CAD: Curves and Surfaces: Part II of III

Historically, the solution to the problem of complex shapes has been solved by using ***cubic (or higher order) polynomials***. Polynomials have several advantages. They allow easy computation of points, tangents etc. They require simple data structures to store in a computer, and computations using polynomials are robust (not easily susceptible to floating point errors). The simplest vector cubic polynomial function is of the form:

$\mathbf{r}(u) = \mathbf{a} + \mathbf{b}u + \mathbf{c}u^2 + \mathbf{d}u^3 = [1\ u\ u^2\ u^3]\,[\,\mathbf{a}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\,]^T = \mathbf{U}\,\mathbf{A}; \qquad 0 \le u \le 1.$

This equation (called a ***power basis representation***) is good for computations – you can easily find coordinates of a point for any value of u; it is easy to differentiate and get the tangent vector, or the curvature. From a design point of view: it generates a smooth curve passing through four specified points. Substitute each point for $\mathbf{r}(u)$ to obtain four vector equations in four unknowns **a**, **b**, **c**, and **d**. The solution can be obtained by Gaussian elimination. Physically, however, the values of the parameters **a**, **b**, **c**, and **d** do not have an intuitive meaning: if you knew their values, you could not 'guess' the shape of the curve.

Ferguson's cubic curves

We need to calculate four vector values (the **A** matrix, also called the ***coefficient matrix***), in order to specify the curve. Ferguson imposed the following constraints to determine the coefficient matrix:
The two end points, P_0 and P_1, of the curve are assigned;
The tangent directions at the two end points, $\mathbf{t}_0$ and $\mathbf{t}_1$, are defined.

This leads to: $P_0 = \mathbf{r}(0)$; $P_1 = \mathbf{r}(1)$; $\mathbf{t}_0 = d\mathbf{r}/du\,|_{u=0}$ and $\mathbf{t}_1 = d\mathbf{r}/du\,|_{u=1}$.
$d\mathbf{r}/du = \mathbf{b} + 2u\mathbf{c} + 3u^2\mathbf{d}$.

Solving this system, we get:
$\mathbf{a} = P_0$
$\mathbf{b} = \mathbf{t}_0$
$\mathbf{c} = -3P_0 + 3P_1 - 2\mathbf{t}_0 - \mathbf{t}_1$
$\mathbf{d} = 2P_0 - 2P_1 + \mathbf{t}_0 + \mathbf{t}_1$

Which is expressed as:

$$r(u) = UA = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & 1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ t_0 \\ t_1 \end{bmatrix} = UCS$$

It is easy to see that ***Ferguson's curve form*** is equivalent to the power basis form (hence it si equally easy to compute the coordinates, tangents etc), but provides a different way to *design* the curve.

Bézier curves

This is an elegant polynomial representation with several nice properties. An n-th degree Bezier curve is defined using *n* points, P_i, as:

$r(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i \qquad 0 \le u\, 1$, where the ***basis*** (also called ***blending***) ***functions*** are the Bernstein polynomials, given by $B_{i,n}(u) = \frac{n!}{i!\,(n-i)!} u^i \, (1-u)^{n-i}$. The points P_i are called control points.

Why is this form any better than power basis, or Ferguson's curves ? To answer this, we look at several 'nice' properties of Bezier curves.

BP1. Since Bezier curves are polynomial functions, they are easily computed, and infinitely differentiable (so tangents and curvatures are easily computed).

BP2. The curve begins at the first control point, P_0, and ends at the last control point, P_n. It *does not* pass through any of the intermediate control points.

BP3. The tangent at the start point (u = 0) lies along the vector from P_0 to P_1; the tangent at the end point (u = 1) lies along the vector from P_{n-1} to P_n.

BP4. The entire curve lies in the interior of the convex hull of the control points. This is called the convex hull property, and is extremely useful for many CAD/CAM routines.

BP5. Bezier curves are invariant over affine transformations of the control points. In other words, if the control points are translated, or rotated, the curve moves to the corresponding new coordinate frame without changing its shape.

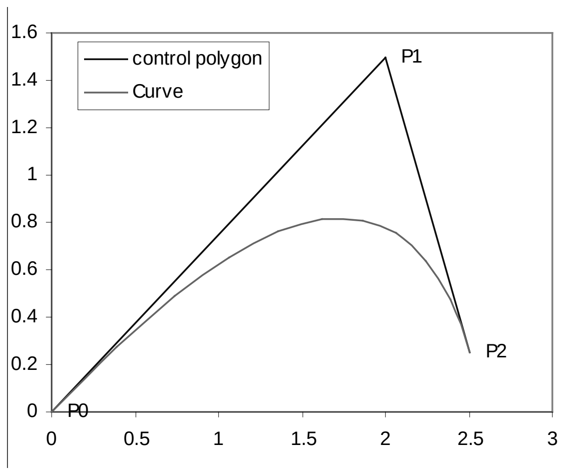
We look at some examples of Bezier curves.

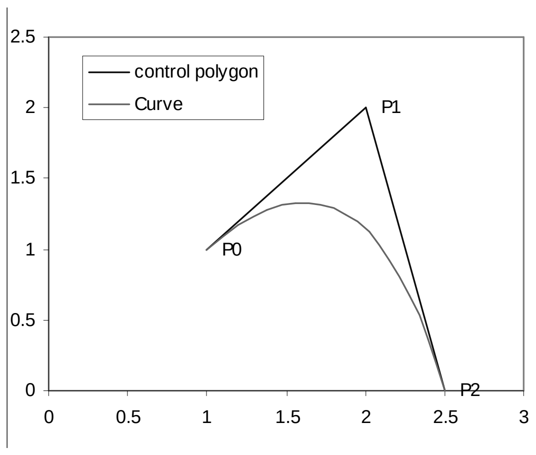
Bezier Example 1. n = 1. $B_{0,1}(u)$ = 1-u, and $B_{1,1}(u)$ = u.

