3 Spline Curves and Surfaces

We have talked about points and lines, and perhaps triangles as primitives that we can transform, project, and draw. But these do not make an ideal representation of a smooth curve or surface as too many points are necessary to describe smooth geometry. This is very evident in editing tasks where one does not want to move dozens of points to change the shape of a smooth curve. So instead we will consider other surface representations.

Two main categories of curve and surface representations exist: implicit and parametric. Implicit surfaces are those where the points on the surface are the roots of an equation. Examples of common implicit surface representations are circles in 2D, spheres in 3D, 2D lines ($y = mx + b$), 3d planes ($N \cdot (X - P) = 0$, where N is the surface normal, P is a point on the plane, and $X = (x, y, z)$ is a 3D point. Many of these curves and surfaces also have parametric representations. That is, representations where another variable is used to parametrize points on the surface. A circle can be parameterized as $x(t) = cos(t)$, $y(t) = sin(t)$, and lines can be parameterized as $p(t) = p_0 + vt$ where p_0 is a point on the line and v is a vector giving the direction of the line.

The two different representations have different advantages for different situations. For instance, to answer the question "Compute a point on the surface", a parametric representation provides an easy answer as we can choose an arbitrary parameter value and evaluate, but an implicit surface representation involves finding a root. Alternatively the question "is point X on the surface" is easy with an implicit formulation as we can just substitute and check to see if we get zero, but for a parametric surface we need to search for the parameter values which give this point (a harder problem).

In general, many modeling problems benefit from a parametric form as this form can be quite natural for shape design. We will see some properties below that help to justify this argument.

3.1 Cubic Curves

So far, we have consider simple shapes only: lines, polygons, quadrics. We will increase the complexity of our models by considering smooth cubic curves, which are parametric curves defined by third order polynomials. Such curves are useful for two reasons. First, they can be used to define the paths of points in an animation. These paths can be the trajectory of an object, or they can be the paths of the camera. Second, smooth curves can be used to define smooth (bicubic) surfaces, which we will examine later in the lecture.

3.2 Power Basis

In 3D, the typical form of parametric curve is

$$\mathbf{p}(t) = (x(t), y(t), z(t))$$

with $t \in [0, 1]$, and where we can see each component as being a cubic polynomial, e.g.,

$$x(t) = \sum_{i=0}^{n} \alpha_{xi}\, t^i$$

$$y(t) = \sum_{i=0}^{n} \alpha_{yi}\, t^i$$

$$z(t) = \sum_{i=0}^{n} \alpha_{zi}\, t^i$$

with $n = 3$.

The coefficients of the different terms entirely describe the shape of the curve. We can write the curve as

$$p(t) = \begin{bmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix} \begin{bmatrix} t^0 \\ t^1 \\ t^2 \\ t^3 \end{bmatrix} = G_{power} I \begin{bmatrix} t^0 \\ t^1 \\ t^2 \\ t^3 \end{bmatrix}. \tag{1}$$

The coefficients, α, we can see as forming the columns of a matrix G, which we label with a "power" subscript to say that this describes the geometry of the curve in the power basis. In reality, the power basis has geometric meaning, but is not ideal of shape editing as it largely describes the behaviour of the curve at $t = 0$. That is, the first column can be seen as the point of the curve at $t = 0$, with the second column describes the tangent at $t = 0$, and the subsequent columns describe the values of higher derivatives.

Lets look at a different means of describing the shape of a curve that has more geometric intuition

3.3 Interpolating Curves

Suppose we want to define a 3D curve that passes through three points $\mathbf{p}(0), \mathbf{p}(1), \mathbf{p}(2) \in \Re^3$. This is relatively easy. We can always fit a quadratic to any three points. And for a cubic curve, we would need four points. As above, we let each of the coordinates of our curve $p(t) = (x(t), y(t), z(t))$ be a cubic polynomial, that is, a third order polynomial in parameter t, for example,

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

and similarly for $y(t)$ and $z(t)$.

Since each polynomial has four coefficients, we need to specify 12 coefficients in total. Since we have 12 degrees of freedom, we might guess that we can specify a curve by chosing four points in $\Re^3$ (since $4 \times 3 = 12$). This is correct, as we show next.

We define a curve $\mathbf{p}(t)$ by choosing four points in $\Re^3$ and defining these points to be *on the curve* and at values $t = 0, 1, 2, 3$, that is, the points are $\mathbf{p}(0), \mathbf{p}(1), \mathbf{p}(2), \mathbf{p}(3)$. How do we do it? Let's define $\mathbf{p}(t)$ to be a column vector:

$$\mathbf{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \tag{2}$$

Solving for the twelve a, b, c, d coefficients turns out to be easy. Just make a 3×4 matrix from the four points and substitute the four values $t = 0, 1, 2, 3$:

$$\begin{bmatrix} \mathbf{p}(0) & \mathbf{p}(1) & \mathbf{p}(2) & \mathbf{p}(3) \end{bmatrix} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix} \begin{bmatrix} 0 & 1 & 8 & 27 \\ 0 & 1 & 4 & 9 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

We then solve for the coefficients by right-multiplying by a matrix

$$\mathbf{B} = \begin{bmatrix} 0 & 1 & 8 & 27 \\ 0 & 1 & 4 & 9 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1}$$

i.e.

$$\begin{bmatrix} \mathbf{p}(0) & \mathbf{p}(1) & \mathbf{p}(2) & \mathbf{p}(3) \end{bmatrix} \begin{bmatrix} 0 & 1 & 8 & 27 \\ 0 & 1 & 4 & 9 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix}$$

We refer to the 3×4 matrix with the $\mathbf{p}()$ values as the *geometry matrix* $\mathbf{G_{interpolation}}$, or the matrix of *control points*. We rewrite Eq. (2) as:

$$\mathbf{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \mathbf{G_{interpolation}} \mathbf{B_{interpolation}} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}.$$

The product

$$\mathbf{B_{interpolation}} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

is a 4×1 column vector whose rows (single elements) are *blending functions*. These are four (third order) polynomials in t which define the contribution of each of the four control points $\mathbf{p}(0), \mathbf{p}(1), \mathbf{p}(2), \mathbf{p}(3)$ to the curve point $\mathbf{p}(t)$.

