www.cad-journal.net
Computer-Aided Design AND APPLICATIONS

Beta-Bezier Surfaces

Seifalla Moustafa[1], Anatasia Kazadi[1], Fuhua (Frank) Cheng[1], Shuhua Lai[2] and Alice J. Lin[3]

[1]University of Kentucky, seifalla.moustafa@uky.edu, ansm226@g.uky.edu, cheng@cs.uky.edu
[2]Georgia Gwinnett College, slai@ggc.edu
[3]Austin Peay State University, lina@apsu.edu

Corresponding author: Seifalla Moustafa, seifalla.moustafa@uky.edu

Abstract. In this paper, the concept of tension control [1] is developed for Bezier surfaces so that one can reshape a so-called Beta-Bezier surface without moving its control points, a property motivated by NURBS surfaces but can be performed more efficiently and in a friendlier manner with the new surface representation technique. In addition to developing the concept of tension control for Bezier surfaces, an efficient rectangular mesh interpolation scheme for Beta-Bezier surfaces is also developed.

Keywords: Bezier curves, Bezier surfaces, Beta-Bezier Curves, Beta-Bezier Surfaces, tension control, interpolation.
DOI: https://doi.org/10.14733/cadaps.2024.693-704

1 INTRODUCTION

In the 1960s, Pierre Bezier invented Bezier curves which are defined explicitly in terms of the Bernstein basis functions. In the 1980s, a numerically stable algorithm known as de Casteljau's algorithm for evaluating Bezier curves was published. In addition to being numerically stable, De Casteljau's algorithm is widely thought to be more intuitive than the explicit definition of a Bezier curve. The algorithm takes as input an array of n+1 points known as control points and produces as output a point on a Bezier curve of degree n by initially performing linear interpolation (i.e. c = a(1-t) + bt, where a, b, and c are points and t is a real number between 0 and 1) between the adjacent points in the input array and from this point onwards recursively performing linear interpolation between the resulting points n times using the initial value of t. Tensor product surface patches have been around for a long time, but they were probably first considered for use in CAGD modeling in the early 1960s by Paul de Casteljau. A Bezier surface patch is defined as a tensor product patch; that is, a Bézier curve is swept along another Bézier curve.

Bezier surface patches are used in computer graphics and computer-aided design. They satisfy the convex hull property, which makes their shapes easier to control than other types of surfaces. A Bezier surface patch interpolates the four corner points of its control net. This means that complex shapes can be modeled using a piecewise Bezier surface (i.e. multiple patches pieced together). Piecewise Bézier surfaces are superior to triangle meshes as a representation of smooth surfaces. They require fewer points (and thus less memory) to represent curved surfaces, are easier to manipulate, and have much better continuity properties.

If a 3D mesh is to be interpolated and a C^2-continuous piecewise surface is to be produced, the locations of a Bezier surface's control points are determined solely by continuity conditions; that is, we cannot freely move them around. Researchers [2; 8-10] tried to find ways to extend/modify the

definition of a Bezier surface so that one could reshape the surface without moving the control points, but an intuitive and straightforward approach was not available for quite a while.

Thanks to the introduction of the tension control concept to curves and triangular patches [3], [1] was able to invent a Beta-Bezier curve (see equation 1) which (adopting the tensor-product approach) can be used to define the type of surface patches that we propose in this paper: Beta-Bezier patches. Mathematical functions have one or more arguments that are designated in the definition by variables. A function definition can also contain parameters. When parameters are present, the definition actually defines a whole family of functions. Beta is a parameter. Parameters influence the shape of curves in a way that is more complicated than a simple linear transformation. In the case of Beta, the curve flattens out as its value increases. In addition to extending the work of [3] to surfaces, we propose an efficient rectangular mesh interpolation scheme that makes use of Beta-Bezier patches.

$$B_k^n(t;\beta) = \sum_{k=0}^{n} \mathbf{P_k}\binom{n}{k} \frac{\prod_{i=0}^{k-1}(t+i\beta)\prod_{j=0}^{n-1-k}((1-t)+\mathrm{j}\beta)}{\prod_{m=0}^{n-1}(1+m\beta)} \tag{1}$$

In addition to extending the work of [1] to surfaces, we propose an efficient rectangular mesh interpolation scheme that makes use of Beta-Bezier patches. Our scheme yields C2-continuous piecewise Beta-Bezier surfaces which improves on the existing algorithms [5-7] in two ways:

The existing algorithms yield G1-continuous surfaces; ours yields C2-continuous surfaces.

- Surfaces produced by the existing algorithms cannot be reshaped; ours can be.

The rest of the paper is organized as follows. Section 2 provides a little background for readers unfamiliar with Bézier curves. Section 3 gives the definition of a Beta-Bezier surface patch. Section 4 describes our interpolation scheme and derives its time complexity. Section 5 summarizes our work.

2 BACKGROUND

The explicit form of a Bezier curve segment is as follows:

$$B(t) = \sum_{i=0}^{n}\binom{n}{i}t^i(1-t)^{n-i}\boldsymbol{P_i} \tag{2}$$

n is the degree of the curve. The coefficients of the control points (i.e $\binom{n}{i}t^i(1-t)^{n-i}$) are known as the Bernstein basis functions and are denoted by $B_i^n(t)$.

Control point locations are usually affected by external factors. For example, when C^2-continuous interpolation is desired, the locations of a Bezier curve's control points are determined solely by continuity conditions. This necessitated the development of techniques that provide another means for controlling curves' shapes. Beta-Bezier curves have this feature; their shapes can be changed without moving their control points.

