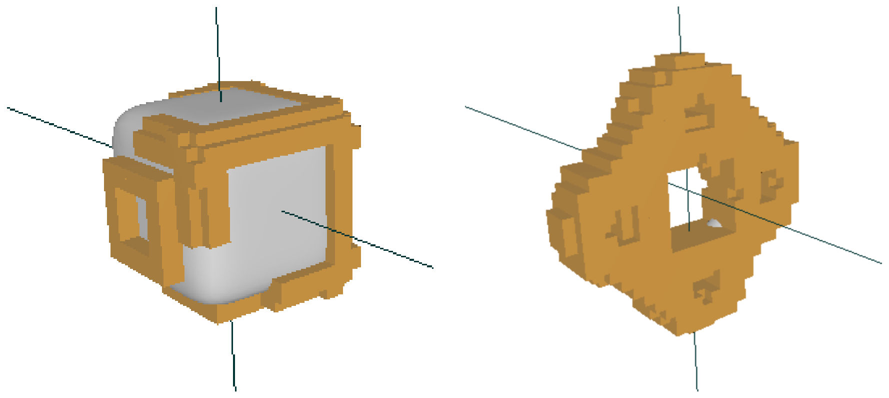

Figure 22: Two examples of more complex algebraic surfaces where the algorithm failed to converge.

While the algorithm works reasonable well on simple algebraic surfaces, for more complex surfaces, it encounters the same problem that the curve algorithm faced: for some data, the algorithm is slow to converge or possibly never converges at all. Figure 22 shows two examples in which the algorithm had difficulties. In both these examples, the depth of the subdivision was reduced as otherwise the algorithm ran out of memory. Both examples indicate that an additional single sheet criteria is needed to make the algorithm usable.

6.2 Improving the Surface Tessellation Algorithm

Mathematically, the extended single sheet criteria for curves generalizes in a straightforward way to surfaces. Unfortunately, the implementation of the generalized criteria is difficult. Instead, I propose alternative conditions that allow for faster convergence while still meeting most of the A-patch guarantees.

For curves, I created a new single sheet criteria that allowed for one change layers of control points. As each layer was effectively a curve, by requiring that the roots on these layers be in increasing/decreasing order, we can ensure a single sheet of the curve passes through the triangle. For surfaces, each layer is a triangular array of values, with a zero curve passing through the layer. The values on one side of the curve are positive, and on the other side they are negative. To ensure that a single sheet of the algebraic surface passes through the tetrahedron, we must ensure that the positive/negative layers are nicely stacked. This requires (a) that the curves do not intersect; and (b) that the negative regions all contain one common point of the same sign. Figure 23 illustrates this condition.

Unfortunately, this condition is hard to test computationally. As a pre-condition, it is clear that there must be a sequence of mixed sign coefficients separating a group of layers with positive coefficients from a group of layers having negative coefficients. However, even the conditions on the sign of the scalars within a triangular array for a layer are hard to specify and test. A simple test would be to require that the A-patch curve conditions hold for the mixed sign layers, with the additional condition that all the mixed layers have the same signs at their corners. However, weaker conditions would also guarantee that the tetrahedron has a single sheet passing through it, and such weaker conditions may be needed to make the test useful.

Instead, I chose to allow a set of mixed sign layers separating a set of positive layers from a set of negative layers. This arrangement ensures that there is at least one root on each segment between a vertex of one sign and a face of opposite sign (or between an edge of one sign and an edge of opposite sign). Further, I require that the faces of the tetrahedron meet the extended single sheet conditions of Section 5.2, or (in the