Finally, we can approximate the curve $\mathbf{p}(t)$ using a set of lines. This can be done by dividing the interval $t \in [0,3]$ into pieces (possibly of equal length, possibly not) and substituting these t values into the equation for $\mathbf{p}$(t).

Note that we defined our curve on the interval $[0,3]$, but we could have easily done this instead on the interval $[0,1]$ by setting the two middle points to be at t values $1/3$ and $2/3$. Likewise, we could have said that the curve would go through the selected points at different parameter values, e.g., $1/4$ and $3/4$, and this will change the shape of the curve! Likewise, notice that we are really just defining a cubic polynomial, and it is still defined beyond the interval of interest.

3.4 Hermite Curves

A similar approach is to take two points $\mathbf{p}(0)$ and $\mathbf{p}(1)$ and to choose the tangent vector at each of these two points. By *tangent vector*, I mean the derivative with respect to the parameter t:

$$\mathbf{p}'(t) \equiv \frac{d\mathbf{p}(t)}{dt} \equiv lim_{\delta t \to 0} \frac{\mathbf{p}(t+\delta t) - \mathbf{p}(t)}{\delta t}$$

Intuitively, if t were interpreted as time, then the tangent vector would be the 3D velocity of the curve. This interpretation is relevent, for example, in animations if we are defining the path of a camera or the path of a vertex.

From Eq. (2), we observe that

$$\mathbf{p}'(t) = \begin{bmatrix} x'(t) \\ y'(t) \\ z'(t) \end{bmatrix} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix} \begin{bmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{bmatrix}$$

Substituting for $t = 0$ and $t = 1$ for $\mathbf{p}(t)$ and $\mathbf{p}'(t)$ gives us a 3×4 matrix:

$$\begin{bmatrix} \mathbf{p}(0) & \mathbf{p}(1) & \mathbf{p}'(0) & \mathbf{p}'(1) \end{bmatrix} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$

Again, we can compute the coefficient matrix by right multiplying by the inverse of the 4×4 matrix at the right end of the above equation. Define this inverse to be

$$\mathbf{B}_{Hermite} \equiv \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}^{-1}$$

and define the 3×4 *geometry matrix* (or *control points*)

$$\mathbf{G}_{Hermite} \equiv \begin{bmatrix} \mathbf{p}(0) & \mathbf{p}(1) & \mathbf{p}'(0) & \mathbf{p}'(1) \end{bmatrix}$$

then we can write

$$\mathbf{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \mathbf{G}_{Hermite}\mathbf{B}_{Hermite} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}. \tag{3}$$

The 4×1 column vector

$$\mathbf{B}_{Hermite} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

contains four *Hermite blending functions*. These four functions are each third order polynomials in t, and define the contribution of the four control points $\mathbf{p}(0), \mathbf{p}(1), \mathbf{p}'(0), \mathbf{p}'(1)$, respectively, to the curve $\mathbf{p}(t)$. Note that these blending functions themselves are independent of the chosen control points.

In practice, we often want to draw a complicated curve in which we have n points $\mathbf{p}(0)$ to $\mathbf{p}(n-1)$ and we have tangents $\mathbf{p}'(0)$ to $\mathbf{p}'(n-1)$ defined at these points as well. We would like to fit a curve (say Hermite) between each pair of points. We can be sure that the curve passes through the points and that the tangents are continuous, simply by making sure that the endpoint and tangent of one curve fragment (say, from $t = k-1$ to $t = k$) is the same as the beginning point and tangent of the next curve fragment (from $t = k$ to $t = k+1$).

3.5 Bezier

In the interpolation example above, the product of $B(t^0, t^1, t^2, t^3)^T$ can be seen as blending weights, or interpolation weights. For a given value of t, these weights can be seen as saying take so much of each of these points, and add them together. If the sum of the weights is always 1, then we have an affine combination and all the benefits that come with such a combination (see properties below).

In Bezier curves, these weights come from Bernstein polynomials. These polynomials are given by

$$B_{in}(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{4}$$

which can be seen as a binomial expansion of $(1+(1-t))^n$, which is simply equal to 1^n. We must be careful at this point not to confuse the polynomials $B_{in}(t)$ with the matrices B we have defined above (it should be clear from context).

By expanding the cubic Bernstein polynomials, we can see that

$$p(t) = G_{Bezier} B_{Bezier} (t^0, t^1, t^2, t^3)^T$$

where

$$B_{Bezier} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bezier curves have many nice properites.

- The sum of the Bernstein polynomials is 1, which means it is an affine combination, but we also have $B_{in}(t) \geq 0$ for $t \in [0,1]$, so this is a convex combination too! This means the curve never leaves the area defined by the convex hull of its control points.
- Affine invariance. Given any affine map, the image of the curve under this map is the same as the curve produced by transforming the control points (the contents of the geometry matrix G_{Bezier}) by this map.
- $B_{0n}(0) = B_{nn}(1) = 1$, meaning the curve interpolates the two ends of the control points.
- $B_{in}(t)$ attains only one maximum on $[0,1]$ at $t = i/n$, meaning edits to the curve by moving the control point have a maximum effect at only one point.
- The Bernstein functions are symmetric about $t = 0.5$, i.e., $B_{in}(t) = B_{(n-i)n}(1-t)$, which means we can change the order of the control points (i.e., reverse them) and get the same curve parameterized in the opposite direction.
- The Bernstein polynomials have a recursive definition, which leads to efficient evaluation methods, and the same is true for their derivatives.
- The k^{th} derivative at $t = 0$ depends only on the first $k+1$ control points (and likewise the last $k+1$ control points for the k^{th} derivative at $t = 1$).

