1994

Converting a Rational Surface to a Standard Rational Bernstein-Bezier Surface

Chandrajit Bajaj

Guoliang Xu

Report Number:
94-044

Converting a Rational Surface to a Standard Rational Bernstein-Bézier Surface

Chandrajit Bajaj and Guoliang Xu
Computer Sciences Department
Purdue University
West Lafayette, IN 47907

CSD-TR-94-044
June, 1994

Converting a Rational Surface to a Standard Rational Bernstein-Bézier Surface

Chandrajit Bajaj
Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398, USA
bajaj@cs.purdue.edu

Guoliang Xu
Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398, USA
xuguo@cs.purdue.edu

June 20, 1994

Introduction

When geometric designers and engineers use an RBB (Rational Bernstein-Bézier) surface given by $R(U) = \sum_{|I|=n} w_I b_I B_I^n(U) / \sum_{|I|=n} w_I B_I^n(U)$ with $I = (i, j, k)$, $U = (u, v, w)$ and $B_I^n(U) = \binom{n}{I} u^i v^j w^k$ on a triangle $\Delta abc \subset \mathbb{R}^2$, as a geometric modeling tool, they usually assume that the coefficients of denominator polynomial are positive[4, 3]. This assumption is quite strong, but rids the surface of real poles (real roots of the denominator polynomial) and gives the RBB surface its convex hull property. For most of the known methods for generating rational curve and surface approximations [2] one can guarantee that the rational functions have no poles (positive denominator), however the RBB form may quite easily have non-positive coefficients. Therefore, it is nice to have an algorithm to convert a RBB curve or surface with no real poles into a standard RBB representation where the denominator has only positive coefficients. In [1], we present an algorithm to convert a rational curve into to a standard RBB representation. In this paper, we present a subdivision based algorithm to convert a rational surface defined over the triangle Δabc in $\mathbb{R}^2$ into a finite number of C^∞ standard RBB surface patches. We use this conversion algorithm as a final step in our NURBS approximation of algebraic surfaces[2].

Subdivision of the Domain Triangle

We assume that the given rational surface $R(U)$ is in RBB form and has no poles in its defining triangle $\Delta abc \subset \mathbb{R}^2$. Several methods are known for converting a rational surface into the RBB form[3]. Our goal here is to transform it into a standard RBB form with positive denominator coefficients. It then suffices to consider the conversion of the denominator Bernstein-Bézier (BB) polynomial $P(U) = \sum_{|I|=n} w_I B_I^n(U)$ of degree n, with $I = (i, j, k)$, $U = (u, v, w)$ and $B_I^n(U) = \binom{n}{I} u^i v^j w^k$ defined on the triangle $\Delta abc \subset \mathbb{R}^2$. As the rational surface has no real poles, without loss of generality, $P(U) > 0$ for any $U \in \Delta abc$.

The first step is to subdivide the triangle Δabc such that the BB representation of $P(U)$ on the edges of each subtriangle has nonnegative coefficients. This can be achieved by connecting the center of the triangle with its vertices and the break points on the edges (see Figure 1(a)). These break points on the edges are determined using the one-dimensional conversion technique [1], applied to $P(U)$ restricted to the edge (a one-dimensional BB polynomial). This step is repeated if $P(U)$ has negative coefficients on the newly added edges interior to triangle Δabc. Note that this repeated subdivision is bounded since $P(U) > 0$ and so are its coefficients on any sufficiently small sized triangle.