$C(u) = (1-u)\boldsymbol{P_0} + u\boldsymbol{P_1}$, which is the equation of a straight line between $\mathbf{P_0}$ and $\mathbf{P_1}$.

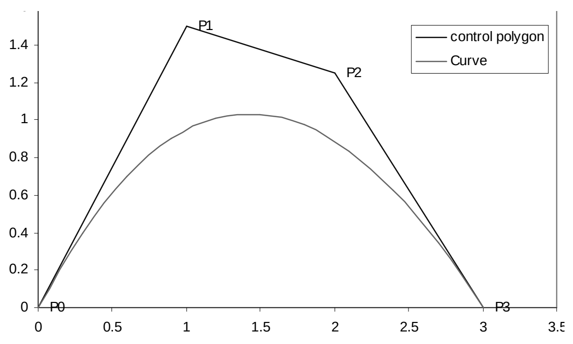
Bezier Example 2. n = 2. Now we get $C(u) = (1-u)^2\, \boldsymbol{P_0} + 2u(1-u)\, \boldsymbol{P_1} + u^2\, \boldsymbol{P_2}$, which is a parabolic arc from $\mathbf{P_0}$ to $\mathbf{P_2}$.

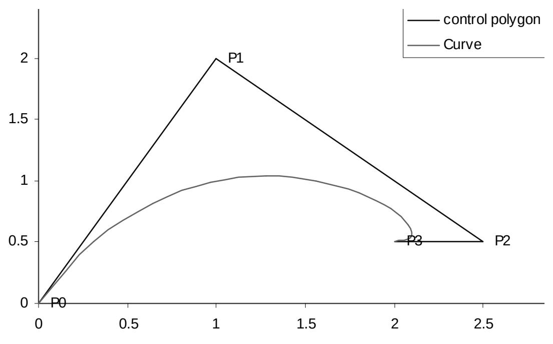
While the control points can be in 3D space, the curve lies entirely in the plane defined by the three control points. Also, the curve is approximated quite nicely by the polygon formed by the control points (the control polygon) – as seen in the examples of the figure below.

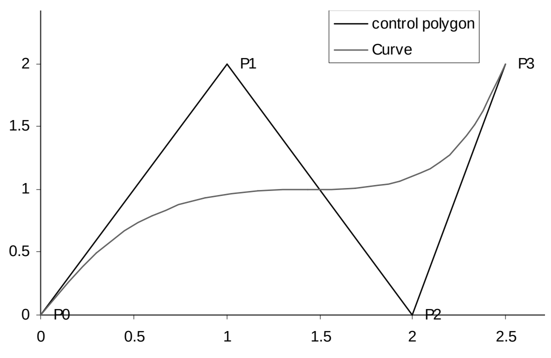


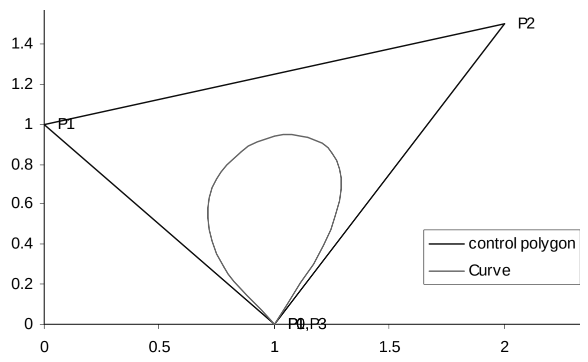


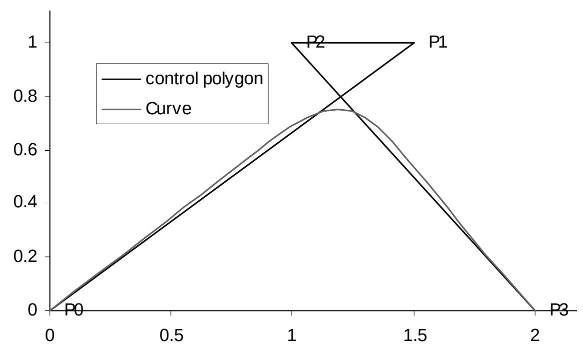
Bezier Example 3. n =3. This is a very commonly used from – it can represent a fairly complex set of shapes of curves, as seen in the figures below. Notice how the shape of the control polygon approximates the shape of the curve. The convex hull property is true (as it is for all Bezier curves). A loop in the control polygon may or may not lead to a loop in the Bezier curve – as seen in the example below. By proper selection of control points, you can get a cusp (point where the derivative is not defined.)

