NURBS

Overview

NURBS, and B-Splines in general, have become powerful computer graphics primitives. Here we will look at some of the basics of B-Spline curves as well as how to create objects from NURBS surfaces using techniques such as lofting.

1 B-Spline Basics

B-Splines are a common way to represent curves in computer graphics. There are many reasons for this, as you will soon see. For instance, a cubic B-spline made up of several segments composited together will possess C^2 continuity between these segments. Unlike Bezier segments, we do not need to impose any continuity requirements to enforce this continuity. B-splines also possess many other useful properties which will be explained in the following section.

1.1 Uniform B-Splines

Let us call the *ith* segment of a cubic B-spline, Q_i. This segment is defined by four control vertices, $p_i, p_{i+1}, p_{i+2}, p_{i+3}$. Therefore, we can see that the following segment, Q_{i+1}, will have control vertices of, $p_{i+1}, p_{i+2}, p_{i+3}, p_{i+4}$. From this, we can see that B-spline segments will share control vertices.

Let us now define $Q_i(u)$ on the interval $u \in [0, 1]$ in the following way:

$$Q_i(u) = \Sigma_{k=0}^{3}\ p_{i+k}\ B_k(u)$$

where the B_i's are basis functions.

With Q_{i+1} defined similarly, we can derive the form of the basis functions in order to ensure C^0, C^1, and C^2 continuity. For the C^0 continuity, we have:

$$\Sigma_{k=0}^{3}\ p_{i+k}\ B_k(1) = \Sigma_{k=0}^{3}\ p_{i+k+1}\ B_k(0)$$

Since this must hold for all values of p, we must satisfy the coefficients, giving:

$$\begin{aligned} B_0(1) &= 0 \\ B_1(1) &= B_0(0) \\ B_2(1) &= B_1(0) \\ B_3(1) &= B_2(0) \\ 0 &= B_3(0) \end{aligned}$$

Repeating this process for the C^1 and C^2 conditions yields:

$$\begin{array}{rclrcl} B_0'(1) & = & 0 & B_0''(1) & = & 0 \\ B_1'(1) & = & B_0'(0) & B_1''(1) & = & B_0''(0) \\ B_2'(1) & = & B_1'(0) & B_2''(1) & = & B_1''(0) \\ B_3'(1) & = & B_2'(0) & B_3''(1) & = & B_2''(0) \\ 0 & = & B_3'(0) & 0 & = & B_3''(0) \end{array}$$

This creates an underconstrained system (15 equations with 16 unknowns), so we must enforce an additional constraint. We decide to enforce that the basis functions sum to one at a point, $u = 0$ (necessary for the control polygon to be on the convex hull of the curve).

$$B_0(0) + B_1(0) + B_2(0) + B_3(0) = 1$$

Solving the above system of equations yields the following forms for the bsis functions:

$$\begin{array}{rcl} B_0(u) & = & \frac{(1-u)^3}{6} \\ B_1(u) & = & \frac{(3u^3-6u^2+4}{6} \\ B_2(u) & = & \frac{-3u^3+3u^2+3u+1}{6} \\ B_3(u) & = & \frac{u^3}{6} \end{array}$$

Or in matrix form:

$$Q_i(u) = \frac{1}{6} \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{pmatrix}$$

Now we have an equation for a single segment of the B-spline curve, but how do we generate the entire curve? In order to do this, we must find a global parameterization of the curve. We will call this parameter U. Then, the local parameter, u can be found by $u = U - i$.

From the above derivations, we can see that each curve segment is governed by four control points and that each control point (except for the end control points) influencs four curve segments, $Q_i, Q_{i+1}, Q_{i+2}, Q_{i+3}$, being weighted by the basis functions B_3, B_2, B_1, and B_0, respectively. Moving across a boundary between segments simply causes p_i to be weighted by a different basis function. We can collect these basis functions into one global "blending" function, $N_i(U)$.

$$N_i(U) = \begin{cases} B_3(U - (i-3)) & i-3 \leq U < i-2 \\ B_2(U - (i-2)) & i-2 \leq U < i-1 \\ B_1(U - (i-1)) & i-1 \leq U < i \\ B_0(U - (i-0)) & i \quad \leq U < i+1 \end{cases}$$

In the following section, we will see that this is but a specific instance of a more general class of (nonuniform) B-Splines.

1.2 Knot Vectors

As we saw in the last section, a uniform B-spline is made up of curve segments whose ends are at parametrically uniform intervals (eg. [0,1,2,...]). We call these endpoints knots and term the collection of knots, the knot vector. We now examine the generalization of uniform B-splines by allowing the knot vector to be nonuniform.