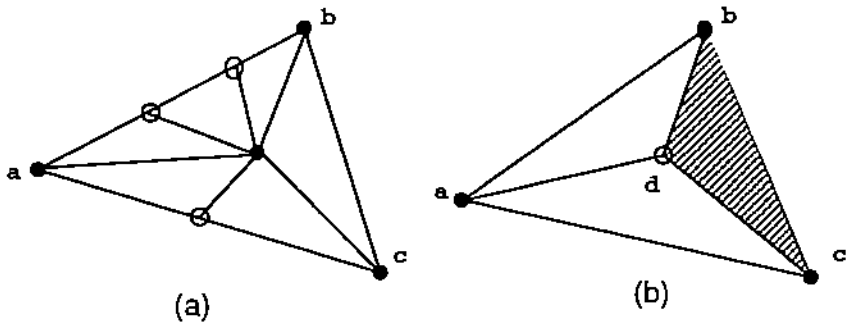

Figure 1: (a) Subdivide the Edges; (b) Subdivide the Triangle

In the next step, we assume $P(U)$ has nonnegative coefficients on all the edges (including interior) of Δabc. Let d be a point in the triangle whose barycentric coordinate is $V = (x, y, z)$, i.e. $d = xa + yb + zc$ and $x + y + z = 1$. Now we want to choose d such that the representation of $P(U)$ on the new triangle Δdbc (the shaded part of Figure 1(b)): $P(U) = \tilde{P}(\tilde{U}) = \sum_{|I|=n} \tilde{w}_I B_I^n(\tilde{U})$ has nonnegative coefficients, where U and $\tilde{U}$ represent the same point in Δdbc in the old and new barycentric coordinate system, respectively. Then $\tilde{w}_{(i,j,k)} = w^i_{(0,j,k)}(V), i = 0, 1, \ldots, n$ where $w^i_{(0,j,k)}(V) = \sum_{|J|=i} w_{(0,j,k)+J} B^i_J(V), j + k = n - i$, i.e., $\tilde{w}_{(ijk)}(V)$ is a Bernstein polynomial in V of degree i. For subdivision formulas such as above the reader may also consult [3]. Now we choose d such that every $\tilde{w}_{(i,j,k)}(V)$ is nonnegative, and at the same time $y + z$ is as small as possible in order to have the fewest number of pieces in the subdivision. Let $f_I(y, z) = \tilde{w}_{(i,j,k)}(1 - y - z, y, z)$. Then we need to find (y, z) such that $\begin{cases} f_I(y, z) = 0 \\ y + z \text{ as large as possible} \end{cases}$. Let $X = y, \quad Y = y + z, \quad F_I(X, Y) = f_I(y, z)$. Then the above problem is equivalent to determining (X, Y) such that $\begin{cases} F_I(X, Y) = 0 \\ Y = \text{maximum} \end{cases}$. This (X, Y) can be obtained by solving the following equation

$$\begin{cases} F_I(X, Y) = 0 \\ \frac{\partial F_I}{\partial Y}(X, Y) = 0 \end{cases} \tag{1}$$

Among all the solutions of (1) for all I, $|I| = n$, take the one that makes $y + z$ as large as possible and

$$y \geq 0, \quad z \geq 0, \quad y + z \leq 1. \tag{2}$$

We can then obtain the barycentric coordinate of $d = (1 - y - z, y, z)$. If (1) has no solution to satisfy (2) for some I, $|I| = n$, then $\tilde{w}_I$ is nonnegative on Δabc.

After d is determined in this way, $P(U)$ will have nonnegative coefficients on Δdbc. For the remaining two triangles Δabd and Δadc, we repeat the above two steps. This procedure will terminate at some stage because of $P(U) > 0$. The final subdivision will be as shown in Figure 2.

A special choice of d is to select it on the the line $(1 - \alpha)y - \alpha z = 0$ for fixed $\alpha \in [0, 1]$. In this case we solve the equation

$$\begin{cases} f_I(y, z) = 0 \\ (1 - \alpha)y - \alpha z = 0 \end{cases} \tag{3}$$

Choose a solution that makes $y + z$ larger and also satisfies (2). If $\alpha = 0$ or $\alpha = 1$, then d lies on the edge $[a, c]$ or $[a, b]$, respectively. The final subdivision will be as shown in Figure 3. If we choose $\alpha = 1/2$, the subdivision will be as shown in Figure 2. The system of equations (3) is much easier to handle than the system of equations (1), since the second polynomial in the latter system is linear.

Unlike the one-dimension problem, there are many more possibilities to choose the subdivision points. An open problem that remains is the optimal way to choose the break points such that the partition has the fewest pieces.