A famous function in mathematics is the beta-function: $b(\alpha,\gamma) = \int_0^1 x^{\alpha-1}(1-x)^{\gamma-1}dx$ (3). Originally, in this formula, β was used in place of γ. We use γ because β has a different meaning in this paper. An important property of the beta-function is the recursion property:

$$b(\alpha+1,\gamma) = \frac{\alpha}{\alpha+\gamma}\,b(\alpha,\gamma) \tag{4}$$

$$b(\alpha,\gamma+1) = \frac{\gamma}{\alpha+\gamma}\,b(\alpha,\gamma) \tag{5}$$

Let $\alpha = \lambda t$ and $\gamma = \lambda(1-t)$. Applying (4) k times and then (5) n-k times, we get $b(\lambda t+k, \lambda(1-t)+n-k) = \frac{\lambda t}{\lambda}\frac{\lambda t+1}{\lambda+1}\ldots\frac{\lambda t+k-1}{\lambda+k+1}\cdot\frac{\lambda(1-t)}{(\lambda+k)}\frac{\lambda(1-t)+1}{(\lambda+k+1)}\ldots\frac{\lambda(1-t)+\mathrm{n}-\mathrm{k}+1}{(\lambda+n-1)}\,b(\lambda t,\lambda(1-t)) = \frac{\prod_{i=0}^{k-1}(\lambda t+i)\prod_{j=0}^{n-k-1}(\lambda(1-t)+\mathrm{j})}{\prod_{m=0}^{n-1}(\lambda+m)}\,b(\lambda t,\lambda(1-t))$.
If we multiply this by $\binom{n}{k}$ and divide it by $b(\lambda t,\lambda(1-t))$, we get a function that has a close relationship with the Bernstein polynomial discussed earlier: $B_k^n(t;\lambda) = \binom{n}{k}\frac{\prod_{i=0}^{k-1}(\lambda t+i)\prod_{j=0}^{n-k-1}(\lambda(1-t)+\mathrm{j})}{\prod_{m=0}^{n-1}(\lambda+m)}$. The relationship is $\lim_{\lambda\to\infty}\binom{n}{k}\frac{\prod_{i=0}^{k-1}(\lambda t+i)\prod_{j=0}^{n-k-1}(\lambda(1-t)+\mathrm{j})}{\prod_{m=0}^{n-1}(\lambda+m)} = \binom{n}{k}t^k(1-t)^{n-k}$. This was developed by Zeng et al. [3].

It has been shown [1] that the properties of Beta-Bezier curves are easier to study when $\lambda = \frac{1}{\beta}$. In this case, we have $B_k^n(t;\beta) = \binom{n}{k} \frac{\prod_{i=0}^{k-1}(t+i\beta)\prod_{j=0}^{n-1-k}((1-t)+\mathrm{j}\beta)}{\prod_{m=0}^{n-1}(1+m\beta)}$. A Beta-Bezier curve segment is defined as $C(t;\beta) = \sum_{k=0}^{n} \mathbf{P_k} B_k^n(t;\beta)$. Examples of Beta-Bezier curve segments are shown in figure 1.

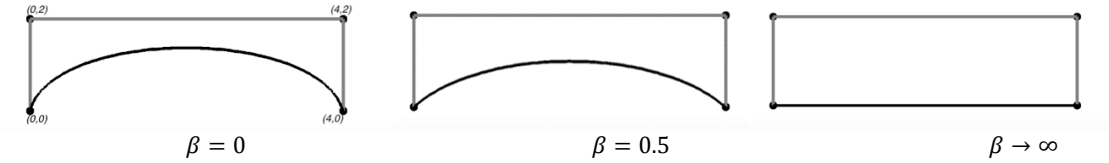


Figure 1: a Beta-Bezier curve segment at different values of Beta.

Note that $\lim_{\beta\to\infty} B_k^n(t;\beta) = \lim_{\beta\to\infty} \binom{n}{k} \frac{\prod_{i=0}^{k-1}(t+i\beta)\prod_{j=0}^{n-1-k}((1-t)+\mathrm{j}\beta)}{\prod_{m=0}^{n-1}(1+m\beta)} = 0$, which means that the curve becomes flatter as β increases. This is shown in the rightmost figure above.

A Beta-Bezier curve segment interpolates its first and last control points. Suppose we have a set of data points that we wish to interpolate. We can use multiple cubic Beta-Bezier curve segments to interpolate these data points. To understand this idea, we consider the following example. Suppose we have three data points D0, D1, and D2 that we wish to interpolate. Let x- and x+ be two cubic Beta-Bezier curve segments with control point sets {P0, P1, P2, P3} and {Q0, Q1, Q2, Q3}, respectively. To interpolate the data points, we connect x- and x+ such that D0 = P0, P3 = Q0 = D1, and Q3 = D2 (see figure 2).

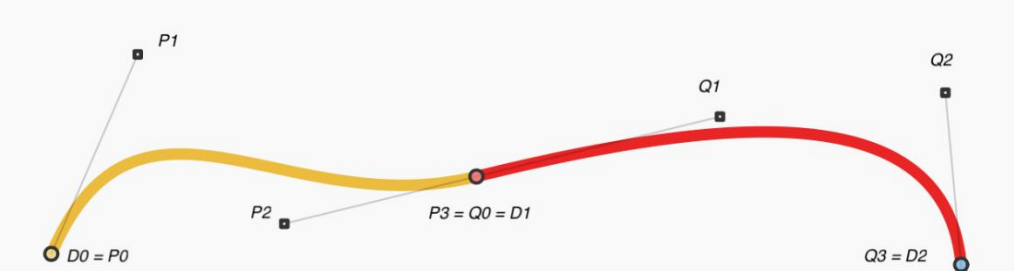


Figure 2: Interpolating three data points with two Bezier curves.

One problem with this approach is that the resulting curve might have a removable discontinuity. Recall that if a curve f is differentiable at a point x0, then f must also be continuous at x0. So, to solve this problem, we set the first and second derivatives equal to each other at the shared point; that is, we set $x'_- = x'_+$ and $x''_- = x''_+$ at P3. Doing so gives us two equations in four unknowns. So, we need two more equations which we can get by setting the second derivatives at P0 and Q3 equal to 0 (since the curve does not have to be continuous at these points). We now generalize this idea to n data points. To interpolate n data points, we need n-1 curve segments, each contributing two unknown control points, for a total of 2(n-1) unknown control points. If we set the first derivatives equal to each other, we have an equation for every two segments, for a total of n-2 equations. Similarly, setting the second derivatives equal to each other gives us n-2 more equations. So, in total, we have 2n-4 equations. This means we need two more equations, which we can get by setting the second derivatives at end data points equal to zero. For a closed curve, we have n segments, each contributing two unknown control points, for a total of 2n unknown control points. If we set the first derivatives equal to each other, we have an equation for every two segments, for a total of n equations. Similarly, setting the second derivatives equal to each other gives us n more equations, for a total of 2n equations. Unlike Bezier curves, Beta-Bezier curves allow us to change the shape of the interpolating curve. It should be noted that each segment can have its own β or there can be a global β. Figures 3 and 4 show examples of Beta-Bezier curves, respectively.