3.5.1 DeCasteljau Algorithm

There is a simple algorithm for finding a point $t \in [0,1]$ on a Bezier curve. First, divide each segment of the control polygon into two parts (ratio t to $(1-t)$, or if $t = 0.5$, then divide each in half). Second, connect the new subdivided points to get a shorter control polygon. Repeat until there is only one point, and this will be a point on the curve.

3.6 Changing Basis

Since all the curves we are considering above are cubic polynomials, all of the different representations are interchangeable and can be seen as a change of basis. To convert a Bezier basis into an interpolation basis, consider

$$\begin{aligned} p(t) &= G_{Bezier}B_{Bezier}(t^0,t^1,t^2,t^3)^T \\ &= G_{Bezier}B_{Bezier}(B^{-1}_{interpolation}B_{interpolation})(t^0,t^1,t^2,t^3)^T \\ &= (G_{Bezier}B_{Bezier}B^{-1}_{interpolation})B_{interpolation}(t^0,t^1,t^2,t^3)^T \end{aligned}$$

thus,

$$G_{interpolation} = G_{Bezier}B_{Bezier}B^{-1}_{interpolation}.$$

3.7 Piecewise Curves and Continuity

So far, we've been only looking at a cubic curved defined on a small interval. But in practice it is far more interesting to put many of these small curves one after another to make a larger curve. This combination is called a *piecewise polynomial*, and it is useful as we can edit one portion of the curve without having an effect on other portions. But some effort must be made to keep the curve continuous!

In ensuring continuity, if all the endpoints connect from one to the next segment, then the curve can be said to have C_0 continuity. If the derivative vectors also match, then there is parametric continuity of the derivatives and we can say that the curve is C_1. But it is also interesting to talk only about the shape of the curve. If two adjoining curves share the same tangent, then they can be said to have G_1 or geometric continuity. Note that the two parametric derivatives could be different magnitues, but as long as the point in the same direction then we will have geometric continuity. Higher order parametric continuity (e.g., C_2 for piecewise cubic curves) is common, but the notion of higher order Geometric continuity can get somewhat complicated and is out of the scope of this document (see [Barsky and Derose 1988] if you are curious).

Note that for Hermite curves, C_1 continuity is easy to ensure by making sure the end points and end derivatives match starting points and derivatives. For Bezier curves a similar rule can be applied by matching end points, and ensuring that the two last control points of one curve form a vector in the same direction as the two first control points of the next curve. We will see shortly that there are potentially better ways to get around this problem of ensuring continuity with the use of other basis functions.

3.8 Subdivision of Polynomial Curves

Given a curve $p(t)$, consider $p_L(t) = p(\frac{1}{2}t)$ and $p_R(t) = p(\frac{1}{2}+\frac{1}{2}t)$. These *left* and *right* curves have the same shape as the original (it is a simple change of variable), but they are not the same curve. The main difference is that the interval $t \in [0,1]$ only gives us the first half, or the second half of the original curve, and the derivative vectors are smaller.

While we have not specified this subdivision with respect to a given basis, it is usually the task which needs to be performed. That is, we have control points for a Bezier curve, and we would like to cut it into two segments. This is useful as it allows one part of the curve to be edited while leaving the other half unchanged.

This subdivision task can also be seen as a simple change of basis. We can also figure out the weights relating original control points to the left and right control points by examining the intermediate points of the DeCaseljau algorithm (this was shown in class and can be found in reference material).

3.9 Rational Bezier Splines

Rational Bezier curves have the form

$$p(t) = \frac{\sum B_{in}(t) w_i g_i}{\sum B_{in}(t) w_i} \tag{5}$$

where $g_i = (x_i, y_i, z_i)$ and w_i is a weight. We can relate this exactly to the homogeneous representation of points, and many of the desire able properties of Bezier curves still apply (e.g., convex hull property, and affine invariance).

If all the weights are equal to 1 we end up with the Bezier case, but if weights are increased, then the curve of a particular control point is increased then the curve will be pulled towards that control point. Likewise, lowering the weight to zero will reduce the influence of that control point to nothing.

The big power of rational curves is that they fit easily with the framework of homogeneous coordinates and homogeneous transformations, but also allow us to represent circular curves (something which is impossible with a non-rational polynomial).

Note that the following rational quadratic on the interval $t \in [0,1]$ is equivalent to a quarter circle.

$$x(t) = \frac{1-t^2}{1+t^2}$$
$$x(t) = \frac{2t}{1+t^2}$$

This can be seen by substituting $t = tan(u/2)$, and working through with trig identities until $x(u) = cos(u)$ and $y(u) = sin(u)$.

3.10 Bicubic Surfaces

We next turn from curves to surfaces. We wish to define a two parameter surface:

$$\mathbf{p}(s,t) = (x(s,t), y(s,t), z(s,t))$$

which maps

$$\mathbf{p} : \Re^2 \rightarrow \Re^3.$$

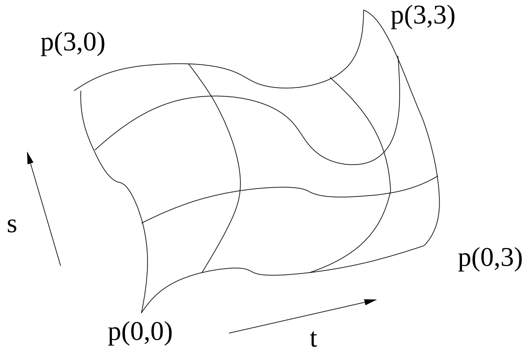
such that each x, y, z is a polynomial in s, t of (maximum) degree 3, and $\mathbf{p}(s,t)$ passes through a given set of 3D data points (see below). The surface **p**(s,t) is a *parametric bicubic* surface with parameters s and t, in the following sense. For any fixed s, the curve $\mathbf{p}(s,t)$ is a bicubic function of t and, for any fixed t, the curve $\mathbf{p}(s,t)$ is a bicubic function of s.