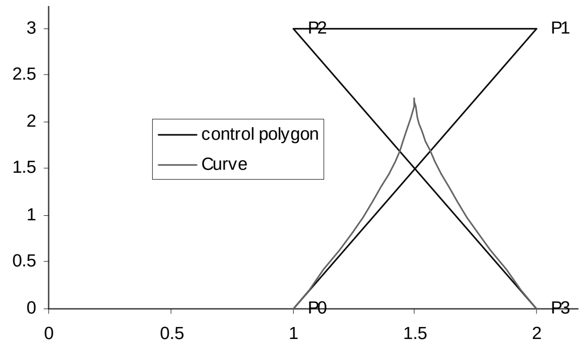


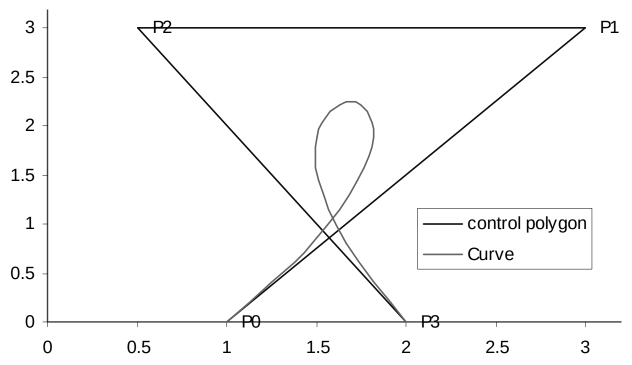












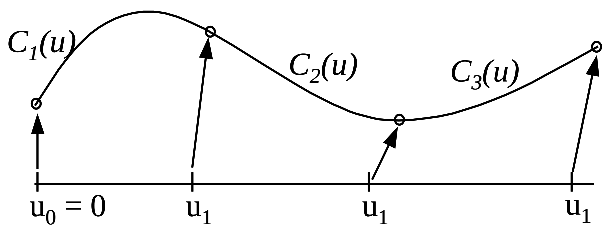
Intuitively, Bezier curves follow the shape of the control polygon. The reason is .seen from an analysis of the basis functions. At (u =0), the curve is influenced purely by the first control point, $\mathbf{P_0}$. At (u = 1), it is purely influenced by $\mathbf{P_n}$. In the middle, each control point, $\mathbf{P_i}$, ‘influences’ or ‘attracts the curve most when the value of the parameter u is closest to *i/n*. To convince yourself about this, draw the functions (a) $B_{i,2}$ for i= 0, 1, 2 on a single graph; (b) $B_{i,3}$, for i= 0, 1, 2, 3 on a single graph.

B-spline curves

Imagine that you need to represent a very complex, curved shape approximated by, say, 20 points. If we design this shape using a Bezier, we will need a degree 19 curve. Such curves are difficult to maintain/modify and compute. We would prefer to work with lower degree functions for computational reasons. Further, depicting a curve by a single high degree function is not good when we modify shapes – since each control point has some effect on the shape of the curve. Thus, changing the location of P_{18} will change the shape (a little) near the control point P_2. But often, in design, we would like more “localized” control on shape modification. We now study a method to solve both of the above problems.

The solution to the problem is the following: divide the entire shape into smaller segments, and represent each segment with a low order polynomial. The maths of this simple idea is not too simple, but we shall look at the basics at least. The figure below shows this idea, where the curve $\boldsymbol{C(u)}$ is made up of 3 segments, defined as $\boldsymbol{C_i(u)}$, i = 1, 2, 3, the domain of $\boldsymbol{C_i}$ is over

$u_{i-1} \leq u \leq u_i$, and $u_0 = 0 < u_1 < u_2 < u_2 < u_3$. Now we can use any of the conventional (e.g. power basis, or Ferguson, Bezier etc.) mechanisms to describe each curve segment ***C**_i*, but to be useful, we must ensure that at the intermediate vertices, called the breakpoints, the curves maintain some level of continuity.



There are many ways to implement the above idea. We shall look at ***B-Splines***, which are a general form of Bezier. B-splines provide functions of the form: $C(u) = \sum_{i=0}^{n} f_i(u)P_i$, where $\boldsymbol{P_i}$ are the control points, and the $\{f_i(u),\ i = 0, .., n\}$ are ***piecewise polynomial functions*** that form a basis for the vector space of all piecewise polynomial functions of some given degree and continuity at a given set of breakpoints $\{u_i,\ i = 1,.., m\}$. Note that continuity (at least with respect to the parameter *u*) is dictated purely by the basis functions $f_i(u)$ – the control points can be modified without affecting the continuity. Further, we would like ***C**(u)* to possess the convex hull property, as well as the coordinate frame invariance property. Another property we wish our $f_i(u)$'s to have is that of local support – each $f_i(u)$ is non-zero in only a (given) small number of intervals, but not over the entire domain $[u_0, u_m]$. since $\boldsymbol{P_i}$ is multiplied by $f_i(u)$, moving $\boldsymbol{P_i}$ will only affect the curve shape in the range where $f_i(u)$ is non-zero.
All these, and other beautiful properties are possessed by B-splines. To understand B-splines, we need to first study the ***B-spline basis functions***. Let $U = \{ u_0, \ldots, u_m\}$ be a non-decreasing sequence of real numbers, that is, $u_0 \leq u_1 \leq u_2 \ldots \leq u_m$. The u_i are called ***knot points***, or simply ***knots***, and *U* is the ***knot vector***. then the *i-th* B-spline basis function of degree-*p* (order *p+1*), denoted $N_{i,p}(u)$, is defined as:

$$N_{i,0}(u) = \begin{cases} 1 & if\ u_i \leq u \leq u_{i+1} \\ 0 & otehrwise \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

Looks strange ? Let's examine it further:

(1) What are the values of $N_{i,0}(u)$?
(2) When p > 0, $N_{i,p}(u)$ is a linear combination of two basis functions of degree *(p-1).*
(3) If you know the knot vector, and the degree, p, then all basis functions can be calculated easily.

(4) It is easy to see that each $N_{i,p}$ is a polynomial function (what is the degree of the polynomial ?)
(5) The *i-th* ***knot span*** is defined as the half-open interval $[u_i, u_{i+1})$

In order to get a feel for B-spline functions, you have to work with a few examples.

Exercise 1:
Let p = 2 and U = { 0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5}. Compute and plot the functions for all the zero-, one- and two-degree basis functions.

Exercise 2:
Let p = 3, and U = { 0, 0, 0, 0, 1, 1, 1, 1}. compute the B-spline basis functions. Do the functions look familiar ?