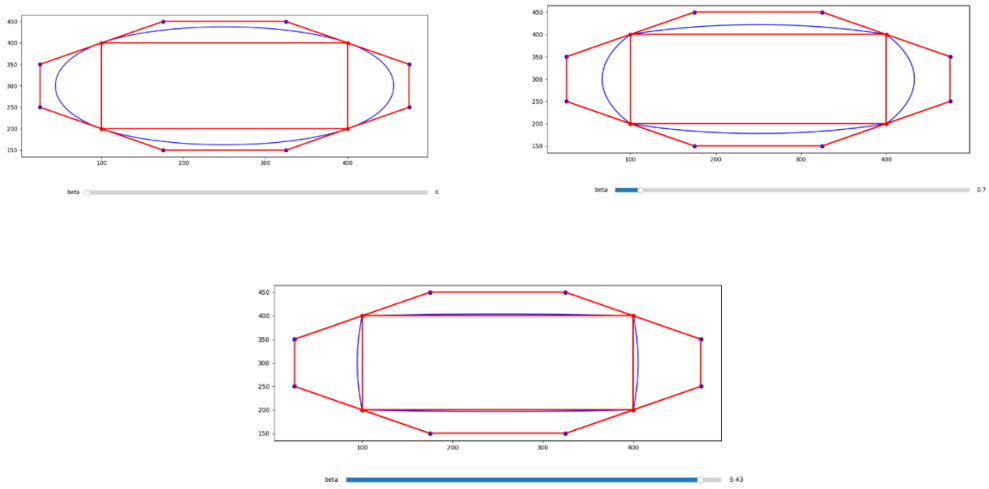


Figure 3: a closed composite Beta-Bezier curves with $\beta = 0$, $\beta = 0.7$, and $\beta = 9.43$.

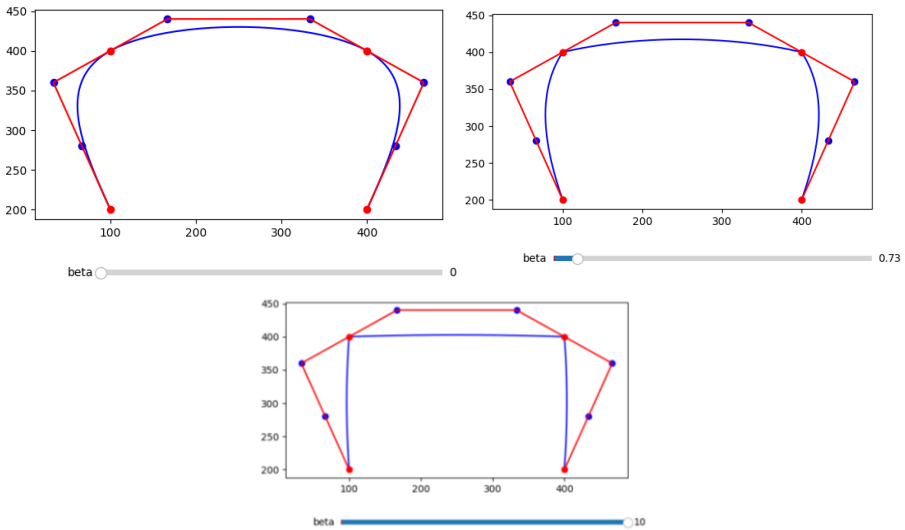


Figure 4: an open composite Beta-Bezier curve with $\beta = 0$, $\beta = 0.73$, and $\beta = 10$.

A Beta-Bezier curve $P(\boldsymbol{u};\boldsymbol{\beta})$ with control points P0, P1, P2, and P3 can be expressed as a cubic Bezier curve in u as follows:

$$P(\boldsymbol{u};\boldsymbol{\beta}) = \sum_{i=0}^{3} Q_i B_i^3(u).$$

Q_0, Q_1, Q_2 and Q_3 can be found as follows:

$$Q_0 = P_0$$

$$Q_1 = \frac{\beta(3+4\beta)}{3(1+\beta)(1+2\beta)}P_0 + \frac{1}{1+2\beta}P_1 + \frac{\beta}{(1+\beta)(1+2\beta)}P_2 + \frac{2\beta^2}{3(1+\beta)(1+2\beta)}P_3$$

$$Q_2 = \frac{2\beta^2}{3(1+\beta)(1+2\beta)}P_0 + \frac{\beta}{(1+\beta)(1+2\beta)}P_1 + \frac{1}{1+2\beta}P_{2,j} + \frac{\beta(3+4\beta)}{3(1+\beta)(1+2\beta)}P_3$$

$$Q_3 = P_3$$

A cubic Beta-Bezier curve segment can be represented as a cubic B-spline curve segment.
We compute $\bar{P}_0$, $\bar{P}_1$, $\bar{P}_2$ and $\bar{P}_3$:

$\bar{P}_1 = Q_1 + (Q_1 - Q_2)$

$\bar{P}_2 = Q_2 + (Q_2 - Q_1)$

$A_1 = P_0 + (P_0 - Q_1)$

$A_2 = P_3 + (P_3 - Q_2)$

$\bar{P}_0 = A_1 + 2(A_1 - \bar{P}_1)$

$\bar{P}_3 = A_2 + 2(A_2 - \bar{P}_2)$

If we define a cubic B-spline curve segment using $\bar{P}_0$, $\bar{P}_1$, $\bar{P}_2$ and $\bar{P}_3$ as its control points as follows,

$CB(t) = \frac{(1-t)^3}{6}\bar{P}_0 + \frac{4-6t^2+3t^3}{6}\bar{P}_1 + \frac{1+3t+3t^2-3t^3}{6}\bar{P}_2 + \frac{t^3}{6}\bar{P}_3$

, it can be proven through straight forward algebra that CB(t) = $P(\boldsymbol{u};\boldsymbol{\beta})$. The knot vector is [0, 0, 0, 1, 1, 1].
Figure 5 shows an example of representation conversion.