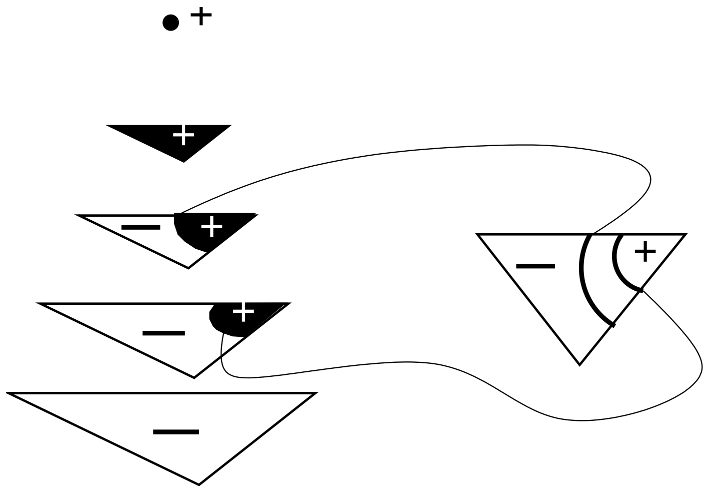

Figure 23: Conditions on layers to have single sheet. Left: tetrahedral array separated into layers; right, common parameter domain.

case of a three sided patch) that the face's coefficients are all of one sign. This ensures that only a single sheet passes through the faces of the tetrahedron.

With these conditions, we might still have multiple sheets within a tetrahedron. As a heuristic to detect such cases, when tessellating the surface within a tetrahedron, I test that there is a single root for each root finding problem. This root finding problem is searching a trivariate polynomial along a line through a tetrahedron, but it can be converted into a root finding problem on a univariate Bézier curve for a three sided patch by doing 3-1 subdivision of the tetrahedron; a similar subdivision can be done to a four sided patch. As the heuristic condition, I require that the coefficients of these univariate Bézier curves have exactly one sign change, although I do 2-1 subdivision of the curve up to four levels in depth to improve the test. This guarantees that the algebraic surface intersects this segment exactly once.

While this root testing condition does not guarantee single sheetedness, in the examples in this paper, the Crixxi surface was the only surface that had multiple zeros between a vertex and face for a patch with multiple mixed sign layers, suggesting that this relaxed condition is sufficient when subdividing deep enough and far enough away from singularities.

Figure 24 shows some examples of tessellations of algebraic surfaces generated using the relaxed single sheet condition.

6.3 Comparison to Alberti et al.

Alberti et al. [ACM05] proposed an algorithm similar to ours. The main difference is that they use a trivariate tensor product Bézier representation of the algebraic function. This in turn leads to different conditions to decide if a single sheet of the algebraic passes through a cube. Roughly speaking, if all columns in the tensor product are one turn and if the derivatives in the column direction are of one sign, then one sheet of the algebraic passes through the cube.

In their paper, Alberti et al. give four examples. Three of these have singularities, while the fourth has no singularities. On the the tangle cube,