The idea for constructing this surface follows from what saw earlier this lecture. Any four distinct points in $\Re^3$, define a cubic curve that passes through these four points. To define a surface, we use four sets of four points, giving us four cubic curves. We then take corresponding points on the four cubic curves and for each of them we define a cubic curve that passes through these four corresponding points. The math behind this is as follows.

We fit a *bicubic surface* such that it passes through a grid of 4×4 points. These sixteen points are the intersections of the eight curves shown in the following sketch. The eight curves correspond to four values

$(0,1,2,3)$ for each of the s,t variables that define the surface. Only the four corner points of the surface patch are labelled in the figure.

Earlier we solved the problem simultaneously for the x,y,z functions. But if you re-examine how we did this, you can see that solutions $x(t), y(t), z(t)$ were independent. So let's just look at one of these functions, namely $x(s,t)$.



Fix the parameter s (we will eventually substitute $s = 0,1,2,3$) and define a cubic curve $x(s,t)$ for $t = 0,1,2,3$. Applying the solution from earlier, we get

$$x(s,t) = \begin{bmatrix} x(s,0) & x(s,1) & x(s,2) & x(s,3) \end{bmatrix} \mathbf{B} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \tag{6}$$

Next, suppose we define four curves by choosing $s = 0,1,2,3$ and stacking them. The x component of these four curves would be:

$$\begin{bmatrix} x(0,t) \\ x(1,t) \\ x(2,t) \\ x(3,t) \end{bmatrix} = \begin{bmatrix} x(0,0) & x(0,1) & x(0,2) & x(0,3) \\ x(1,0) & x(1,1) & x(1,2) & x(1,3) \\ x(2,0) & x(2,1) & x(2,2) & x(2,3) \\ x(3,0) & x(3,1) & x(3,2) & x(3,3) \end{bmatrix} \mathbf{B} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \tag{7}$$

The left hand side is interesting. For any t, it is a four-tuple. To fit a cubic to this four-tuple (for fixed t), we apply the solution as before, but now the curve parameter is s.

The only minor subtlety is that the four-tuple is written a column vector whereas previously we wrote it as row vector. So to use the same form as Eq. 6 (but using s as the parameter instead of t), we need to take the transpose, i.e. we consider

$$x(s,t) = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} \mathbf{B}^T \begin{bmatrix} x(0,t) \\ x(1,t) \\ x(2,t) \\ x(3,t) \end{bmatrix} \tag{8}$$

Substituting Eq. (7) into Eq. (8), we get:

$$x(s,t) \;=\; \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}^T \mathbf{B}^T \begin{bmatrix} x(0,0) & x(0,1) & x(0,2) & x(0,3) \\ x(1,0) & x(1,1) & x(1,2) & x(1,3) \\ x(2,0) & x(2,1) & x(2,2) & x(2,3) \\ x(3,0) & x(3,1) & x(3,2) & x(3,3) \end{bmatrix} \mathbf{B} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}.$$

which is a parametric bicubic function. The 4×4 $x(*,*)$ matrix is a geometrix matrix which we can call $\mathbf{G}_x$. We do do the same thing to define $y(s,t)$ and $z(s,t)$ via geometry matrices $\mathbf{G}_y$ and $\mathbf{G}_z$.

Note that bicubic surface patches can also be defined with other bases too. Bezier bicubic patches can be written as

$$p(s,t) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_{i3}(s)B_{j3}(t)g_{ij} \tag{9}$$

$$= \sum_{i=0}^{3} B_{i3}(s)\left(\sum_{j=0}^{3} B_{j3}(t)g_{ij}\right) \tag{10}$$

$$\tag{11}$$

where g_{ij} give the 16 control points of the 4 by 4 control grid. The second line of the equation above highlights that you can consider the bicubic Bezier patch to be the result of first evaluating four Bezier curves, and using the resulting points as control points for a new Bezier curve.

3.10.1 Surface Normals

As we will see later in the course, it is often very useful to know the surface normal for any (s,t). How could we define this? Once we have the geometry matrice $\mathbf{G}_x, \mathbf{G}_y, \mathbf{G}_z$, we can compute tangent vectors to the surfaces for any (s,t), namely

$$\frac{\partial}{\partial t}x(s,t) = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} \mathbf{B}^T \, \mathbf{G}_x \, \mathbf{B} \begin{bmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{bmatrix}.$$

and

$$\frac{\partial}{\partial s}x(s,t) = \begin{bmatrix} 3s^2 & 2s & 1 & 0 \end{bmatrix} \mathbf{B}^T \, \mathbf{G}_x \, \mathbf{B} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

The same form is used for the partial derivatives of $y(s,t)$ and $z(s,t)$.

Since these two partial derivatives are both tangent to the surface (and are not identical), the surface normal must be perpendicular to both of these tangent vectors. Hence, the surface normal is parallel to the cross product of these two partial derivatives.

$$\frac{\partial}{\partial s}\mathbf{p}(s,t) \times \frac{\partial}{\partial t}\mathbf{p}(s,t)$$

We will use this surface normal in later lectures e.g. when we discuss bump mapping.

3.11 A quick intro to B-Splines

If we have a long curve (or a larger surface) that we are tyring to edit, we don't always want the control points to influence the entire shape. Instead we would like the control points to have some sort of local support. This we described earlier with the idea of using piecewise cubic curves, but lets now define a basis that lets us move control points as we like without the headache of trying to maintain continuity between piecewise segments when using the Bezier basis.