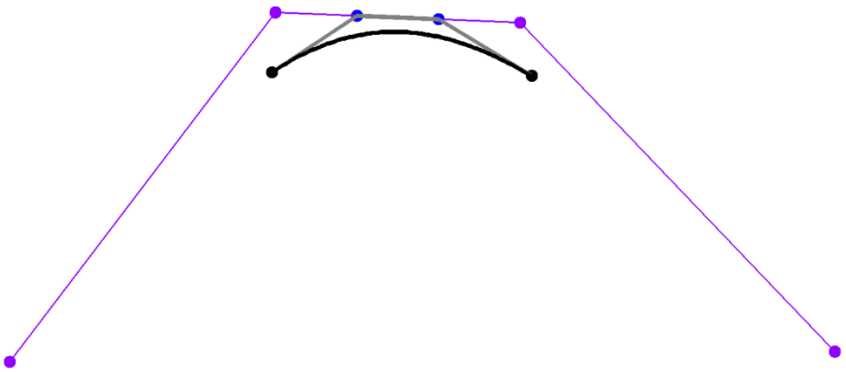


Figure 5: the purple polygon is the B-spline control polygon. The gray polygon is the Bezier control polygon.

A Bezier surface patch is defined as the locus of a moving, deforming Bezier curve segment. An important assumption here is that each control point $\boldsymbol{P_i(u)}$ moves along a Bezier curve which has its own control points. The equation of a moving Bezier curve segment is

$$B(u,v) = \sum_{i=0}^{n} \binom{n}{i} v^i (1-v)^{n-i} \boldsymbol{P_i(u)}$$

Our discussion suggests that we have a grid of control points. This grid of control points is called a control net. Each control point is denoted by Pij. Since $\boldsymbol{P_i(u)}$ is a Bezier curve, it is defined as

$$\boldsymbol{P_i(u)} = \sum_{j=0}^{n} \binom{n}{j} u^j (1-u)^{n-j} \boldsymbol{P_{ij}}$$

If we substitute $\sum_{j=0}^{n} \binom{n}{j} u^j (1-u)^{n-j} \boldsymbol{P_{ij}}$ for $\boldsymbol{P_i(u)}$ in the equation of a moving Bezier curve segment, we get

$\sum_{i=0}^{n}\binom{n}{i} v(1-v)^{n-i} \sum_{j=0}^{n} \binom{n}{j} u^j (1-u)^{n-j} \boldsymbol{P_{ij}} = \sum_{i=0}^{n}\sum_{j=0}^{n}\binom{n}{i}\binom{n}{j} u^j (1-u)^{n-j} v(1-v)^{n-i}\boldsymbol{P_{ij}} =$
$\sum_{i=0}^{n}\sum_{j=0}^{n} B_j^n(u) B_i^n(v) \boldsymbol{P_{ij}}$,

which is the equation of a Bezier surface patch. It is easy to show that a Bezier surface patch interpolates the four corner points of its control net. An example of a Bezier surface patch is shown in figure 6.

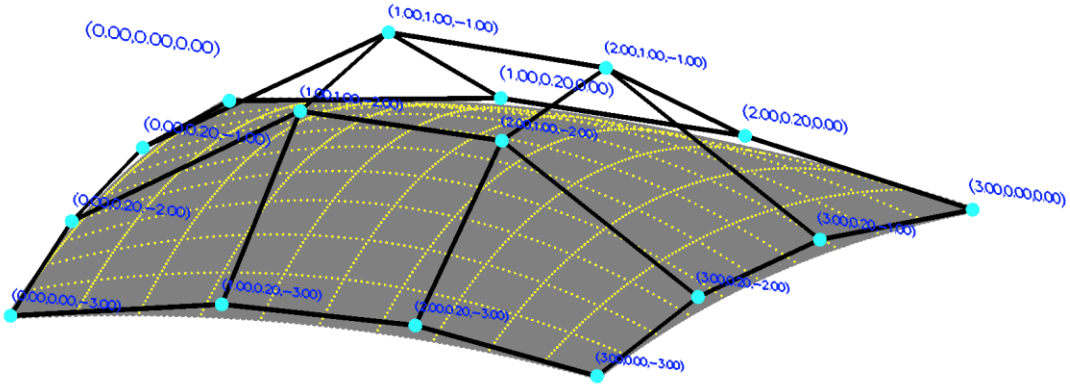


Figure 6: A Bezier surface patch with Beta = 0.

3 BETA-BEZIER SURFACE

A Beta-Bezier surface patch is defined as the locus of a moving, deforming Beta-Bezier curve segment. Proceeding in the same manner as above, we get

$$B(u,v;\beta_u;\beta_v) = \sum_{i=0}^{n}\sum_{j=0}^{n} B_j^n(u;\beta_u)B_i^n(v;\beta_v)\boldsymbol{P_{ij}}.$$

A Beta-Bezier surface patch interpolates its four corner control points. The following is the proof:

$$B_0^n(0;\beta) = \binom{n}{0}\frac{\prod_{i=0}^{0-1}(0+i\beta)\prod_{j=0}^{n-1-0}((1-0)+\mathrm{j}\beta)}{\prod_{m=0}^{n-1}(1+m\beta)}$$

By the definition of a binomial coefficient, $\binom{n}{0} = 1$. $\prod_{i=0}^{-1} i\beta$ is an empty product and is, by definition, equal to 1. The remaining two products are equal and, therefore, cancel out. So, $B_0^n(0;\beta) = 1$. For 0 < k < n, $\prod_{i=0}^{k-1} i\beta$ is 0 and, therefore, $B_k^n(0;\beta)$ is 0. Thus, $B(0,0;\beta) = P_{00}$.

$$B_n^n(1;\beta) = \binom{n}{n}\frac{\prod_{i=0}^{n-1}(1+i\beta)\prod_{j=0}^{n-1-n}((1-1)+\mathrm{j}\beta)}{\prod_{m=0}^{n-1}(1+m\beta)}$$

By the definition of a binomial coefficient, $\binom{n}{n} = 1$. $\prod_{j=0}^{-1}(\mathrm{j}\beta)$ is an empty product, and the other two products cancel out, which means $B_n^n(1;\beta) = 1$. For 0 < k < n, $\prod_{i=0}^{n-1-k} j\beta$ is 0 and, therefore, $B_k^n(1;\beta)$ is 0. This implies the following:

$$B(1,1;\beta) = P_{nn}$$
$$B(0,1;\beta) = P_{0n}$$
$$B(1,0;\beta) = P_{n0}$$

A Beta-Bezier surface patch lies within the convex hull of its control points. To prove this, it suffices to show that $\sum_{i=0}^{n}\sum_{j=0}^{n} B_j^n(u;\beta_u)B_i^n(v;\beta_v) = 1$ and that $B_k^n(x;\beta) > 0$ for $x \in [0,1]$. We start with the latter. The beta function (3) is positive for $\forall\alpha,\gamma > 0$. The binomial coefficient is positive. Recall that

$$B_k^n(x;\lambda) = \binom{n}{k}\frac{b(\lambda t+k,\lambda(1-t)+n-k)}{b(\lambda t,\lambda(1-t))}, \text{ where } \lambda = \frac{1}{\beta}.$$