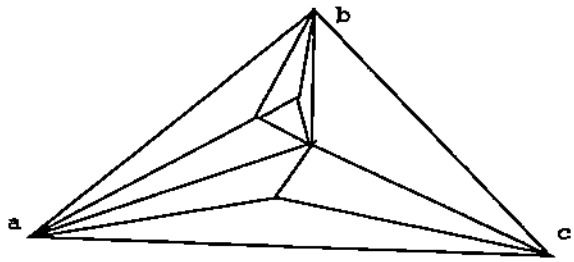

Figure 2: Subdivision in the Interior

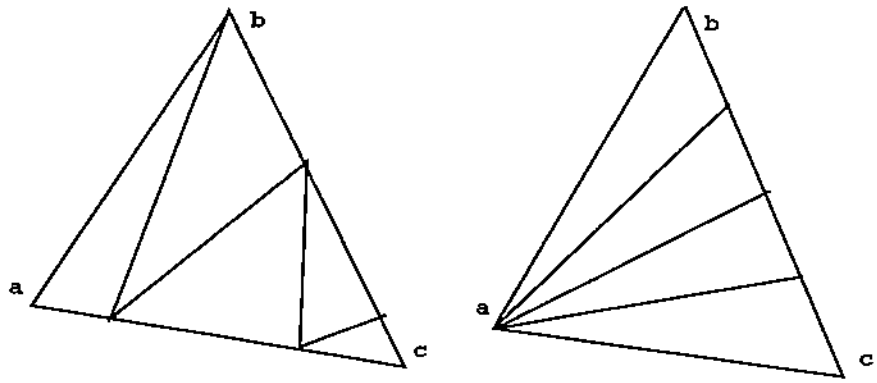

Figure 3: Subdivision on the Edges

Finally, it should be noted that if we perturb each of the break points of the subdivision by a small amount in the proper direction, then we can make all the coefficients of $P(U)$ to be positive.

Example 0.1 *We now give an example of this conversion for a denominator polynomial.*

The original cubic polynomial is defined on the domain triangle [(0,0),(1,0),(0,1)] with the following RBB representation :

$$\begin{array}{lll} b_{300} = 0.700000 & b_{201} = 1.100000 & b_{102} = 1.300000 \\ b_{003} = 1.000000 & b_{210} = 0.450000 & b_{111} = -0.50000 \\ b_{012} = 1.200000 & b_{120} = 0.300000 & b_{021} = 1.440000 \\ b_{030} = 1.000000 & & \end{array}$$

The subdivision of the original domain triangle is done at the point (0.33333,0.333333) to give the three subpolynomials defined over the three subtriangles. The first subpolynomial defined on [(0,0),(1,0),(0.33333,0.333333)] is given by

$$\begin{array}{lll} b1_{300} = 0.700000 & b1_{201} = 0.750000 & b1_{102} = 0.488889 \\ b1_{003} = 0.632222 & b1_{210} = 0.450000 & b1_{111} = 0.083333 \\ b1_{012} = 0.570000 & b1_{120} = 0.300000 & b1_{021} = 0.913333 \\ b1_{030} = 1.000000 & & \end{array}$$

The second subpolynomial defined on [(1,0),(0,1),(0.33333,0.333333)] is given by

$$b2_{300} = 1.000000 \quad b2_{201} = 0.913333 \quad b2_{102} = 0.570000$$

$$b2_{003} = 0.632222 \quad b2_{210} = 1.440001 \quad b2_{111} = 0.713333$$
$$b2_{012} = 0.837778 \quad b2_{120} = 1.200000 \quad b2_{021} = 1.166667$$
$$b2_{030} = 1.000000$$

The third subpolynomial defined on [(0,0),(0.33333,0.333333),(0,1)] is given by

$$b3_{300} = 0.700000 \quad b3_{201} = 1.100000 \quad b3_{102} = 1.300000$$
$$b3_{003} = 1.000000 \quad b3_{210} = 0.750000 \quad b3_{111} = 0.633334$$
$$b3_{012} = 1.166667 \quad b3_{120} = 0.488889 \quad b3_{021} = 0.837778$$
$$b3_{030} = 0.632222$$

Psuedocode of the Algorithm

The above algorithm is given below in psuedocode with code comments given in italics.
Main program of the conversion algorithm
Triangle *is the input triangle, a 3×2 array containing the coordinates of the three vertices of the triangle*
Coeffs *is the input denominator polynomial coefficient array over the triangle*
TriangleCoeffs *is the output triangle-coefficient list with positive coefficients. The element* TriangleCoeff = TriangleCoeffs[i] *has two parts. One part is the triangle, denoted as* TriangleCoeff → triangle. *The other part is the coefficients, denoted as* TriangleCoeff → coeff

```
TriangleCoeffs = NULL
TriCoeffs = NULL
call BREAKEDGES(Triangle, Coeffs, TriCoeffs)
TriCoeff = TriCoeffs(0)
while (TriCoeff ≠ NULL)
    Tri = TriCoeff → triangle
    Coeff = TriCoeff → coeff
    call BREAKFACE(Tri, Coeff, TriangleCoeffs)
    next TriCoeff
end while
return
```