We give up interpolation of the endpoints, but by doing so we can define a curve instead as

$$p(t) = \sum_{i=-\infty}^{\infty} g_i N_i(t) \tag{12}$$

where $N_i(t) = N_0(t-i)$ where $N_0(t)$ is a function with limited support (i.e., is zero outside of some interval) defining the interpolation, and of course $\sum_{i=-\infty}^{\infty} N_i(t) = 1$ for any value of t (i.e., it is an affine combination). Note that because N_0 has limited support is that any point on the curve only depends on the position of a few control points g_i, specifically those for which the function $N_i(t)$ is nonzero.

This is a fairly informal definition, up to this point, but we can state this formally by defining a knot vector $[u_0, u_1, ...u_m]$, a non decreasing sequence of real numbers. The u_i are called knot values, and if the values in the knot vector are an increasing sequence of integers (i.e., $[0,1,2,...m]$), then we fall into the case of a *uniform* knot vector and this matches the description of B-splines above. If the knot vector has repeated values, or non-integer values, then we have a *non uniform* knot vector and a non uniform b-spline, and we no longer have this function $N_0(t)$ which describes the shape of all the basis functions. Note that in the case of rational curves, then we are actually dealing with NURBS, i.e., non uniform rational b splines.

The basis functions are defined recursively starting with a box function.

$$N_{i,0}(t) = \begin{cases} 1 & t \in [u_i, u_{i+1}] \\ 0 & \text{otherwise} \end{cases}$$

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

For uniform knot vectors and degree $p = 0$, the function $N_{i,p}(t)$ is a box, and the curve is a piecewise constant function (discontinuous jumps at every integer value t). When $p = 1$, the basis function is a triangular hat function, and the resulting curve is piecewise linear.

This may not be too intuitive, but there is another interpretation of how the basis functions can be constructed (which may or may not help). In the case of uniform knot vectors the basis functions ban be seen as a convolution of box functions, i.e.,

$$\begin{aligned} N_{0,1}(t) &= (N_{0,0} \star N_{0,0})(t) \\ &= \int_{-\infty}^{\infty} N_{0,0}(t) N_{0,0}(t-s)\, ds, \\ N_{0,p}(t) &= (N_{0,p-1} \star N_{0,0})(t) \end{aligned}$$

Each time we increase the degree p we increase the size of the interval on which the degree p basis function is nonzero.

Some important facts on how B-splines are used is that repeating control points is a method for ensuring the curve interpolates a point. For instance, for a cubic, knot vector [0,1,2,3,4,5,6,7] and a control point sequence $g_0 = Q$, $g_1 = Q$, $g_2 = Q$, $g_3 = R$, will give a curve on interval $[3,4]$ that goes through point Q at $t = 3$. This is another way to get the curve to go through selected points by using a non-uniform knot vector that contains several repeated knot values. For instance, for a cubic, knot vector $[0,0,0,0,1,1,1,1]$ and a control point sequence g_0, g_1, g_2, g_3 will give a curve which interpolates g_0 and g_3 on interval $t \in [0,1]$ (this is actually equivalent to the Bezier curve).

In the case of a cubic, because the curve is defined by a sub sequence of the control points, we can end up looping the control points to close the curve, for instance, for a uniform cubic b-spline, given g_0, g_1, g_2, g_3, choosing $g_3 = g_0$, $g_4 = g_1$, $g_5 = g_2$, will give us piecewise cubic curve segments that close the curve into a loop.

3.12 A quick intro to Polar Forms and Blossoming

A polar form (or blossom) of a polynomial $p(t) : R \rightarrow R$ of degree n is a *multiaffine* and *symmetric* function

$$P(t_1, t_2, ...t_n) : R^n \rightarrow R$$

such that

$$P(t, t, ...t) = p(t).$$

That is, plugging the same value t into the polar form n times will give us back our polynomial.

A function $f(t_1...t_n)$ is symmetric if it gives the same value for any permutation of its parameters. The function is multiaffine if for any arguments $t_1, ...t_j^0, t_j^1, ...t_n$ and any value α,

$$f(t_1, ..., \alpha t_j^0 + (1-\alpha)t_j^1, ...t_n) = \alpha f(t_1, ..., t_j^0, ...t_n) + (1-\alpha) f(t_1, ..., t_j^1, ...t_n)$$

For any polynomial, there exists a unique symmetric and multiaffine form which is not difficult to construct. Here is an example. For the cubic curve

$$p(t) = \sum_{i=0}^{3} a_i t^i$$

the polar form is given by

$$P(t_1, t_2, t_3) = a_0 + a_1 \frac{(t_1 + t_2 + t_3)}{3} + a_2 \frac{(t_1 t_2 + t_2 t_3 + t_1 t_3)}{3} + a_3 t_1 t_2 t_3$$

Polar forms are interesting because the original polynomial can be determined from its polar form evaluated with different combinations of 0 and 1, i.e., $P(0,0,0), P(0,0,1), P(0,1,1), P(1,1,1)$. Since these values uniquely determine the cubic polynomial, we can consider this to be the cubic function represented in some basis, which in this case turns out to be the Bezier basis. Another interesting example is $P(0,1,2), P(1,2,3), P(2,3,4), P(3,4,5)$ which gives the representation of the polynomial on interval $t \in [2,3]$ in the B-spline basis with knot vector $[-1,0,1,2,3,4,5,6]$.

Why does this knot vector have -1 and 6 on either end? Well, a cubic curve is considered order 4, and require at least 4 control points, and the knot vector has the number of control points plus the order of the curve (i.e., 8). The intuition here is that these end values of the knot vector are important for defining the basis functions, and are important for ensuring continuity with adjacent curves, but these values have no effect on the shape of the curve in the interval $[2,3]$.

3.13 Surfaces of Revolution

We can create surfaces in a variety of other ways. A common technique for creating a surface is to define a surface of revolution. This works by taking a curve

$$c(u) = \begin{pmatrix} c_x(u) \\ c_y(u) \\ c_z(u) \end{pmatrix}$$

and then rotating its points about an axis, say the z axis, giving

$$s(u,\theta) = R_z(\theta)p(u) = \begin{pmatrix} cos(\theta)c_x(u) - sin(\theta)c_y(u) \\ sin(\theta)c_x(u) + cos(\theta)c_y(u) \\ c_z(u) \end{pmatrix}$$

That is, we simply rotate the (x,y) coordinate by the z axis.