$$x^4 - 5x^2 + y^4 - 5y^2 + z^4 - 5z^2 + 11.8 = 0,$$

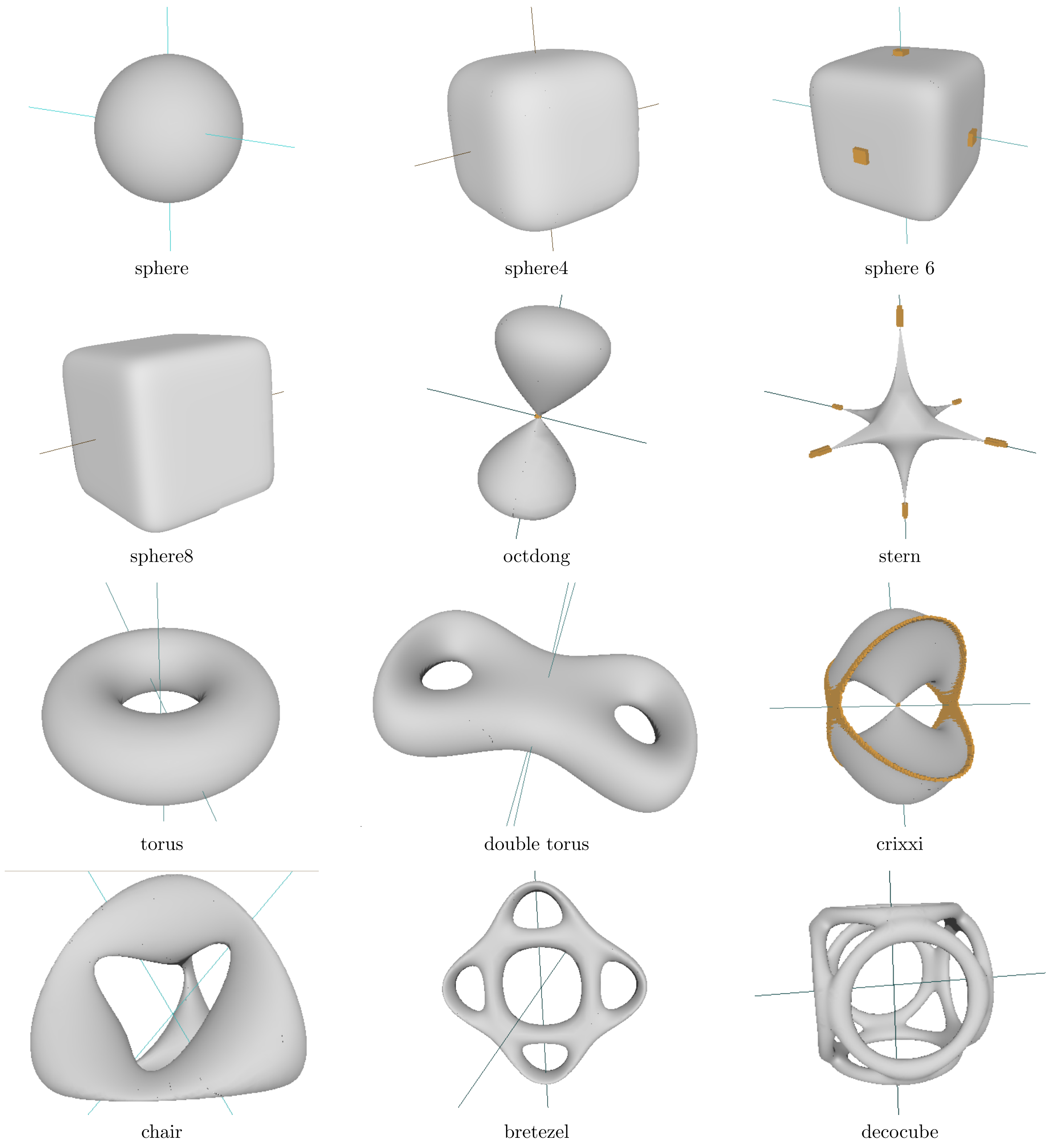

Figure 24: Tessellations generated using the relaxed single sheet criteria.

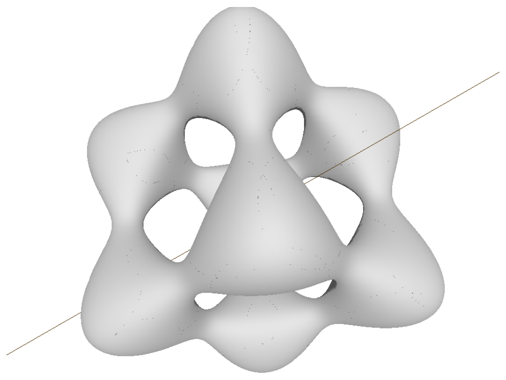

Figure 25: Tangle cube

the surface with no singularities, their method subdivides to 4165 cells, while my method subdivides to 777 cells (Figure 25). However, a direct comparison is difficult, since (a) I further divide each cell into five tetrahedron; (b) the Alberti et al. paper is a bit vague on some details such as the initial cube size, so I had to guess an initial cube size; and (c) Alberti et al. have two epsilons, one of which appears to be a maximum cube size. If they increased this maximum, then they might get a similar number of cell divisions as my method did. To do a fair comparison, a common code base would be needed.