Exercise 3:
Prove that B-spline basis functions for U = { 0, 0 …p+1 times … 0, 1, 1, … p+1 times … 1} and degree p are the same as the degree-p Bezier basis functions.

In this sense, Bezier curves are just a special case of B-splines.

Properties of the B-spline functions:

1. Local Support Property
$N_{i,p}(u) = 0$ for u outside the interval $[u_i, u_{i+p+1})$. This property can be deduced from the observation above that $N_{i,p}(u)$ is a linear combination of $N_{i,p-1}(u)$ and $N_{i+1,p-1}(u)$.

2. In any given knot span, at most $p+1$ of the basis functions are non-zero. Which basis functions can be non-zero in the knot span $[u_j, u_{j+1})$?

3. Non-negativity
$N_{i,p}(u) \geq 0$ for all i, p and u. This is proved easily using induction on p.

4. Partition of unity
For any knot span, $[u_i, u_{i+1})$, sum of $N_{j,p}(u) = 1$.

5. Continuity
All derivatives of $N_{i,p}(u)$ exist in the interior of any knot span (since it is a polynomial). At a knot, $N_{i,p}(u)$ is $(p\text{-}k)$ times differentiable, where k is the multiplicity of the knot (that is, the number of knots of the same value is the multiplicity of the knot at that value).
Consequences:
(a) Increasing the degree, p, of a B-spline curve increases its continuity.
(b) Increasing the knot multiplicity decreases the continuity.

6. Derivatives
The following formula describes the k-th derivative of $N_{i,p}(u)$ in terms of the k-th derivatives of the basis functions of $N_{i,p-1}(u)$ *and* $N_{i+1,p-1}(u)$. It can be used to compute all values of derivatives for a given value of u.

$$N_{i,p}^{(k)} = \frac{p}{p-k}\left(\frac{u-u_i}{u_{i+p}-u_i}N_{i,p-1}^{(k)} + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}^{(k)}\right), k = 0,\ldots,p-1$$

The above formula reduces the k-th derivatives of a degree-p basis function in terms of the k-th derivatives of degree p-1. Two other computationally useful forms are the following two:

$$N_{i,p}^{(k)}(u) = p\left(\frac{N_{i,p-1}^{(k-1)}}{u_{i+p}-u_i} - \frac{N_{i+1,p-1}^{(k-1)}}{u_{i+p-1}-u_{i+1}}\right)$$

$$N_{i,p}^{(k)}(u) = \frac{p!}{(p-k)!}\sum_{j=0}^{k} a_{k,j}N_{i+j,p-k},$$

$$where$$

$$a_{0,0} = 1,$$

$$a_{k,0} = \frac{a_{k-1,0}}{u_{i+p-k+1}-u_i},$$

$$a_{k,j} = \frac{a_{k-1,j}-a_{k-1,j-1}}{u_{i+p+j-k+1}-u_{i+j}}, when \quad j = 1,\ldots,k-1,$$

$$a_{k,k} = \frac{-a_{k-1,k-1}}{u_{i+p+1}-u_{i+k}}$$

In the last form, when computing the ai,j's, if the denominator = 0, the expression is defined to have value=0. All derivatives k $\geq$ p are zero (the curve is of degree p).

The lowest and the highest knot values are called the *extreme values* of the knots. All other values of knots are called *interior values*. If all interior knots are spaced uniformly between the extreme values, the resulting B-spline is called a ***uniform B-spline***. Otherwise, it is called a ***Non-uniform B-spline (NUBS)***. The example is Exercise 1 above is non-uniform, since there are two knots of value 4.

A degree-p B-spline curve is defined as:

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u)P_i \qquad a \leq u \leq b,$$

$\{P_i\}$ are the n control points,

$\{N_{i,p}(u)\}$ are degree - p B - spline basis functions over the knot vector with m intervals

$$U = \{a, \ldots, a, u_{p+1}, \ldots, u_{m-p-1}, b, \ldots, b\}$$

(1) Extreme values, *a* and *b*, are repeated (*p+1*) times each. There are (m+1) knots.
(2) The knot vector is, in general, non-periodic and non-uniform

As before, we list some 'nice' properties of B-splines, before going through some examples.

BSP1. If $n = p$, and $U = \{0, \ldots, 0, 1, \ldots, 1\}$, then $\mathbf{C}(u)$ is a Bezier curve.

BSP2. For a curve of degree p, with $(n+1)$ control points and $(m+1)$ knots, $m = n + p + 1$.

BSP3. For $a \le u \le b$, $\mathbf{C}(a) = P_0$, and $\mathbf{C}(b) = P_n$.

BSP4. Affine invariance. Affine transformations of the coordinate system do not change the shape of the B-spline curve.

BSP5. Strong Convex Hull Property.

(a) The curve is contained in the convex hull of its control points.

(b) If $u_i \le u < u_{i+1}$, and $p \le i < m\text{-}p\text{-}1$, then $\mathbf{C}(u)$ is contained in the convex hull of the control points $\mathbf{P}_{i-p}, \ldots, \mathbf{P}_i$. This result can be derived from the non-negativity and partition of unity properties of the basis functions.

BSP6. Local modification

Moving $\mathbf{P}_i$ changes $\mathbf{C}(u)$ only in the interval $[u_i, u_{i+p+1})$, since outside this interval, $N_{i,p}(u) = 0$.

[Why ?]