3.14 Swept Surfaces

Another common surface the swept surface. That is, take a curve $c(u)$ and then sweep it through space. We can think of this first in the simple case of just translating the surface along some other curve, say $p(v)$, to get

$$s(u,v) = c(u) + p(v)$$

but if the curve p turns, then we should change the orientation of c(u) to follow, thus

$$s(u,v) = R(v)c(u) + p(v)$$

Where R is a rotation that can be thought of as transforming coordinates of some local frame containing the curve $c(u)$ to a world frame, and it is parameterized by v as we will let this rotation change along the path of the curve. One common way of defining this rotation is via the Frenet frame or parallel transport frame of the curve, as defined in the following seciton.

3.15 Frenet Frame and Parallel Transport Frame

This frame is defined by the tangent and the second derivative as follows

$$\begin{aligned} T(v) &= \frac{p'(v)}{||p'(v)||} \\ B(v) &= \frac{p'(v) \times p''(v)}{||p'(v) \times p''(v)||} \\ N(v) &= B(v) \times T(v) \end{aligned}$$

or alternatively, more naturally in the TNB order by computing N by removing the non-tangent-orthogonal component of p'' from p''

$$\begin{aligned} T(v) &= \frac{p'(v)}{||p'(v)||} \\ N(v) &= \frac{p''(v) - (p''(v) \cot T(v))T(v)}{p''(v) - (p''(v) \cot T(v))T(v)} \\ B(v) &= T(v) \times N(v) \end{aligned}$$

The T, N, and B vectors, given in world coordinates, define a local coordinate system on the curve. The normal vector points in the direction of the osculating circle, which is the circle which matches the shape of the curve at the given point. The radius of the osculating circule is $1/\kappa$ where κ is the curvature. Note that

$$\kappa = \frac{||p'(t) \times p''(t)||}{||p(v)||^3} \quad \text{or } \kappa = ||p''_a(s)||$$

where $p_a(s)$ is the arc length parameterization of the curve.

Unfortunately, the Frenet frame is not always defined, as we can have the second derivative equal to zero, or pointing in the same direction as the first, or the first derivative can even vanish. An example of when this comes up is when a curve is completely straight. Other problems that can arise is that the Frenet frame can suddenly flip when there are discontinuities in the second derivative (this can certainly happen often with piecewise curves).

An alternative to the Frenet frame is the parallel transport frame. This frame keeps one axis in the direction of the tangent (i.e, T(v) is the same as for the Frenet frame), but selects an arbitrary perpendicular vectors U_0 and V_0, perpendicular to the tangent at the beginning of the curve, i.e., $U(0) = U_0$ and $V(0) = V_0$. $T(0)$, $U(0)$, and $V(0)$ form a coordinate frame as above, but as we move along the curve, the U and V vectors make minimal rotations while staying perpendicular to the tangent. If we discretize over small steps h along the curve, we can see this rotation as having an axis A in the direction of the cross product of two subsequent tangents, and the angle α as the angle between the two tangents. That is,

$$\begin{aligned} A &= T(v) \times T(v+h) \\ \alpha &= \cos^{-1}(T(v) \cdot T(v+h)) \\ V(v+h) &= R(A,\alpha)U(v) \\ U(v+h) &= R(A,\alpha)T(v), \end{aligned}$$

where $R(A,\alpha)$ is the rotation by α around the axis A.

The only problem with the parallel transport frame is that if you close a loop, you may not end up with the same frame as you started. To fix this you would need to add some small amount of twist along the curve (or in places where it would be least noticable) in order to make the frames line up at the end points. In contrast, for the Frenet frame you will get the same frame since it only depends on the paramaterization of the curve at the current parameter value.

3.16 Arc Length

Sometimes it is useful to know how long a curve is, or likewise how long a curve is on an interval. Given a curve $p(t)$ for $t \in [0,1]$, then the length of the curve is

$$\int_0^1 ||p'(u)||du.$$

We can define a function to give us the length from $t = 0$ to a given parameter value t as

$$L(t) = \int_0^t ||p'(u)||du.$$

Note that there is no closed form integral, and this needs to be computed numerically. If we define the inverse arc length function as

$$l(s) = L^1(s), \;\; s \in [0, L(1)]$$

then we can write our original curve with an arc length parameterization as

$$p_a(s) = p(l(s)).$$

The arc length parameterization has the nice property that the derivative has a magnitude of one at every point on the curve (thus the tangent is simply the derivative $T(s) = p'_a(s)$).

Arc length parameterized curves, or curves with constant speed in general, are of interest for animation (for instance, for moving along a path at a constant speed) and tessellation (breaking a curve into regular linear segments for rendering). An interesting observation is that B-splines with non uniform knot vectors allow a curve to be defined with less speed variation (though it is generally not possible to adjust the knots to produce a NURBS curve with a natural arc length parameterization).

3.17 Subdivision Curves and Surfaces

The idea behind a subdivision curve (or surface) is to repeatedly refine the control polygon. The general algorithm proceeds as follows.

1. Start with a piecewise linear curve
2. insert points at the midpoint of each edge
3. apply an affine averaging mask to each vertex
4. repeat (i.e., goto step 2) until satisfied

As an optional final step, it is possible to push the positions of vertices to where they would be in the limit (i.e., on the limit curve). For curves, Chaikin's corner cutting algorithm (1974) is a nice example which results in a quadratic B-spline. Given a piecewise linear curve, you first divide the edges, and then you average each point with the neighbour on its right (i.e., an affine combination of 1/2 and 1/2 of each vertex). What do we do with the end points? We could treat them as special cases and create new "rules" for them. The easiest thing to do is leave them fixed. Alternatively, if we have a closed curve that loops, then there are no end points and no need for special rules.