On the surfaces with singularities, both methods subdivide the space around the singularities until the minimum threshold is reached. Since most of the cells are in the region of singularities, my method and Alberti et al.'s method produce a similar number of cells, although again an exact comparison is difficult for the reasons mentioned above.

Computationally, their method has one large advantage: when they subdivide a cell, they can do an 8-1 subdivision on the trivariate tensor product representation using repeated curve subdivision. My method requires doing a change of basis for each of the new tetrahedron in the subdivided cells, which is a more expensive computation. However, the actual comparison of costs likely depends on how deep each algorithm needs to subdivide before tessellating.

Note also that in their paper, they mention

> ...the univariate Bernstein representation also extends to a so-called triangular Bernstein basis. This representation can also be used in our approach, but we will concentrate on the tensor product representation.

My work can be considered this use of the "so called" triangular Bernstein basis.

Finally, in their paper, they prove the termination, correctness, and complexity bounds of their algorithm. Much of their proofs apply to my work, although the details would need to be investigated before making any specific claims.

7 Implementation Details

The following sections contain some implementation details. Most of the details here are not new, but I include these details for completeness and to document what I did.

7.1 Change of Basis

The change of basis algorithm I used to convert the algebraic to Bernstein form is based on the following: if we evaluate our Bernstein basis functions at a particular point and weight them by the unknown coefficients, then the result should be the algebraic function evaluated at this particular point:

$$\sum s_{\vec{i}} B^n_{\vec{i}}(P_0) = F(P_0).$$

If we do this at a number of locations P_i equal to the size of the polynomial space, then we will have a system of equations and unknowns:

$$Bs = f$$

Inverting B and multiplying on both sides yields

$$s = B^{-1} f.$$

To ensure a stable matrix for inversion, we can locate the P_i uniformly through the triangle (tetrahedron) over which we are working. This matrix has a reasonable condition number; for example, for the degree 12 two dimensional case, the condition number (as reported by Octave) is 23,083. Note further that since the Bernstein polynomials are computed in Barycentric form, then the actual domain triangle in 2D Euclidean space is irrelevant and its coordinates disappear from the computation. E.g., for the triangular case, we can use $(i/n, j/n, k/n)$ with $i, j, k \geq 0$ and $i + j + k = n$ as the Barycentric coordinates of our points of evaluation. This has the further advantage that we need only construct and invert B once for each degree we are working with, independent of the particular triangle or algebraic function we are working with.

7.2 Root Finding

Most of the time for this algorithm is spent in root finding when tessellating the curve or surface. I used a modified Regula Falsi root finder, where it did two linear steps followed by one bisection step, and then repeated until convergence.

There is a way to speed up the root finding significantly, though. For curves, we need to root find along a line through a triangle. Rather than do a bivariate de Casteljau evaluation for every step of the root finder, we can first do a 2-1 subdivision of the patch, extract the boundary edge between the two patches, and then do univariate root finding along this edge. For surfaces, we do a similar thing; do a 3-1 subdivision of the tetrahedral volume, extract the common edge and do univariate root finding along this edge.

Note that root finders for univariate Bernstein bases are well studied and that I did nothing new here; see for example [MRR05].

7.3 Testing for A-patch Format

At first glance, testing whether a patch is in A-patch format is a daunting task: in the curve case, you have to test in one of three directions, and in the surface case, you have seven possible directions in which to test. The key to simplifying the algorithm is to separate it into two tasks: first, classify each row as “all zero”, “all positive”, “all negative”, or “mixed sign”, based on the coefficients in the row. Then scan the classifications to test for the A-patch format.

As a first step, you must determine in which direction you should arrange the rows. This is easily determined by testing the corner coefficients. Note that if all the corners have the same sign, you should test to see if all the coefficients are of that sign (or zero), in which case the algebraic does not pass through the triangle/tetrahedron and no longer needs to be considered.