Since this formula only consists of divisions and multiplications, $B_k^n(x;\lambda) > 0$ for $x \in [0,1]$ and, therefore, $B_k^n(x;\beta) > 0$ for $x \in [0,1]$[1]. It remains to show that

$$\sum_{i=0}^{n}\sum_{j=0}^{n} B_j^n(u;\beta_u)B_i^n(v;\beta_v) = 1.$$

[1] Since α and γ in the beta function are required to be greater than 0 and since $\alpha = \lambda t + k$ and $\gamma = \lambda(1-t)+n-k$, λ is required to be greater than or equal to 0. It follows that, in order for the convex hull property to hold, β must be nonnegative.

We first show that $\sum_{k=0}^{n} B_k^n(t;\beta) = 1$; that is, we show that $\sum_{k=0}^{n}\binom{n}{k}\frac{b(\lambda t+k,\lambda(1-t)+n-k)}{b(\lambda t,\lambda(1-t))} = 1$. We note that $b(\lambda t, \lambda(1-t))$ is independent of the summation index and, hence, can be factored out. In other words, we only need to show that $\sum_{k=0}^{n}\binom{n}{k} b(\lambda t+k, \lambda(1-t)+n-k) = b(\lambda t, \lambda(1-t))$. $b(\lambda t+k, \lambda(1-t)+n-k) = \int_0^1 x^{\lambda t+k-1}(1-x)^{\lambda(1-t)+n-k-1}dx$. So, $\sum_{k=0}^{n}\binom{n}{k} b(\lambda t+k, \lambda(1-t)+n-k) = \sum_{k=0}^{n}\binom{n}{k}\int_0^1 x^{\lambda t+k-1}(1-x)^{\lambda(1-t)+n-k-1}dx$. If we factor out $x^k(1-x)^{n-k}$, we get $\sum_{k=0}^{n}\binom{n}{k}\int_0^1 (x^k(1-x)^{n-k})x^{\lambda t-1}(1-x)^{\lambda(1-t)+1}dx$. After slight rearrangement, we get $\int_0^1(\sum_{k=0}^{n}\binom{n}{k}x^k(1-x)^{n-k})x^{\lambda t-1}(1-x)^{\lambda(1-t)+1}dx$. By the binomial theorem, $\sum_{k=0}^{n}\binom{n}{k}x^k(1-x)^{n-k} = (x+(1-x))^n = 1$. This simplifies the expression to $\int_0^1 x^{\lambda t-1}(1-x)^{\lambda(1-t)+1}dx$. Recall that this is the beta function with $\alpha = \lambda t$ and $\beta = \lambda(1-t)$; that is, $\int_0^1 x^{\lambda t-1}(1-x)^{\lambda(1-t)+1}dx = b(\lambda t, \lambda(1-t))$, which is what we were to prove. $\sum_{i=0}^{n}\sum_{j=0}^{n} B_j^n(u;\beta_u)B_i^n(v;\beta_v)$ can be rewritten as $\sum_{i=0}^{n} B_i^n(v;\beta_v)\sum_{j=0}^{n} B_j^n(u;\beta_u)$. The inner summation evaluates to 1. So,

$$\sum_{i=0}^{n} B_i^n(v;\beta_v)\sum_{j=0}^{n} B_j^n(u;\beta_u) = \sum_{i=0}^{n} B_i^n(v;\beta_v) \tag{6}$$

This also evaluates to 1, which completes our proof.

A bicubic Beta-Bezier surface patch can be represented by a bicubic Bezier surface patch. Recall that a Beta-Bezier surface patch is defined as the locus of a moving, deforming Beta-Bezier curve segment. So, it can be written as

$$B(u,v;\beta_u;\beta_v) = \sum_{i=0}^{n} B_i^n(v;\beta_v)\,\boldsymbol{P_i}(\boldsymbol{u};\boldsymbol{\beta_u}) \tag{7}.$$

For each $\boldsymbol{P_i}(\boldsymbol{u};\boldsymbol{\beta_u})$, we find $\bar{P}_{0,j}$, $\bar{P}_{1,j}$, $\bar{P}_{2,j}$ and $\bar{P}_{3,j}$, so that $\boldsymbol{P_i}(\boldsymbol{u};\boldsymbol{\beta_u})$ can be expressed as a cubic Bezier curve in u as follows:

$$\boldsymbol{P_i}(\boldsymbol{u};\boldsymbol{\beta_u}) = \sum_{i=0}^{3} \bar{P}_{i,j}B_i^3(u) \tag{8}.$$

$\bar{P}_{0,j}$, $\bar{P}_{1,j}$, $\bar{P}_{2,j}$ and $\bar{P}_{3,j}$ can be found as follows:

$$\bar{P}_{0,j} = P_{0,j}$$

$$\bar{P}_{1,j} = \frac{\beta(3+4\beta)}{3(1+\beta)(1+2\beta)}P_{0,j} + \frac{1}{1+2\beta}P_{1,j} + \frac{\beta}{(1+\beta)(1+2\beta)}P_{2,j} + \frac{2\beta^2}{3(1+\beta)(1+2\beta)}P_{3,j}$$

$$\bar{P}_{2,j} = \frac{2\beta^2}{3(1+\beta)(1+2\beta)}P_{0,j} + \frac{\beta}{(1+\beta)(1+2\beta)}P_{1,j} + \frac{1}{1+2\beta}P_{2,j} + \frac{\beta(3+4\beta)}{3(1+\beta)(1+2\beta)}P_{3,j}$$

$$\bar{P}_{3,j} = P_{3,j}$$

By substituting (7) into (8), we get

$$\sum_{j=0}^{3}[\sum_{i=0}^{3}\bar{P}_{i,j}B_i^3(u)]B_j^3(v;\beta_v) = \sum_{i=0}^{3}[\sum_{j=0}^{3}\bar{P}_{i,j}B_j^3(v;\beta_v)]B_i^3(u) = \sum_{i=0}^{3}\bar{P}_i(v;\beta_v)B_i^3(u) \tag{9}$$

For each $\bar{P}_i(v;\beta)$, we find $Q_{i,0}$, $Q_{i,1}$, $Q_{i,2}$ and $Q_{i,3}$ using the same procedure as above, so that $\bar{P}_i(v;\beta)$ can be expressed as a cubic Bezier curve in v as follows:

$$\sum_{j=0}^{3} Q_{i,j}B_{3,j}(v) \tag{10}$$

By substituting (9) into (10), we have a bicubic Bezier surface patch representation for $B(u,v;\beta_u;\beta_v)$. Figures 7 and 8 show two examples of Beta-Bezier surface patches.