Instead of just averaging two points, we can instead apply larger arbitrary *masks* where we do a weighted average (specifically, an affine combination) of our neighbours. We call it an averaging mask because it computes an average using only some of the values (i.e., the mask hides or removes the influence of far away points by setting their weights to zero. We could write the mask as $\boldsymbol{r} = (...r_{-1},\ r_0,\ r_1,\ ...)$, and note we have $(\frac{1}{2}\ \frac{1}{2})$ for Chaikin's algorithm. The Lane-Riensfeld algorithm (1980) uses masks that come from Pascal's triangle,

$$\boldsymbol{r} = \frac{1}{2^n}\left(\begin{pmatrix} 0 \\ n \end{pmatrix}, \begin{pmatrix} 1 \\ n \end{pmatrix}, \dots \begin{pmatrix} n \\ n \end{pmatrix}\right)$$

and it turns out that if you choose masks in this manner then you get B-splines of degree n (order $n+1$) in the limit.

Note that we generally need to subdivide an infinite number of times (this is unlike the Decastlejau algorithm discussed previously) to get the limit curve. I say generally because if it is flat then we could stop early. So, where does a vertex go to in the limit? Lets look at the Lane-Riensfeld algorithm for $n = 2$, that is,

$$\boldsymbol{r} = \left(\frac{1}{4},\ \frac{1}{2},\ \frac{1}{4}\right), \tag{13}$$

which we are told will converge to a cubic polynomial curve. First notice what happens if we combine steps two and three. For now, let us write p_i^j to denote point i on the curve at subdivision level j. Suppose we have $n+1$ points at level 0, then we will have p_i^0 for $i = 0..n$, and p_i^1 for $i = 0..2n$. As a first step in combining the steps, let us just notice what happens at *even* and *odd* indices at the next subdivision level when we insert the new vertices (i.e., step 2). The odd vertices are new and take on the average of the two endpoints of the edge, while the even vertices are copied. We will denote this first version of the subdivision curve after step 2 as $\tilde{p}$, thus,

$$\begin{aligned}\tilde{p}^{j+1}_{2i+1} &= \frac{1}{2}p^j_i + \frac{1}{2}p^j_{i+1},\\ \tilde{p}^{j+1}_{2i} &= p^j_i.\end{aligned}$$

For both the even and odd case, we can apply the mask in Equation 13 to get the final result,

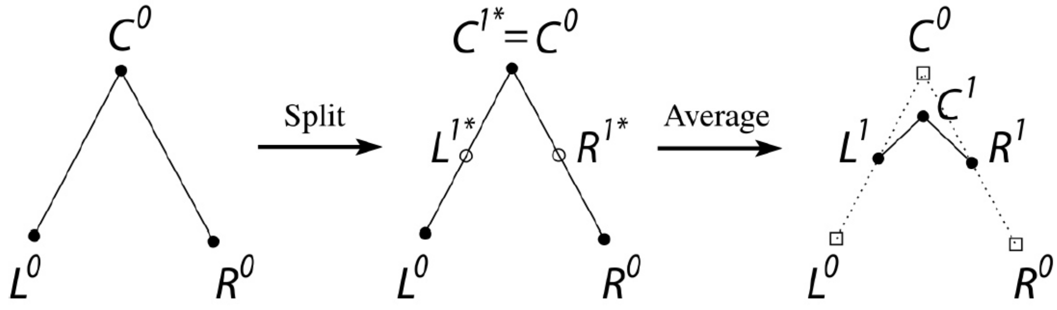
$$\begin{aligned}p^{j+1}_{2i+1} &= \frac{1}{4}\tilde{p}^{j+1}_{2i} + \frac{1}{2}\tilde{p}^{j+1}_{2i+1} + \frac{1}{4}\tilde{p}^{j+1}_{2i+2},\\ p^{j+1}_{2i} &= \frac{1}{4}\tilde{p}^{j+1}_{2i-1} + \frac{1}{2}\tilde{p}^{j+1}_{2i} + \frac{1}{4}\tilde{p}^{j+1}_{2i+1}.\end{aligned}$$

Combining the two steps we can expand to get the even and odd rules written with respec to vertices at the previous subdivision level.

$$\begin{aligned}p^{j+1}_{2i+1} &= \frac{1}{2}p^j_i + \frac{1}{2}p^j_{i+1},\\ p^{j+1}_{2i} &= \frac{1}{8}p^j_{i-1} + \frac{6}{8}p^j_i + \frac{1}{8}p^j_{i+1}.\end{aligned}$$

We will also call these even and odd rules "masks", but you should not confuse them with the ones that we discussed above.

Now lets consider what happens in a small neighbourhood around the vertex in question.



Note that the left and right points don't move after the splitting step, while the center vertex takes on new position in the averaging step based on the weights $\mathbf{r} = (\frac{1}{4}\ \frac{1}{2}\ \frac{1}{4})$. Combining the two steps (as we did moment ago), we can write a subdivision matrix to describe how this small neighbourhood moves as we repeat the process.

$$\begin{pmatrix}L^j\\ C^j\\ R^j\end{pmatrix} = \begin{pmatrix}\frac{1}{2} & \frac{1}{2} & 0\\ \frac{1}{8} & \frac{6}{8} & \frac{1}{8}\\ 0 & \frac{1}{2} & \frac{1}{2}\end{pmatrix}\begin{pmatrix}L^{j-1}\\ C^{j-1}\\ R^{j-1}\end{pmatrix}$$