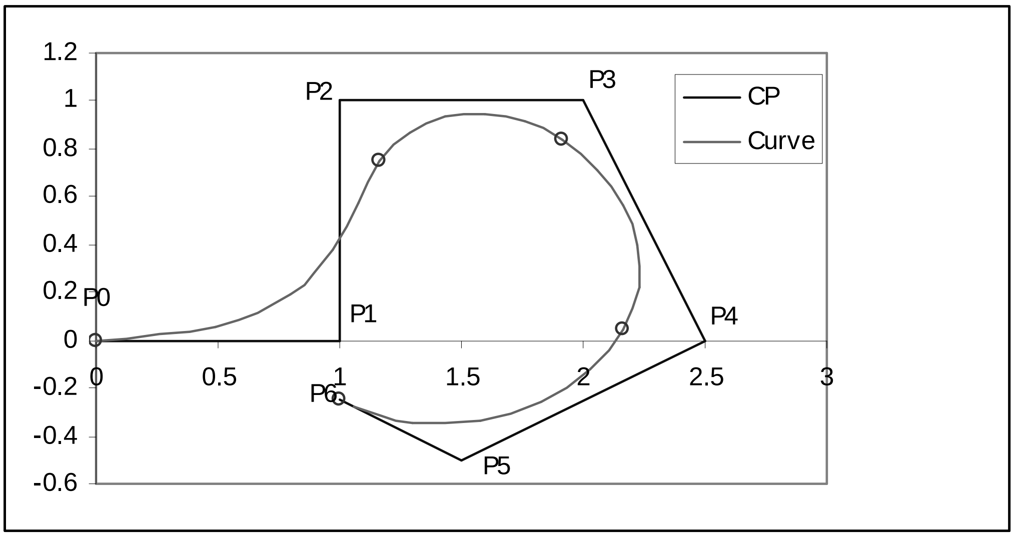
BSP7. A B-spline of degree-1 is identical to the control polygon. As the degree of the B-spline is lowered, it comes closer and closer to the control polygon.

[Why ?]

Now let's look a few examples to understand how B-spline curves behave.

B-spline Example 1.

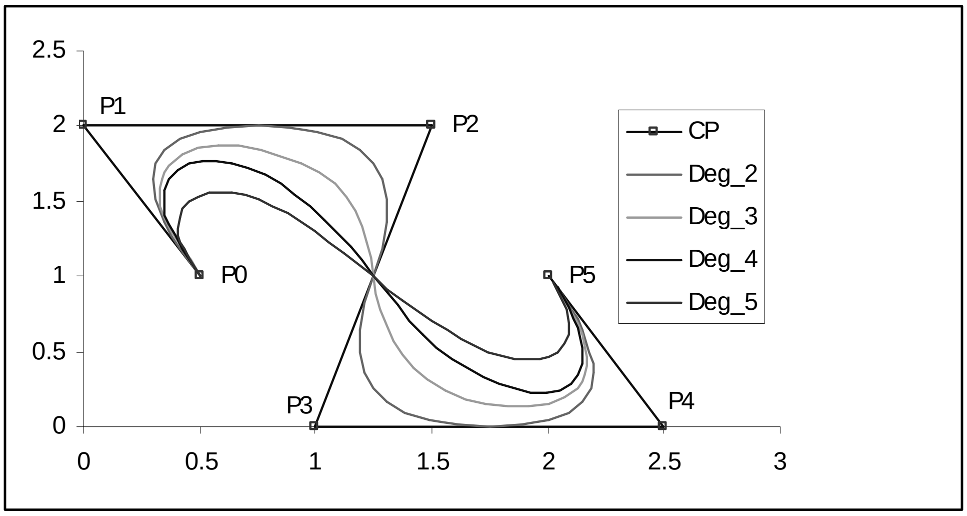
Degree 3, 7 control points, Knot vector U = { 0 0 0 0 0.25 0.5 0.75 1 1 1 1}.



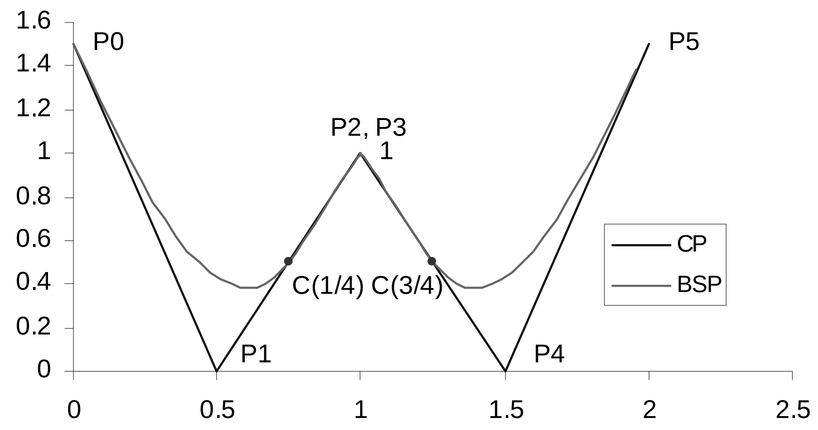
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
0.5
1
1.5
2
2.5
3
P0
P1
P2
P3
P4
P5
P6
CP
Curve

B-spline Example 2.
As degree decreases, the curve gets closer to the Control polygon. Degree=1 coincides with the control polygon.