Previously, we derived the blending function for cubic B-splines. In an effort to be even more general, we now let the order k of the spline to vary (the degre = k - 1). Therefore, the blending functions $N_i(u)$ become $N_{i,k}(u)$ where $k = 4$ for cubic spline. Once the blending functions are defined, a spline of order k can be defined by:

$$Q(u) = \Sigma_{i=0}^{n}\ p_i\ N_{i,k}(u)$$

The Cox de Boor algorithm is a recursive definition for the blending function $N_{i,k}(u)$ and can be stated as follows:

$$N_{i,1} = \begin{cases} 1 & if\ t_i \leq u < t_{i+1} \\ 0 & otherwise \end{cases}$$

$$N_{i,k}(u) = \frac{(u-t_i)N_{i,k-1}(u)}{t_{i+k-1}\ -\ t_i} + \frac{(t_{i+k}-u)N_{i+1,k-1}(u)}{t_{i+k}\ -\ t_{i+1}}$$

where the t_i's are the nondecreasing sequence of knots.

We can see some important properties from this formulation:

1. $N_{i,k}$ has support over the knot range $[t_i, t_{i+k}]$
2. All $N_{i,k}$ are non-negative

   $$N_{i,k}(u) \geq 0 \qquad forall\ i, k, u$$

   $N_{i,k}$ partition unity.

   $$\Sigma_{i=0}^{n} N_{i,k}(u) = 1$$

   These two properties together mean that the control polygon of a B-spline curve is the convex hull of that same curve.
3. If we define 0/0 to be zero, we allow the possibility of knots having the same value (commonly called multiple knots).

Since $N_{0,k}$ has support of $[t_0...t_k]$ and $N_{n,k}$ has support of $[t_n...t_k]$, there must be $n + k + 1$ knots. One can show:

$$Number\ of\ knots = Number\ of\ Control\ Vertices + order$$

From the above equations, we can also see that each blending function is nonzero over k intervals, and therefore there are at most k blending functions that are

nonzero at any given point. From this, we can see that each curve segment requires k control points for its definition. Therefore, there must be $n-k+2$ distinct curve segments, which implies $n-k+2$ distinct knot intervals, and $n-k+3$ distinct knots. That leaves $(n+k+1)-(n-k+3)=2k-2$ knots left to define.

The most common way to define these extra knots is through the use of a nonperiodic knot vector. This vector forces the curve to interpolate the first and last control points by repeating the end (first and last) knots k-1 more times. The knot vector then has the form:

$$< 0, ..., 0, t_k, ..., t_n, 1, ..., 1 >$$

For additional control of the curve, we allow the control points to be defined in homogenious space. This allows the added flexibility of assigning "weights" to each of the control points. As the w component is increased, the curve is pulled towards the control point. The addition of the w paramter also allows the curves to represent rational functions such as circles. With all of these features, we have defined NURBS, or Nonuniform rational B-splines.

1.3 Knot Insertion

Once we have a NURBS curve defined by a set of conrtol points, a knot vector and an order, we can add a new knot to the knot vector (and with it a new control point) anywhere along the B-spline without changing the shape of the underlying curve. This is extremely useful in modeling since adding a new knot allows more local control of the curve. Therefore, we can insert new knots in the section of the curve that needs refinement, preserving the shape of the other sections of the curve.

Assume the original curve is defined by:

$$Q(u) = \Sigma_{i=0}^{n}\ p_i\ N_{i,k}(u)$$

$$\textit{with the knot vector} < t_0, ..., t_{n+k} >$$

We insert the new knot t' such that $t_j < t' \leq t_{j+1}$.
Then the new control points are given by:

$$p'_0 = p_0,\ p'_{n+1} = p_n,\ \textit{and}\ p'_i = (1-a_i)p_{i-1} + a_i p_i$$
$$a_i = \begin{cases} 1 & i = 1, ..., j-k \\ \frac{t'-t_i}{t_{i+k}-t_i} & i = j-k+1, ..., j \\ 0 & i = j+1, ..., n \end{cases}$$

These new control points leave the original curve unchanged.

2 NURBS Surfaces

Now that we have mastered the use of curves, we extend the notion of NURBS to surfaces. This is a simple extension of NURBS surface in which we separate the two blending function and take the sum.

$$Q(u,v) = \Sigma_{i=0}^{n_u}\Sigma_{j=0}^{n_v}\ p_{ij}\ B_i(u)\ B_j(v)$$

Therefore, where we once had a control polygon, we now have a control mesh. In the next sections, we will look at how to construct models from NURBS surfaces. First we will look at fitting NURBS curves using polar forms. Then we will use these curves and a technique called lofting to create the three-dimensional shape.