Let $X^j = (L^j, C^j, R^j)^T$, and if you are thinking of these points in 2D or 3D, then consider X^j to contain the different coordinates of these points in the rows (though we would normally express them as columns). So, we can say that we want to find,

$$X^\infty = \lim_{j\to\infty} S^j X^0$$

That is, apply S repeatedly to X^0 forever. We can expect this to converge otherwise the subdivision scheme is not too useful. Note that in the limit, the left, center, and right points will be the same but we can worry about extracting the middle row of X^∞. The natural approach to solving this problem is to compute the eigenvalue decomposition of the subdivision matrix S. In this case, we have

$$\begin{aligned} S &= V\Lambda V^{-1} && (14)\\ &= \begin{pmatrix} \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{2}} & -\frac{2}{3} \\ \frac{1}{\sqrt{3}} & 0 & \frac{1}{3} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & -\frac{2}{3} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix} \begin{pmatrix} \frac{1}{2\sqrt{3}} & -\frac{2}{\sqrt{3}} & \frac{1}{2\sqrt{3}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{2} & \frac{1}{\sqrt{2}} & -\frac{1}{2} \end{pmatrix} && (15) \end{aligned}$$

Note that the first eigenvalue is one, which means it converges, and it doesn't converge to zero. The first eigenvalue is always one in all subdivision schemes. Thus,

$$\begin{aligned} S^\infty &= V\lambda^\infty V^{-1} \\ &= \begin{pmatrix} \frac{1}{\sqrt{3}} & * & * \\ \frac{1}{\sqrt{3}} & * & * \\ \frac{1}{\sqrt{3}} & * & * \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2\sqrt{3}} & -\frac{2}{\sqrt{3}} & \frac{1}{2\sqrt{3}} \\ * & * & * \\ * & * & * \end{pmatrix} \end{aligned}$$

where the stars denote that we no longer care about these values. Reading off the center row of the product (or any row for that matter as they are all the same), we can write,

$$C^\infty = \begin{pmatrix} \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} L^0 \\ C^0 \\ R^0 \end{pmatrix}$$

This same analysis applies for computing the tangent of the curve, which we can approximate at any level by $R^j - C^j$, but we'll normalize the length otherwise it will be zero in the limit.

$$T^\infty = \lim_{j\to\infty} \frac{R^j - C^j}{||R^j - C^j||}$$

Notice that the subtraction of R^j and C^j involves subtracting two rows of V,

$$\begin{aligned}
C^j &= \begin{pmatrix} \frac{1}{\sqrt{3}} & 0 & \frac{1}{3} \end{pmatrix} S^j V^{-1} X^0 \\
R^j &= \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & -\frac{2}{3} \end{pmatrix} S^j V^{-1} X^0 \\
R^j - C^j &= \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & -1 \end{pmatrix} S^j V^{-1} X^0 \qquad \text{Look! a zero! Now combine with } S^j, \\
R^j - C^j &= \begin{pmatrix} 0 & \frac{1}{\sqrt{2}}(\frac{1}{2})^j & -(\frac{1}{4})^j \end{pmatrix} V^{-1} X^0 \qquad \text{and if we pull out the 1/2 to the j,} \\
R^j - C^j &= \frac{1}{2^j} \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & -(\frac{1}{2})^j \end{pmatrix} V^{-1} X^0 \qquad \text{and notice the magnitude,} \\
||R^j - C^j|| &= \frac{1}{2^j} || \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & -(\frac{1}{2})^j \end{pmatrix} V^{-1} X^0 || \qquad \text{Thus...} \\
T^\infty &= \lim_{j \to \infty} \frac{\frac{1}{2^j} \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & -(\frac{1}{2})^j \end{pmatrix} V^{-1} X^0}{\frac{1}{2^j} \left\| \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & -(\frac{1}{2})^j \end{pmatrix} V^{-1} X^0 \right\|} \\
&= \lim_{j \to \infty} \frac{\begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & -(\frac{1}{2})^j \end{pmatrix} V^{-1} X^0}{\left\| \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & -(\frac{1}{2})^j \end{pmatrix} V^{-1} X^0 \right\|} \\
&= \frac{\begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 \end{pmatrix} V^{-1} X^0}{\left\| \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} & 0 \end{pmatrix} V^{-1} X^0 \right\|} \\
&= \frac{\begin{pmatrix} -\frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix} X^0}{\left\| \begin{pmatrix} -\frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix} X^0 \right\|}
\end{aligned}$$

So, the tangent weighting comes from the second row of V^{-1}, and in the end, the weighting is not so surprising (i.e., right minus left makes a better approximation of the tangent than right minus center). The same analysis is actually much simpler if you start out with the right point minus the left point to approximate the tangent (this is a useful excercise you can try on your own).

This same analysis applies for surfaces too. Recall again that we call the two rules even and odd for the new vertices and the exising vertices, even though the indices no longer have this property of being even and odd. This only words in the 1D curve case: if we start numbering from zero, then after the splitting step, all the even numbered vertices are the old ones, while the odd numbered vertices are the new ones. There are many different subdivision schemes for surfaces, like there are many for curves. Some interpolate, while others approximate, and they have varying degrees of continuity. Catmull-Clark subdivision works on quad meshes, while Loop works on triangle meshes (note that you can break a triangle into quads, or a quad into triangles as a pre-processing step). These two scheme produce surfaces which are equivalent to splines everywhere except near extraordinary vertices. Extraordinary vertices in a quad mesh are those of valence (i.e., degree) not equal to 4, while for a triangle mesh it is those vertices of valence not equal to 6.

For Loop and Catmull Clark subdivision masks, please look at the SIGGRAPH 2000 course notes on subdivision surfaces (pages 49, 70, 76)

`http://www.cs.nyu.edu/∼dzorin/sig00course/`

Note that special rule modifications are made for boundaries, and these modifications are also useful for defining creases or sharp features on the original non-subdivided mesh.

Acknowledgments

These lecture notes are largely based on those prepared by Mike Langer, and make use of other sources, including MIT Open CourseWare, and University of Washington course notes.

Change History

September 28, 2009. First Posted September 28, 2009. Updates (added sections 3.11 to 3.13) October 1, 2009. Updates (added material, added some section numbers) October 4, 2010. Updates (to subdivision material)