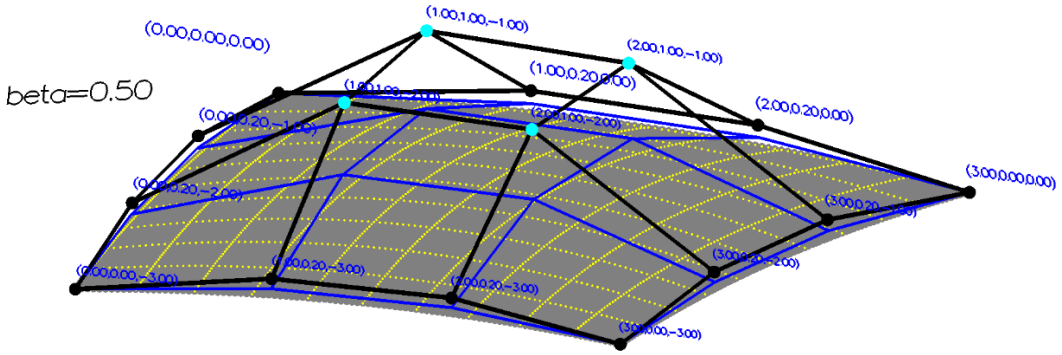


Figure 7: The blue control polygon is the control polygon for the Bezier surface patch.

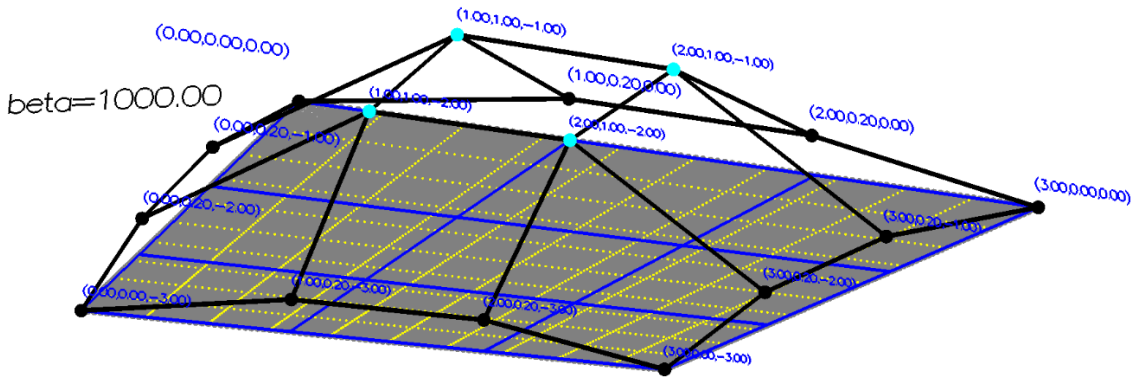


Figure 8: The blue control polygon is the control polygon for the Bezier surface patch.

NURBS surfaces can also be reshaped without moving their control points; each vertex on the surface has a weight which "pulls" the surface towards it. However, this makes it hard to change the overall shape of a surface. Beta-Bezier surfaces provide an easy way to change the overall shape of a surface.

4 COMPOSITE BETA-BEZIER SURFACES

A Beta-Bezier surface patch interpolates its four corner control points. This means that complex shapes can be modeled using a composite Bezier surface (i.e. multiple patches pieced together). For any two patches to meet smoothly (e.g. to be C^n continuous), there are two conditions that must be met:

- The two patches must have a common boundary
- All the columns of their control nets (or rows if the two patches are to be pieced together sideways) must be control polygons for C^n continuous curves

Let's suppose that there is a 3D regular quad mesh that we wish to interpolate. Our algorithm works as follows. It (a) generates topographical curves interpolating data points at the same longitude, (b) generates topographical curves interpolating data points at the same latitude (including the points generated by step (a)), and finally (c) generates a piecewise surface using the control points computed in the process. There are many representations for 3D meshes, none of which directly tells us the topography of the mesh. However, it can be deduced through a simple algorithm. For the sake of argument, let's restrict our attention to mesh files which tell us the vertex locations and the face definitions. This is a common representation. In a regular quad mesh, every vertex has four neighbors, two of which are on the same latitude/longitude (depending on which phase of the interpolation algorithm we are in). Recall that the cross product of two vectors gives us a vector that's orthogonal to the two vectors. The direction of this vector is simply determined using the right-hand rule. More specifically, a positive z-component of the product means the two vectors make a right turn, a negative z means a left turn and a zero z means the vectors are pointing in the same direction. We can apply this to the edges incident to the mesh vertex to choose the neighbor that is at the same latitude/longitude keeping in mind that that edge leading to that neighbor will probably not be in the exact same direction as the other edge. This suggests that we need to traverse the mesh. The depth-first search algorithm is well-suited for this purpose. So, in summary, we compute a topographical curve on the mesh by performing a depth-first search on the mesh choosing the edge with the median z-component of the cross product at every step. This is inspired by Graham's scan [11]. As is the case with curves, each patch can have its own beta or there can be a global β. Composite Beta-Bezier surfaces are shown in figures 9, 10, 11, and 12. The black dots in those figures are the controls points.

To prove the correctness of our algorithm, we need to prove that it produces the right number of control points for each patch and that the control points meet the continuity conditions. We start with the former. We will prove it for the bicubic case. To generate a bicubic Beta-Bezier patch, we need 16 control points. If we interpolate a column of the data mesh, we get two control points for every segment. After interpolating all the columns, each patch has 4 control points plus the 4 data points, for a total of 8 control points. Put another way, each patch has four rows of control points each containing 2 control points.