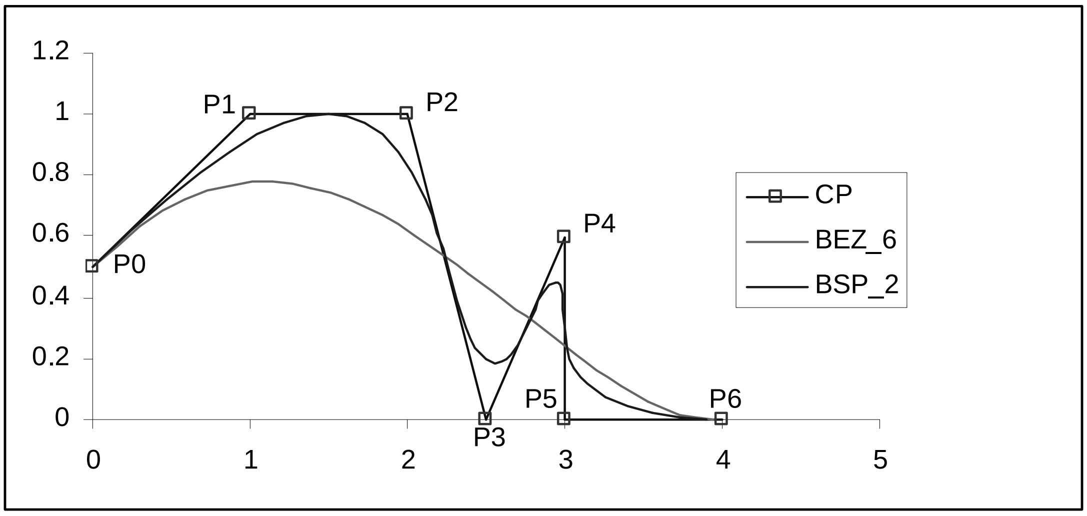
p =1, U ={ 0,0,0.2, 0.4, 0.6, 0.8, 1, 1}
p=2, U = {0, 0, 0, .25, .5, .75, 1, 1, 1}
p=3, U = {0, 0, 0, 0, 0.33, 0.67, 1, 1, 1, 1}
p=4, U = { 0, 0, 0, 0, 0, 0.5, 1, 1, 1, 1, 1}
p=5, U = { 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1} [This is a Bezier]



B-spline Example 3.
By using coincident control points, it is possible to get straight lines.
Degree = 2, U = {0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1}



B-spline Example 4.
This example shows the power of B-splines over Bezier, in terms of how closely we can control the shape of the curve with the control points.
Bezier: Degree 6, U = { 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1}
B-Spline: Degree = 2, U = { 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1}



The above discussion shows the usefulness and power of the B-spline representation. Further, while the mathematical expressions appear to be quite complex, they are in fact quite well behaved, and not too difficult to program. I shall provide a link to a simple perl program that can generate the data for any B-spline that you would like to define. The output data can be copied and pasted into a spreadsheet (I used MSExcel) to plot a chart. That's how I created all the above figures – all in a few hours.

Tensor Product Surfaces

Now that we know how to work with parametric curves, we are ready to move to representations of surfaces. We shall skip power basis and Bezier forms altogether, and jump straight to the definition of B-spline surfaces. Later we shall see that (same as for curves), Bezier surfaces are just a special case of B-splines.
We saw earlier that arbitrary surfaces can be represented implicitly by functions of the form $f(x, y, z) = 0$, or parametrically as $\mathbf{r}(u, v)$. Such representations are impractical to work with, and if we allow arbitrary functions f, then it will often be difficult to derive useful properties such as tangents and normals, area and enclosed volume etc. Instead, we restrict ourselves to well behaved functions of the form $\Sigma_i U(u)V(v)P_i$, where U and V are polynomials with well behaved properties, and P_i's form a (mxn) mesh of points that describe the shape of the surface roughly. Again, by choosing U and V properly, we can get very convenient and powerful representations. For now, we shall let U and V to take the form of B-splines.

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u)\, N_{j,q}(v)\, P_{i,j},\ where$$

$$U = \{0,...,0,u_{p+1},...,u_{r-p-1},1,...,1\} \quad and$$

$$V = \{0,...,0,u_{q+1},...,u_{s-q-1},1,...,1\}.$$

As in B - spline curves, the extreme 0’s and 1’s are repeated p + 1 times in U, and q + 1 times in V

U has *r+1* knots, and *V* has *s+1* knots, and as for curves,
$r = n + p + 1$, and $s = m + q + 1$.

The B-spline surface can be imagined as follows: if we substitute a fixed value of a parameter, say $v=v_0$, into ***S(u,v)***, then we get a function ***R***(u, v_0) of one variable *u*. This is a 3D B-spline curve, which satisfies the equation ***S(u,v)***, and therefore it is a curve on the surface. Such a curve is called an ***iso-parametric*** curve on the surface ***S***. For different values of *u* and *v*, we get different iso-parametric curves, for which we can easily evaluate the points using the methods learnt for B-spline curves.

Specification of a B-spline surface contains the following information:

(a) The number of control points along the *u*-parameter = *n+1*, the degree of the basis functions $N_{i,p}(u) = p$, and the knot vector *U*.
(b) The number of control points along the *v*-parameter = *m+1*, the degree of the basis functions $N_{j,q}(v) = q$, and the knot vector *V*.
(c) The ***control net***, or ***control mesh***, an array of *nm* control points, arranged as follows:

$\mathbf{P_{0,0}, \ldots, P_{0,m}}$
$\mathbf{P_{1,0}, \ldots, P_{1,m}}$
…
$\mathbf{P_{n,0}, \ldots, P_{n,m}}$