Procedure to subdivide the given triangle into subtriangles so that the coefficients on each edge of each subtriangle are positive
procedure BREAKEDGES(Triangle, Coeffs, TriangleCoeffs)
Triangle *is the input triangle, a 3×2 array*
Coeffs *is the input coefficients array over the triangle*

```
TriangleCoeffs is the output triangle-coefficient list
    for i = 1 to 3 step 1
        get the coefficients on the edge
        Coeffs(i) = The coefficients on the i-th edge
        call BREAKINTERVAL(Coeffs(i), BreakPoints(i), NumBreakPts(i))
    next i
    Form the Polygon
    call TRIANGULATION(Polygon, Triangles)
    Tri = Triangles(0)
    while (Tri ≠ NULL)
        call BEZIERCOEFF(Triangle, Coeffs, Tri, NewCoeff)
        if (NewCoeff are positive on edges) then
            Append Tri and NewCoeff to the TriangleCoeffs list
        else
            recursive calling
            call BREAKEDGES(Tri, NewCoeff, TriangleCoeffs)
        end if
        next Tri
    end while
    return
```

Procedure to subdivide the given triangle into subtriangles so that the coefficients on each subtriangle are positive. Here we assume the input coefficients are positive on the edges.

```
procedure BREAKFACE(Triangle, Coeffs, TriangleCoeffs)
Triangle is the input triangle Δabc, a 3×2 array
Coeffs is the input coefficients array over the triangle
```

TriangleCoeffs *is the output triangle-coefficient list*
compute the subdivision point d
Form the equation (1)
Solve the equation (1)
Find the point d
Form three triangles:
Triangle1 = Δabd
Triangle2 = Δacd
Triangle3 = Δdbc
call BEZIERCOEFF(Triangle, Coeffs, Triangle1, NewCoeff1)
call BEZIERCOEFF(Triangle, Coeffs, Triangle2, NewCoeff2)
call BEZIERCOEFF(Triangle, Coeffs, Triangle3, NewCoeff3)
Append Triangle3 and NewCoeff3 to the TriangleCoeffs list
if (NewCoeff1 are positive on Triangle1) **then**
Append Triangle1 and NewCoeff1 to the TriangleCoeffs list
else
recursive calling
call BREAKFACE(Triangle1, NewCoeff1, TriangleCoeffs)
end if
if (NewCoeff2 are positive on Triangle2) **then**
Append Triangle2 and NewCoeff2 to the TriangleCoeffs list
else
recursive calling
call BREAKFACE(Triangle2, NewCoeff2, TriangleCoeffs)
end if
return

Procedure to subdivide interval [0,1] so that the coefficients on each subinterval are positive. This is done using the one dimension conversion algorithm given in [1]. We skip the details here, and refer the reader to the psuedocode in [1].
procedure BREAKINTERVAL(Coeffs, BreakPoints, NumBreakPoints)
Coeffs *is the input coefficients array of the BB form over the interval [0,1]*
BreakPoints *is the output break point array*
NumBreakPoints *is the output number of break point*
.....

Procedure to produce the BB form coefficients over a new triangle from the BB form over the original triangle. The new BB form can be obtained by interpolating the original BB form at the regular points of the new triangle, or by using the subdivision formulas). We again omit the details here.
procedure BEZIERCOEFF(OldTriangle, OldCoeff, NewTriangle, NewCoeff)
OldTriangle *is the input old triangle, a 3×2 array*
OldCoeff *is the input old coefficients array over the old triangle*
NewTriangle *is the input new triangle, a 3×2 array*
NewCoeff *is the output new coefficients array over the new triangle*
.....

Procedure to produce triangulations from a polygon by connecting each vertex with the center of the polygon or iteratively chording the sharpest angle or using the reader's favourite scheme.
procedure TRIANGULATION(Polygon, Triangles)
Polygon *is the input polygon, a two dimensional array*
Triangles *is the output triangle list*
.....

References

[1] C. Bajaj and G. Xu. Converting a Rational Curve to a Standard Rational Bernstein-Bézier Representation. In Paul S. Heckbert, editor, Graphics Gems IV, pages 256–260. Academic Press, 1994.

[2] C. Bajaj and G. Xu. Rational spline approximations of real algebraic curves and surfaces. In H.P. Dikshit and C. Michelli, editors, Advances in Computational Mathematics, pages x–x. World Scientific Publishing Co., 1994.

[3] G Farin. *Curves and Surfaces for CAGD: A Practical Guide.* Academic Press, Boston, 1993.

[4] L. Peigl and W. Tiller. Curve and Surface Construction using Rational B-Splines. *Computer Aided Design*, 19(9):485–498, 1987.