Once the row direction is determined, you can easily scan the rows to determine which of the above four categories each row belongs. You can now scan the row classification to see if the you have a separating layer between positive and negative coefficients.

```
    for(int i=0; i<=deg; i++){
        if ( coefficients[i][0] > 0 ) {
            rows[i] = POS;
        } else if ( coefficients[i][0] < 0 ) {
            rows[i] = NEG;
        } else {
            rows[i] = ZERO;
        }
        for(int j=1; j<=deg - i; j++){
            if ( coefficients[i][j] > 0 && rows[i] == NEG ) {
                rows[i] = MIXED;
                break;
            } else if ( coefficients[i][j] < 0 && rows[i] == PLUS ) {
                rows[i] = MIXED;
                break;
            } else if ( coefficients[i][j] > 0 && rows[i] == 0 ) {
                rows[i] = POS; // Call 0's positive
            } else if ( coefficients[i][j] < 0 && rows[i] == 0 ) {
                rows[i] = NEG; // Call 0's negative
            }
        }
    }
```

8 Code Discussion

A few notes and observations are in order about the code. First, there is a routine to evaluate the polynomial (in monomial form). This routine is called $\#\Pi_k$ times (where $\#\Pi_k$ is the size of a degree n polynomial space for the dimension with which we are working) when doing basis conversion. The original routine was written using `pow`. Converting this routine to compute and storing in linear arrays the monomials $1, x, ..., x^k$, $1, y, ..., y^k$ and (in the 3D case) $1, z, ..., z^k$, and then calling these in place of calls to `pow` gave a factor of 4 speed-up to the code. Note that for degree 12 trivariate polynomials that $\#\Pi_k = 455$.

Second, when doing the root finding, doing a 2-1 split in the 2D case sped up the root finding by a factor of 10. This was a bit higher than expected, and is likely due to the new root finder using C arrays rather than the C++ std::vector class. However, the 2D case executes fast enough that this was not a significant improvement. In the 3D case, however, doing a 3-1 split and then root finding on the univariate Bézier curve is expected to yield a factor of 7 speed-up in the root finding. In practice, this resulted in roughly a factor of 3 speed-up, since the change of basis is also a significant cost in the algorithm.

With these efficiency improvements, the bottleneck in the code is in the change of basis. A gprof of the code executing on the decocube example shows that 58% of the time is spent in evaluating the coefficients for doing the change of basis computation, while only 18% of the time was spent in root finding. Exact ratios between these numbers will depend on many factors, such as how deep we need to subdivide to decide whether to tessellate the A-patches within a cube or if the surface does not pass through the cube; the number of triangles into which each A-patch is tessellated; the degree of the surface; and the ratio of empty cells to cells containing the surface. Regardless, it is clear that the change of basis computation and the root finding are the two most expensive components of the code.

In terms of code complexity, testing the Bernstein representations to see if they are in A-patch format (or if they meet the additional single sheet conditions) is fairly complex code. However, part of the complexity here is that this was experimental code, and multiple variations of this routine were required for comparisons.

Name	Formula	Lopes	A-patches	Additional Single Sheet
Circles	$(x^2+y^2-1)(x^2-y^2-0.02)=0$	341	106	58
Pear	$4y^2+2x^3+x^4-2x-1=0$	237	170	58
Clown Smile	$(y-x^2+1)^4+(x^2+y^2)^4-1=0$	709	3414	54
Ellipse	$x^2+6y^2-6=0$	[160]	42	42
Cubic	$y^2-x^3+x=0$	[54]	38	38
Bicorn	$y^2(a^2-x^2)-(x^2-2ay-a)^2=0$	453	710 (?)	630 (19)
Cubic2	$y^2-x^3+x-0.5=0$	709	82	66
Taubin	(see text)	4505	302	130

Table 1: Comparison on Lopes et al. curves. Numbers in [] were manually counted from figure; numbers in () were number of cells in which method did not converge

Additionally, there are three possible configurations for 2D A-patches (curves), and seven possible configurations for 3D A-patches (surfaces). While some code reuse is possibly between the cases (e.g., all 10 cases use the same `testRows` routine to check if the rows have the appropriate sign mix), much of the code had to be written separately for each case. While using intelligent iterators might allow a reduction from 10 cases down to 3 cases (one for curves, one for 3-sided surfaces, and one for 4-sided surfaces), it is likely that the iterator will confuse the function of the code.