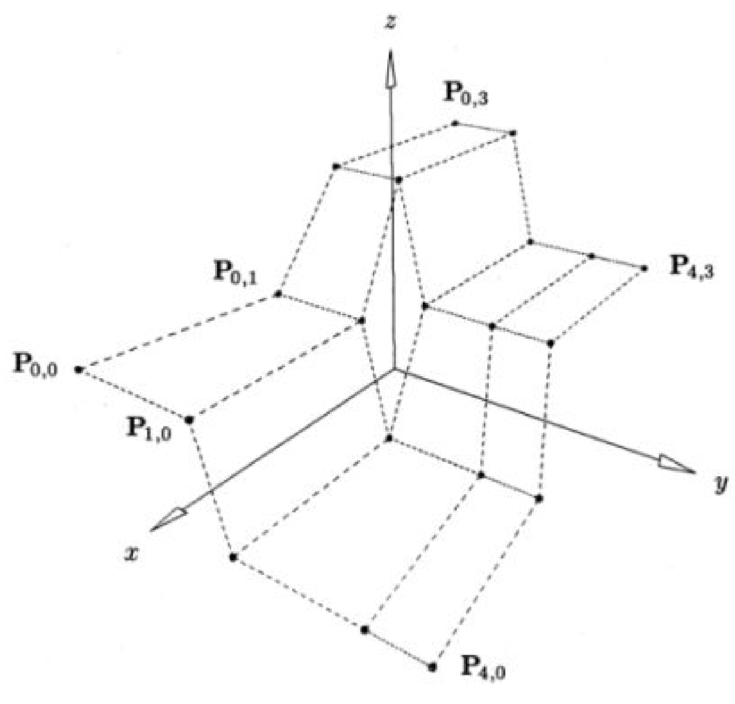
Computing a point on a B-spline

Given a B-spline surface function, and the values of the parameters *(u, v)*, the coordinates of the corresponding surface point are computed as follows:
(a) Find the knot span where u lies;
(b) Find the non-zero basis functions $N_{i-p,p}(u), \ldots, Ni,p(u)$
(c) Find knot span in which v lies;
(d) Find the non-zero basis functions *Nj-q,q(v),…,Nj,q(v)*
(e) Multiply and sum the non-zero basis function values with the corresponding control points.

We now look at a few simple examples, before listing some of the interesting properties of B-splines.

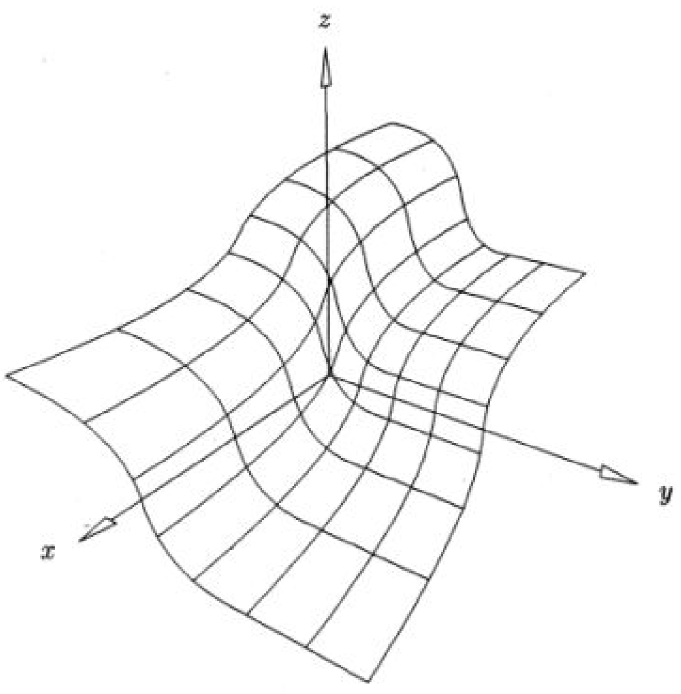
B-Spline Surface Example 1.

This figure shows a control mesh made up of 20 control points, n = 4 and m = 3. The corresponding B-spline surface is plotted in wireframe mode, showing several iso-parametric curves along the surface.



(a)

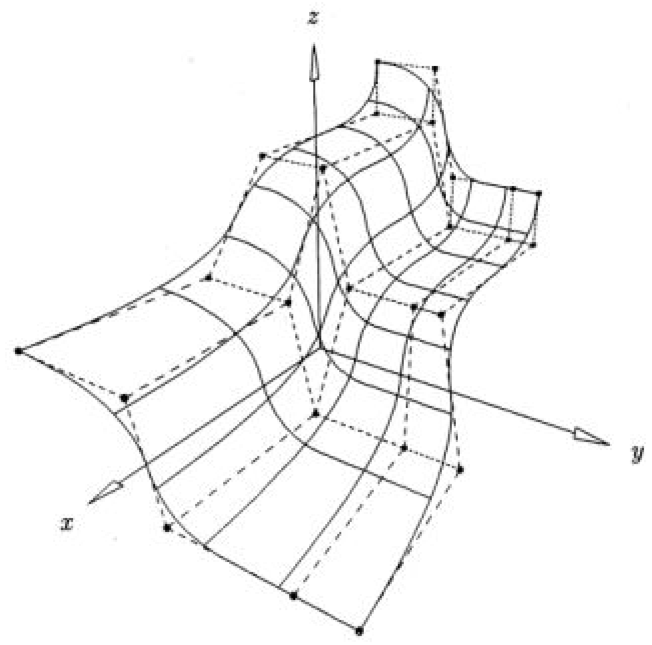
A B-spline surface. (a) The control net; (b) the surface.



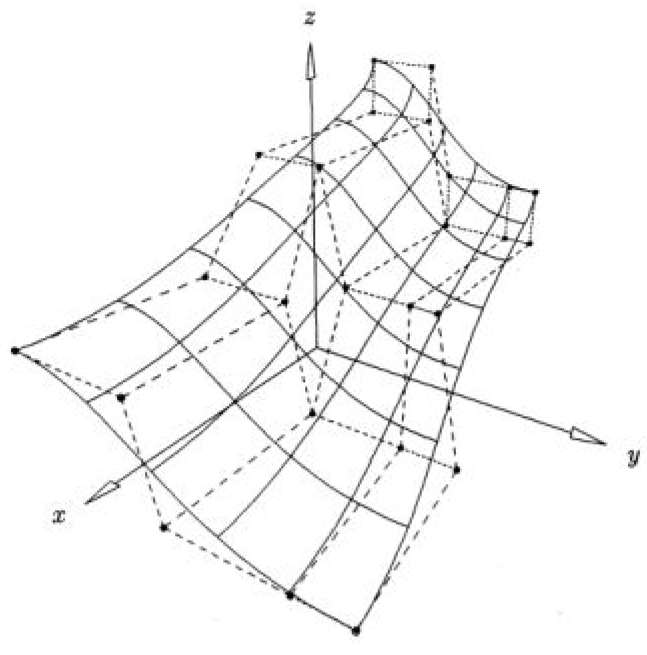
(b)

B-Spline surface Example 2.