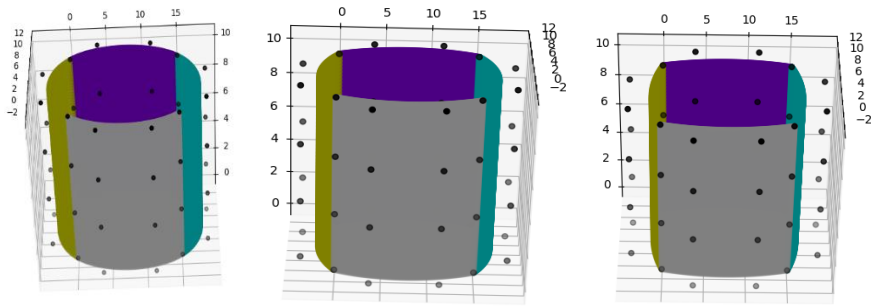


Figure 9: Composite surface with Beta = 0, Beta = 0.5, and Beta = 1.5.

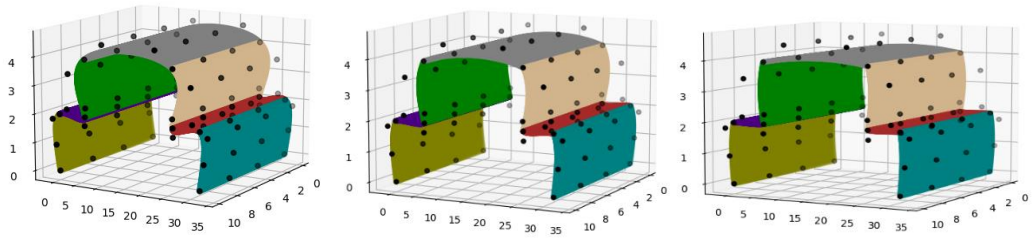


Figure 10: Composite surface with Beta = 0, Beta = 0.5, and Beta = 1.5.

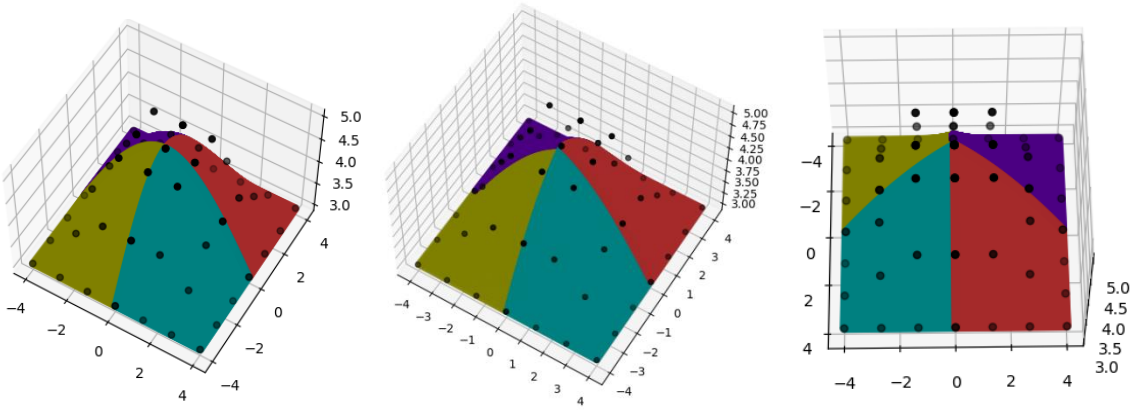


Figure 11: Composite surface with Beta = 0, Beta = 0.5, and Beta = 1.5.

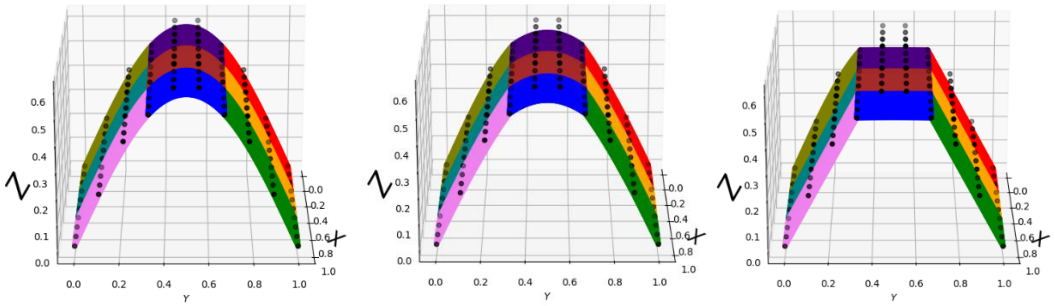


Figure 12: Composite surface with Beta = 0, Beta = 0.5, and Beta = 5.

If we interpolate those rows, we get 8 additional control points, totaling 16 control points. Curve interpolation, as described in section II, generates cubic curves that are C2 continuous. Therefore, the columns (and rows) of the control nets of the adjacent patches produced by our algorithm are control polygons of C2 continuous Beta-Bezier curves (i.e. the second continuity condition). We start by interpolating the data points. This guarantees that we have common boundary curves (i.e. the first continuity condition), which completes our proof.

As for the running time of our algorithm, let's suppose there are n columns and m rows on the mesh. As explained in section II, interpolating m points involves solving an $2m \times 2m$ system which, in a worst-case scenario, takes O(m^3) time. Therefore, step (a) of our algorithm takes O(nm^3) time. Following similar reasoning, we can conclude that step (b) takes O(mn^3) time, for a total running time of O($nm^3 + mn^3$).

A more complex surface (an elliptic torus) is shown in figure 13. The isophotes for that surface are shown in figure 14.