A Formulas

Table 1 gives the formulas used in the examples, as well as the number of cells into which the three methods (Lopes et al., the basic A-patch method, and the extended A-patch method with the new single sheet condition) subdivide the region of space.

The Taubin example [Tau94] has the following formula:

$$\begin{aligned} &0.110x - 0.177y - 0.174x^2 + 0.224xy - 0.303y^2 - 0.168x^3 + 0.327x^2y - 0.087xy^2 \\ &\quad -0.013y^3 + 0.235x^4 - 0.667x^3y + 0.745x^2y^2 - 0.029xy^3 + 0.072y^4 + 0.004 \quad = \quad 0 \end{aligned}$$

B Additional experiments

In Table 3, I give the number of cubes used at each subdivision level for various surfaces using both the A-patch conditions and the new conditions. The minimum edge size was 0.04 unless otherwise noted, where (when using A-patch conditions) the program ran out of memory with an edge size of 0.04. A graphical depiction of these subdivision levels appears in Figure 26.

References

[ACM05] L. Alberti, G. Comte, and B. Mourrain. Meshing implicit algebraic surfaces: the smooth case. In L.L. Schumaker, M. Maehlen, and K. Morken, editors, *Mathematical Methods for Curves and Surfaces: Tromso '04*. Nashboro, 2005.

[BCX95] Chandrajit L. Bajaj, Jindon Chen, and Guoliang Xu. Modeling with cubic A-patches. *ACM Transactions on Graphics, Volume 14, Issue 2*, pages 103–133, 1995.

Name	Formula	Initial Cube Size
Hyperboloid	$x^2 + y^2 - z^2 - 0.8 = 0$	$2 \times 2 \times 2$
Cone	$x^2 + y^2 - z^2 = 0$	$2 \times 2 \times 2$
Sphere	$x^2 + y^2 + z^2 - 1 = 0$	$2 \times 2 \times 2$
Sphere4	$x^4 + y^4 + z^4 - 1 = 0$	$2 \times 2 \times 2$
Sphere6	$x^6 + y^6 + z^6 - 1 = 0$	$2 \times 2 \times 2$
Sphere8	$x^8 + y^8 + z^8 - 1 = 0$	$2 \times 2 \times 2$
Octdong	$x^2 + y^2 + z^4 - z^2 = 0$	$2 \times 2 \times 2$
Stern	$400(x^2y^2 + y^2z^2 + x^2z^2) + (x^2 + y^2 + z^2 - 1)^3 = 0$	$2 \times 2 \times 2$
McMullen	$(x^2 + 1)(y^2 + 1)(z^2 + 1) + 8xyz - 2 = 0$	$2 \times 2 \times 2$
Torus	$(x^2 + y^2 + z^2 + R^2 - r^2)^2 - 4R^2(x^2 + y^2) = 0$	$3 \times 3 \times 3$
Double Torus	$x^8 - 2x^6 + x^4 + 2x^4y^2 - 2x^2y^2 + y^4 + z^2 - 0.04 = 0$	$2 \times 2 \times 2$
Crixxi	$(y^2 + z^2 - 1)^2 + (x^2 + y^2 - 1)^3 = 0$	$2 \times 2 \times 2$
Chair	$(x^2 + y^2 + z^2 - 0.95 \cdot 5^2)^2 - 0.8[(z - 5)^2 - 2x2][(z + 5)^2 - 2y^2] = 0$	$6 \times 6 \times 6$
Bretzel	$((x^2 + y^2/4 - 1)(x^2/4 + y^2 - 1))^2 + z^2 - 0.075 = 0$	$3 \times 3 \times 3$
Decocube	$((x^2 + y^2 - 0.8^2)^2 + (z^2 - 1)^2) \cdot ((y^2 + z^2 - 0.8^2)^2 + (x^2 - 1)^2) \cdot$ $((x^2 + z^2 - 0.8^2)^2 + (y - 1)^2) - 0.02 = 0$	$6 \times 6 \times 6$
Tangle Cube	$x^4 - 5x^2 + y^4 - 5y^2 + z^4 - 5z^2 + 11.8 = 0$	$4 \times 4 \times 4$