This example shows two B-spline surfaces with the same underlying control mesh. The first is a biquadric (that is, p = q = 2), while the second is biquartic (n = p = 4). The U and V vectors are uniform (the only interior knot in each case is at 0.5).
Notice how the lower degree surface is much closer to the control mesh, while the higher degree surface is much “smoother”.



(a)



(b)

Figure 3.22. (a) A biquadratic surface; (b) a biquartic surface ($p = q = 4$) using the same control points as in Figure 3.22a.

B-Spline surface Example 3.

The surface is continuously differentiable if the underlying B-spline curves are. In the interior of a knot span, the surface will be continuously differentiable arbitrary number of times. What about the partial derivatives at a knot ? For a knot of multiplicity *k,* the surface is (partially) differentiable (*p*-*k*) times with respect to *u*, and (*q*-*k*) times with respect to *v*.

The figure below shows a surface that is quadraticXcubic. At the knot u = ½, the curve is differentiable k-p = 2-2 = 0 times. Hence the surface has a 'crease'.

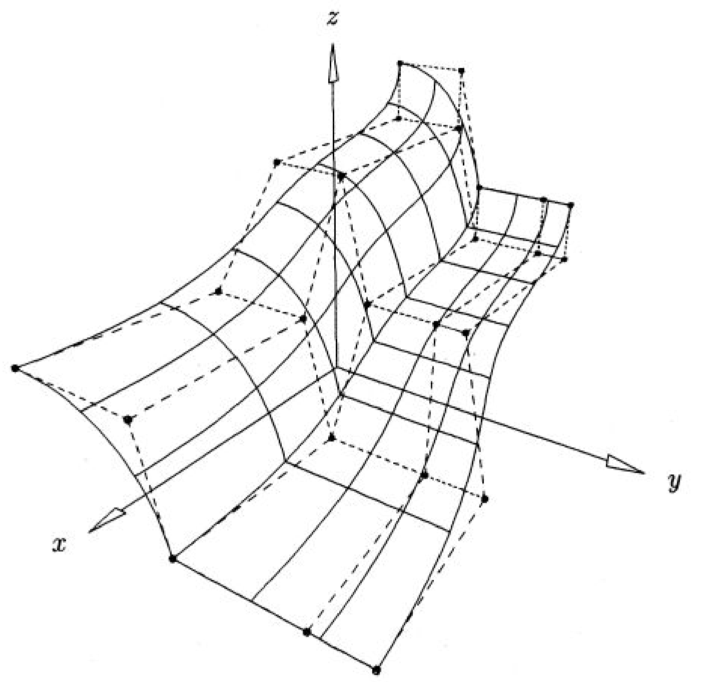


Figure 3.24. A quadratic × cubic surface with crease, $U = \{0,0,0,1/2,1/2,1,1,1\}$ and $V = \{0,0,0,0,1/2,1,1,1,1\}$.

Properties of B-Spline Basis Functions and Surfaces

BSPSP1. The surface interpolates the four corner control points – ***S(0,0)*** = $\boldsymbol{P}_{0,0}$, ***S(0,1)*** = $\boldsymbol{P}_{0,n}$, ***S(1,0)*** = $\boldsymbol{P}_{m,0}$ and ***S(1,1)*** = $\boldsymbol{P}_{n,m}$.

BSPSP2. ***Non-negative, partition of unity basis functions***

The basis functions, Ni,p(u) Nj,q(v) ≥ 0 for u, v; further,

$$\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) = 1, \ \forall (u,v) \in [0,1] \times [0,1]$$

BSPSP3. ***Convex Hull Property***

As a result of property BSPSP2, the entire B-Spline surface is contained inside the convex hull of the its control points.

BSPSP4. ***Affine invariance***

The surface remains unchanged in shape when under affine transformations (rotations and translations).

BSPSP5. ***Local control***
$N_{i,p}(u)\, N_{j,q}(v) = 0$ outside the parameter-space rectangle $[u_i, u_{i+p+1})$ X $[v_j, v_{j+q+1})$.
As a result, if we move $\boldsymbol{P}_{i,j}$, the only portion of the surface that changes shape is the region corresponding to the parameter-space rectangle $[u_i, u_{i+p+1})$ X $[v_j, v_{j+q+1})$.

BSPSP6. ***Special case: Bezier surface***
I we let $n = p$, $m = q$, $U = \{ 0,\ldots, 0, 1,\ldots, 1\}$, and $V = \{ 0, \ldots, 0, 1, \ldots, 1\}$, then $N_{i,p}(u)$ and $N_{j,q}(v)$ are the same as Bernstein polynomials, $B_{i,n}(u)$ and $B_{j,m}(v)$, which are used to define Bezier curves. The surface thus obtained is a special case of B-Spline surfaces, and is called a Bezier surface.

Source: Most of the material for this part is based on *The NURBS book* by Piegl and Tiller – if you plan to work with surfaces, this is simply the best book, and it will be important to have your own copy. also, the figures of the surfaces are copies of the figures from this text.