11/8/2002
CS 258
HW 2

1. Given n+1 points, $x_0, x_1, \ldots, x_n$ and a sequence of n+1 parameters $u_0, u_1, \ldots u_n$ with $u_i < u_i+1$ for all I, our goal is to compute a C^2 cubic spline curve F(u), such that $F(u_i) = x_i$ for all $I \le i \le n$

A) If we use cubic B-spline curves (and compute the appropriate control points), how will you choose the knot sequence? Is your resulting problem under-constrained or over-constrained?

We can choose the knot sequence to be the same as the parameter values. That is if our knots are $t_0, t_1, \ldots t_n$ then we would set them as: $t_0 = u_0, t_1 = u_1, \ldots t_n = u_n$. Then the control points can be positioned to meet the x_i. At worst, the control points could be replicated (3 times) in order to force the curve to interpolate the points. Since we know that n+k+1 = # of control points + degree of cure + 1 = n + 4 + 1 = # of knots = T (Mortenson 97), the number of knots must be greater than 5. We know we have an absolute lower bound on the number of knots we need and hence number of parameters we need to make a cubic B-Spline curve useful for interpolation.

The resulting problem is under-constrained because we have not specified any tangent information for the points even though the curve is capable of C^2 continuity and we have not specified any endpoint information besides positional (so if this curve were part of a larger group we would not have enough information to match it up.

B) If the problem is under-constrained, add sufficient number of tangent conditions at the end points and derive the control points of the resulting [cubic] B-spline curve.

Using Farin 1990 as our guide, we can rewrite every B-spline curve as a piecewise Bezier curve. Hence we can then use the interpolating properties of Bezier curves for each piece and convert them into a form that can be used in the B-spline.

Since it is a cubic curve, the Bezier curve relies on a neighbor on either side to exactly interpolate the point. We take every third point in the Bezier curve to be the exact point we wish to interpolate (namely x_i). In other words, we take:

$$x_i = b_{3i} \qquad i = 0,\ldots,n$$

But then we must fill in the neighbors. To be C^1 continuous at a particular knot (u_i), the Bezier control point xi must be in the same ratio as the knots (parameters) on either side of it. To be more concise and confusing, let's define $\Delta_i = u_{i+1} - u_i$ where the u_i are meant to be the knots but since in part A we took the knots and parameters to be the same, we can just use u_i for both. Then we can say that:

$$b_{3i} = \frac{u_{i+1} - u_i}{(u_i - u_{i-1}) + (u_{i+1} - u_i)} b_{3i-1} + \frac{u_i - u_{i-1}}{(u_i - u_{i-1}) + (u_{i+1} - u_i)} b_{3i+1} \quad \text{or}$$

$$x_i = b_{3i} = \frac{\Delta_i}{\Delta_{i-1} + \Delta_i} b_{3i-1} + \frac{\Delta_{i-1}}{\Delta_{i-1} + \Delta_i} b_{3i+1}$$

This formula says that b_{3i}, the control point for our intermediate Bezier form, is a weighted average of its neighbors in proportion to their relative distance in knot space. We only have a parameter for the b_{3i} and the neighboring points are added in.

The neighboring points can be related to the control vertices in the curve itself through

$$b_{3i-1} = \frac{\Delta_i d_{i-1} + (\Delta_{i-2} + \Delta_{i-1}) d_i}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i} \qquad i = 2 \ldots n-1$$

where the d_i and d_{i-1} are control vertices on the B-spline curve we are seeking from Farin 1990. Similarly,

$$b_{3i+1} = \frac{(\Delta_i + \Delta_{i+1})d_i + \Delta_{i-1}d_{i+1}}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}} \quad i = 1 \ldots n-2$$

So we now have a method of relating the control points on the B-spline curve to the Bezier control points interpolating the data points we were given originally. However, there are still some issues with the end points because they are not specified with neighbors and such. We are free to come up with our own tangents (say based on x_1-x_0 and x_n-x_{n-1}). Using our formulation of Bezier control points in terms of B-spline control points, we can say:

$$b_2 = \frac{\Delta_1 d_0 + \Delta_0 d_1}{\Delta_0 + \Delta_1} \text{ and } b_{3L-2} = \frac{\Delta_{L-1}d_{L-1} + \Delta_{L-2}d_L}{\Delta_0 + \Delta_1}$$