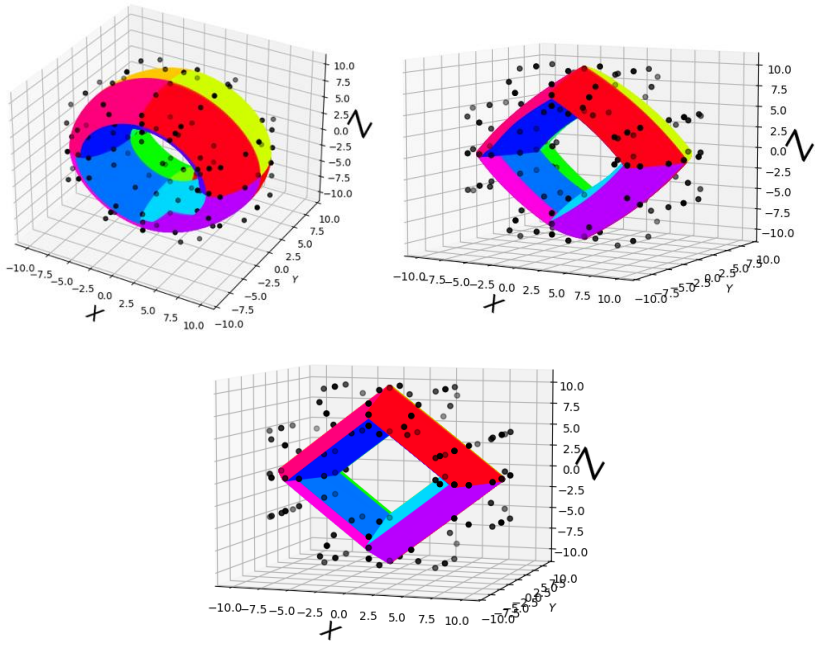
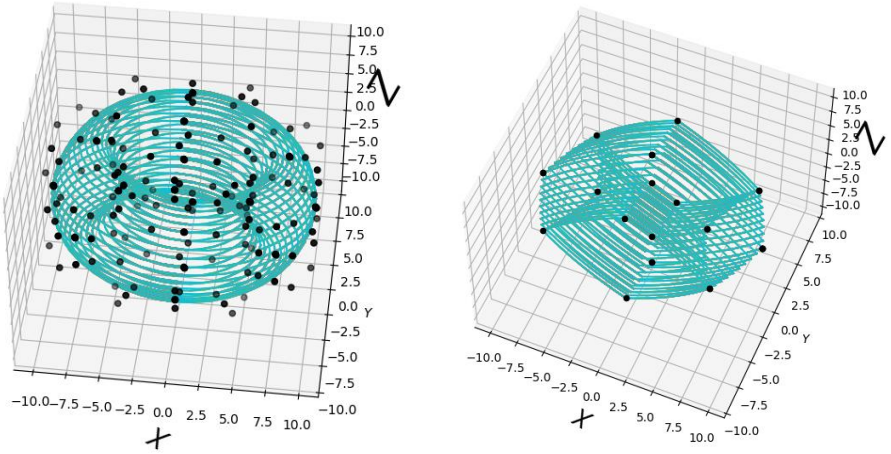


Figure 13: Composite surface with Beta = 0, Beta = 3, and Beta = 100.



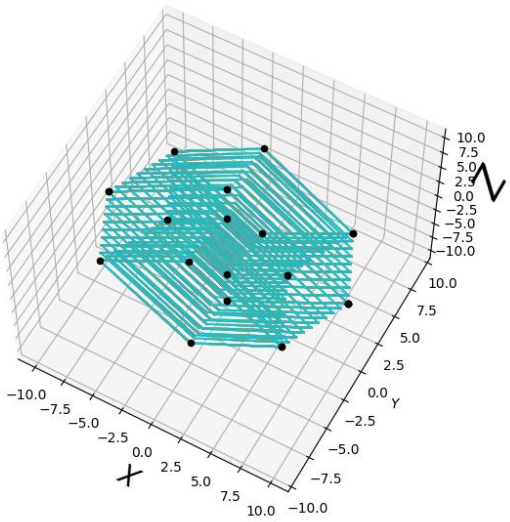


Figure 14: Isophote plots for the surfaces in figure 13.

5 CONCLUSIONS

In this paper, we extend the concept of tension control proposed in [1] from curves to surfaces. The proposed type of surface patches can be reshaped without moving its control points. In addition to extending the work of [1] to surfaces, we propose an efficient rectangular mesh interpolation scheme that makes use of the proposed type of patches.One thing that should be studied is how a Beta-Bezier surface can be represented as a B-spline surface. Also, our interpolation scheme only works with rectangular grids. Meshes with arbitrary topology should also be considered. More work is needed to address these tasks, including providing supporting examples on morphing applications of this new surface tension control technique.

REFERENCES

[1] Cheng, F.; Kazadi, A.; Lin, A.: Beta-Bezier curves, Computer Aided Design and Applications 18(6), 2021, 1265-1278. https://doi.org/10.14733/cadaps.2021.1265-1278

[2] Cao, J.; Wang, G.Z.: An extension of Bernstein-Bezier surface over the triangular domain, Progress Nat. Sci. 17, 2007, 352-357. https://doi.org/10.1080/10020070612331343269.

[3] Chu, L.; Zeng, X.M.: 2014. Constructing curves and triangular patches by Beta functions. Journal of Computational and Applied Mathematics 260, 191-200. https://doi.org/10.1016/j.cam.2013.09.025.

[4] Farin, G.E.: Curves and surfaces for computer aided geometric design: A practical guide. Academic Press. 1998, https://doi.org/10.1016/B978-0-12-460515-2.50020-2

[5] Lin, H.; Chen, W.; Bao, H.: Adaptive patch-based mesh fitting for reverse engineering, Computer-Aided Design, 39(12), 2007, 1134–1142. https://doi.org/10.1016/j.cad.2007.10.002.

[6] Eck, M.; Hoppe, H.: Automatic reconstruction of B-spline surfaces of arbitrary topological type. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH '96). Assoc. for Computing Machinery, New York, NY, USA, 1996, 325–334. https://doi.org/10.1145/237170.237271

[7] Shirman, L.; Sequin, C.: Local surface interpolation with Bezier patches, Computer Aided Geometric Design, 4, 1987, 279-95. https://doi.org/10.1016/0167-8396(87)90003-3.

[8] Yan, L.L.; Liang, J.F.: An extension of the Bezier model, Applied Mathematics and Computation, 218, 2011, 2863-2879. https://doi.org/10.1016/j.amc.2011.08.030

[9] Yang, L.Q.; Zeng, X.M.: Bezier curves and surfaces with shape parameters, Int. J. Comput. Math. 86, 2009, 1253-1263. https://doi.org/10.1080/00207160701821715.

[10] Zhu, Y.; Han, X.: Quasi-Bernstein-Bezier polynomials over triangular domain with multiple shape parameters, Applied Mathematics and Computation, 250, 2015, 181-192. https://doi.org/10.1016/j.amc.2014.10.098

[11] Graham, R.: An efficient algorithm for determining the convex hull of a finite planar set. Information Processing Letters, 1(4), 1972, 132–133. https://doi.org/10.1016/0020-0190(72)90045-2