Table 2: Torus has $R = 1$, $r = 0.2$

Surface	A-patch conditions	Extended conditions
Cone	1 8 64 80 80 80 80 80 0 = 473	1 8 64 80 80 80 80 80 0 = 473
Torus	1 8 64 184 376 920 2224 7304 21904 0 = 32985	1 8 64 64 256 0 = 393
Double Torus	1 8 64 128 320 808 3416 13376 0 = 18121	1 8 64 64 192 0 = 329
Bretzel	1 8 64 256 544 1312 0 = 2185**	1 8 64 256 448 688 736 480 48 0 = 2729
Decocube		1 8 64 64 64 384 960 392 784 8 0 = 2729

Table 3: Comparison of subdivision depth for A-patch condition and extended A-patch conditions. Minimum cube edge size of 0.04 used, ** except for A-Patch conditions for Bretzel (0.2).

Surface	Triangles	Cells	Time
Cone	39,816	473	3 seconds
Torus	16,384		5 seconds
Double torus	18,432		7 seconds
Bretzel	109,376	2729	50 seconds
Decocube	106,576	2729	130 seconds

Table 4: Sizes of triangulations, number of cells, time to compute

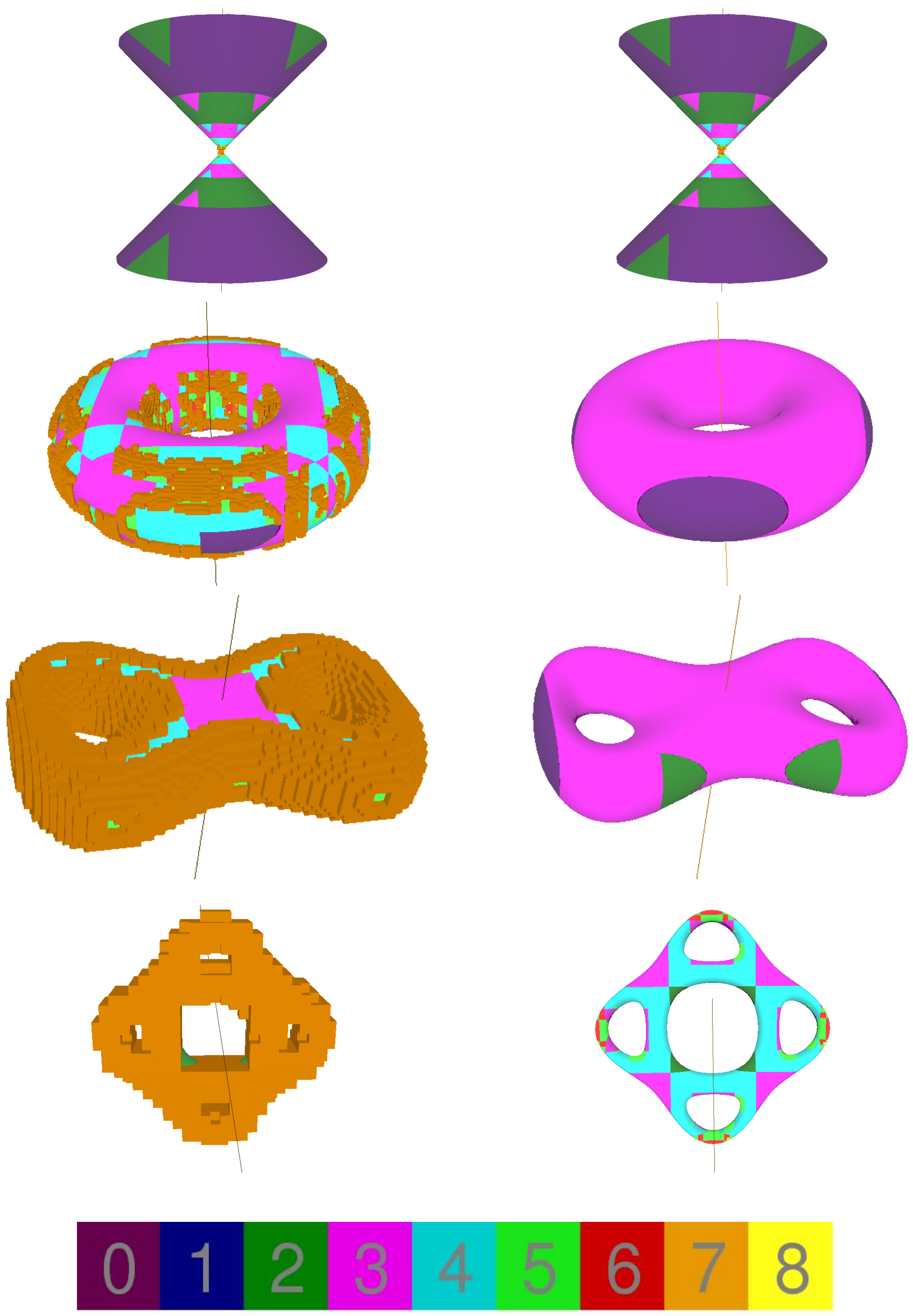

Figure 26: Comparison of depth of recursion between A-patch conditions and new conditions. Bottom: colour map showing depth of recursion.

[BG93] Phillip Barry and Ronald Goldman. Algorithms for progressive curves. In R. Goldman and T. Lyche, editors, *Knot Insertion and Deletion Algorithms for B-spline Curves and Surfaces*, chapter 2, pages 11–63. SIAM, 1993.

[Gal00] Jean Gallier. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms.* Morgan-Kaufmann, 2000.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics, Volume 21, Number 4, July 1987*, pages 163–169, 1987.

[LOdF02] H. Lopes, J. B. Oliveira, and L. H. de Figueiredo. Robust adaptive polygonal approximation of implicit curves. *Computers and Graphics*, 26, 2002.