Now that we have all the b's in terms of d's, we can use our definition of x to get the direct correspondence between the data points x_i and the B-spline control points.
We know:

$$x_i = b_{3i} = \frac{\Delta_i}{\Delta_{i-1} + \Delta_i} b_{3i-1} + \frac{\Delta_{i-1}}{\Delta_{i-1} + \Delta_i} b_{3i+1} = \frac{\Delta_i b_{3i-1} + \Delta_{i-1} b_{3i+1}}{\Delta_{i-1} + \Delta_i} \quad i = 1 \ldots n-1$$

So:

$$(\Delta_{i-1} + \Delta_i)x_i = \Delta_i b_{3i-1} + \Delta_{i-1} b_{3i+1} \quad i = 1 \ldots n-1$$

But we have definitions for the b's. Substituting those in we get:

$$(\Delta_{i-1} + \Delta_i)x_i = \Delta_i b_{3i-1} + \Delta_{i-1} b_{3i+1} \quad i = 1 \ldots n-1$$

$$(\Delta_{i-1} + \Delta_i)x_i = \Delta_i \frac{\Delta_i d_{i-1} + (\Delta_{i-2} + \Delta_{i-1})d_i}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i} + \Delta_{i-1} \frac{(\Delta_i + \Delta_{i+1})d_i + \Delta_{i-1}d_{i+1}}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}} \quad i = 1 \ldots n-1$$

Collecting like terms gives us:

$$(\Delta_{i-1} + \Delta_i)x_i = \left(\frac{(\Delta_i)^2}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i}\right)d_{i-1} +$$

$$\left(\frac{\Delta_i(\Delta_{i-2} + \Delta_{i-1})}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i} + \frac{\Delta_{i-1}(\Delta_i + \Delta_{i+1})}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}\right)d_i + \left(\frac{(\Delta_{i-1})^2}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}\right)d_{i+1}$$

Where for items like Δ_{i-2} and Δ_{i+1} we set $\Delta_{-1} = \Delta_L = 0$. To simplify the notation we can label the coefficients of the B-spline control points as:

$$\alpha_i = \frac{(\Delta_i)^2}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i},\ \beta_i = \frac{\Delta_i(\Delta_{i-2} + \Delta_{i-1})}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i} + \frac{\Delta_{i-1}(\Delta_i + \Delta_{i+1})}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}, \text{ and } \gamma_i = \frac{(\Delta_{i-1})^2}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}$$

Then we can say more concisely,

$$(\Delta_{i-1} + \Delta_i)x_i = \alpha_i d_{i-1} + \beta_i d_i + \gamma_i d_{i+1}$$

so we would have a system of equations such as we progress along i:

$$(\Delta_0 + \Delta_1)x_1 = \alpha_1 d_0 + \beta_1 d_1 + \gamma_1 d_2$$

$$(\Delta_1 + \Delta_2)x_2 = \alpha_2 d_1 + \beta_2 d_2 + \gamma_2 d_3$$

$$\ldots$$

At the end points, we can data points for control points. We can specify the system in matrix form as:

$$\begin{bmatrix} 1 & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & \\ & & \ddots & & & \\ & & & \alpha_{n-1} & \beta_{n-1} & \gamma_{n-1} \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} = \begin{bmatrix} b_1 \\ (\Delta_0 + \Delta_1)x_1 \\ \vdots \\ (\Delta_{n-2} + \Delta_{n-1})x_{n-1} \\ b_{3n-1} \end{bmatrix}$$

The first and last polygon vertices are $d_{-1} = x_0$ and $d_{n+1} = x_n$. Hence now we have specified the entire set of control points given a set of data points and parameters.

2. Find the Bezier control points of a closed B-spline cure of degree 4 whose control polygon consists of the edges of a square, and whose knot sequence is uniform and consists of simple knots.