2.1 Polar Forms

Any polynomial function $F(u)$ of degree n can be represented by what is known as a polar form, $f(u_1, u_2, ..., u_n)$. These polar forms simplify the construction of curves and surfaces. The blossoming principal states that every polynomial has a unique polar form. Further more, this polar form is symmetric (its three arguments can be written in any order without changing its value), and the polar form is affine in each argument (multiaffine). Thus, the polar form is simply a symmetric multiaffine map of the function $F(u)$ such that $F(u) = f(u, u, ..., u)$.

The polar form for a monomial function of the form

$$F(u) = \Sigma_{i=0}^{n}\ a_i u^i$$

can be given by

$$f(u_1, ..., u_n) = \Sigma_{i=0}^{n}\ a_i \left(\begin{array}{c} n \\ i \end{array} \right)^{-1} \Sigma_{|S|=i, S\subseteq\{1,...,n\}}\Pi_{j\in S}\ u_j$$

For example, the cubic polynomial

$$F(u) = a_0 + a_1 u + a_2 u^2 + a_3 u^3$$

has the polar form

$$f(u_1, u_2, u_3) = a_0 + \frac{a_1}{3}(u_1 + u_2 + u_3) + \frac{a_2}{3}(u_1u_2 + u_2u_3 + u_1u_3) + a_3 u_1 u_2 u_3$$

One of the useful features of the polar form is that it can be used to determine the control points for a NURBS curve fitting the polynomial $F(u)$. If the knot vector is defined as:

$$r_{n+1} \leq ... \leq r_1 < s_1 \leq ... \leq s_{n+1}$$

Then the jth control point can be found to be:

$$p_j = f(r_1, ..., r_{n-j}, s_1, ..., s_j)$$

This result allows us to fit NURBS curves to arbitrary rational functions of polynomials.

2.2 Lofting

Modelling in three dimensions is an extremely hard task. Many different techniques have been made to help the artist model three-dimensional objects from simpler, easier to model objects. One technique is known as lofting. Here, the artist defines two curves, a cross section curve and the path curve. The cross section curve is then lofted, or swept, along the path curve, modulating its size as it goes. Therefore, a shape is created by stacking up many scaled versions of the cross section. Since two-dimensional curves are much easier to create (either by hand or made to fit a surface using techniques such as the previous section) this technique allows artists to create interesting models with much more control and ease.

The act of lofting one curve along another is basically creating a tensor product surface. Let us examine this further by the example of creating a parameterized sphere as a tensor product.

Start with the circle in the x-y plane:

$$\begin{aligned} x_1 &= sin\theta \\ y_1 &= cos\theta \end{aligned}$$

And the semi-circle in the x-z plane:

$$\begin{aligned} x_2 &= sin\phi \\ z_2 &= cos\phi \end{aligned}$$

Now we loft the first circle along the second curve by keeping z_2 and modulating $< x_1, y_1 >$ by x_2.

$$\begin{aligned} x &= sin\phi\ sin\theta \\ y &= sin\phi\ cos\theta \\ z &= cos\phi \end{aligned}$$

Now, we know that transforming the control points will correspondingly transform the underlying curve. Therefore, we can perform the same lofting operation on the control points of two curves, $Q(u)$ and $Q(v)$. If we loft curve $Q(u)$ along $Q(v)$ we get:

$$\begin{aligned} x &= \frac{Q_x(v)Q_x(u)}{Q_w(v)Q_w(u)} \\ y &= \frac{Q_x(v)Q_y(u)}{Q_w(v)Q_w(u)} \\ z &= \frac{Q_z(v)}{Q_w(v)} \end{aligned}$$

where Q_x is the x component of curve Q.

Converting back to homogeneous coordinates:

$$\begin{aligned} x &= Q_x(v)Q_x(u) \\ y &= Q_x(v)Q_y(u) \\ z &= Q_z(v)Q_w(u) \\ w &= Q_w(v)Q_w(u) \end{aligned}$$

which is similar to the previous sphere example when the w components are set to 1.

So, in order to make a NURBS surface, we can simply take two NURBS curves and loft one along the other. This basically amounts to taking the tensor product of the two curves (multiplying the appropriate control vertices together). Now we can model in 3D.