[Luk08] Curtis Luk. Tessellating algebraic curves and surfaces using A-Patches. Master's thesis, University of Waterloo, 2008.

[Man06] Stephen Mann. *A Blossoming Approach to Splines.* Morgan-Claypool, 2006.

[MRR05] B. Mourrain, F. Rouillier, and M.-F. Roy. Bernstein basis and real root isolation. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, volume 52 of *Mathematical Sciences Research Institute Publications*, pages 459–478. Cambridge University Press, 2005.

[NY06] Timothy Newman and Hong Yi. A survey of the marching cubes algorithm. *Computer and Graphics*, 30(5):854–879, October 2006.

[Pet94] Jorg Peters. Evaluation and approximate evaluation of multivariate Bernstein form on a regularly partitioned simplex. *ACM Transactions on Mathematical Software*, 20(4):460–480, 1994.

[PLLdF06] A. Paiva, H. Lopes, T. Lewiner, and L.H. de Figueiredo. Robust adaptive meshes for implicit surfaces. *Computer Graphics and Image Processing, 2006. SIBGRAPI '06. 19th Brazilian Symposium on*, pages 205–212, Oct. 2006.

[PT90] Bradley Payne and Arthur Toga. Surface mapping brain function on 3D models. *Computer Graphics and Applications*, 10(5):33–41, September 1990.

[ST90] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. In *Computer Graphics*, pages 63–70, 1990.

[Tau94] Gabriel Taubin. Distance approximations for rasterizing implicit curves. *ACM Transactions on Graphics*, 13(1):3–42, January 1994.

[WvG92] J Wilhelms and A van Gelder. Octrees for faster isosurface generation. *ACM Transcations on Graphics*, 11(3):201–227, 1992.