A uniform, simple knot sequence looks like (0,1,2,3,4… n). However, the curve is closed. Therefore, we must replicate some control points if we are to achieve a closed curve. The degree is 4 so the polynomial governing the B-spline has a form of $a_0x^4+a_1x^3…+a_4$. In this case, what we call k will be k=5 since k=1 contains just x^0, k=2 contains x^1 and so on.

A **uniform** quartic B-spline curve can be represented in a matrix form as:

$$p(u)=\frac{1}{24}\begin{bmatrix}u^4 & u^3 & u^2 & u & 1\end{bmatrix}\begin{bmatrix}1 & -4 & 6 & -4 & 1\\ -4 & 12 & -12 & 4 & 0\\ 6 & -6 & -6 & 6 & 0\\ -4 & -12 & 12 & 4 & 0\\ 1 & 11 & 11 & 1 & 0\end{bmatrix}\begin{bmatrix}p_{i-1}\\ p_i\\ p_{i+1}\\ p_{i+2}\\ P_{i+3}\end{bmatrix} \qquad \text{for } i\in[1:n-2]$$

or more succinctly $p_i(u) = UM_SP_k$. Also since our curve is closed, we place 'mod' on each control point. So we really use the equation: $p_i(u)=UM_S\begin{bmatrix}p_{i-1}\bmod (n+1)\\ p_i \bmod (n+1)\\ p_{i+1}\bmod (n+1)\\ p_{i+2}\bmod (n+1)\\ p_{i+3}\bmod (n+1)\end{bmatrix}$ but the P_k hides this.

where each p_i is a control point and $p_i(u)$ is the curve for the segment between knots i and i+1 (Mortenson). A Bezier curve can similarly be rewritten in matrix form as:

$$p(u)=\begin{bmatrix}u^4 & u^3 & u^2 & u & 1\end{bmatrix}\begin{bmatrix}1 & -4 & 6 & -4 & 1\\ -4 & 12 & -12 & 4 & 0\\ 6 & -12 & 6 & 0 & 0\\ -4 & 4 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}p_0\\ p_1\\ p_3\\ p_4\\ p_5\end{bmatrix}$$

or more succinctly $p(u) = UM_BP$.

If we are to describe the same curve, the we have $UM_BP_B = UM_SP_S$ or equivalently $M_BP_B = M_SP_S$ so to convert from B-spline to Bezier control points, we have the equation:

$$P_B = M_B^{-1}M_SP_S$$

$$P_B=\frac{1}{24}\begin{bmatrix}0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & \frac{1}{4} & 1\\ 0 & 0 & \frac{1}{6} & \frac{1}{2} & 1\\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{3}{4} & 1\\ 1 & 1 & 1 & 1 & 1\end{bmatrix}\begin{bmatrix}1 & -4 & 6 & -4 & 1\\ -4 & 12 & -12 & 4 & 0\\ 6 & -6 & -6 & 6 & 0\\ -4 & -12 & 12 & 4 & 0\\ 1 & 11 & 11 & 1 & 0\end{bmatrix}P_S=\frac{1}{24}\begin{bmatrix}1 & 11 & 11 & 1 & 0\\ 0 & 8 & 14 & 2 & 0\\ 0 & 4 & 16 & 4 & 0\\ 0 & 2 & 14 & 8 & 0\\ 0 & 1 & 11 & 11 & 1\end{bmatrix}P_S$$

So given any control points on the edges of a square, we can place them through the above equation and get our Bezier control points. The mod function for P_S makes the curve closed and prevent us from choosing points that aren't in the original set of control points.

==

The calculation of the B-spline matrix was a bit trying so I will recount the steps here:
The derivation follows Mortensen p123-124 but for higher degrees. The first step is to choose an interval I where k ≤i≤n. This is a generic interval to calculate symbolically on. The t_i knot values are chosen conveniently as ti = i-k+1 = i-5+1 = i-4. Then we enumerate the various base N values as:

$$N_{i,1}(u) = 1 \quad for\ i-4 \le u < i-3 \quad or\ N_{i,1}(u) = 0\ otherwise$$

$$N_{i+1,1}(u) = 1 \quad for\ i-3 \le u < i-2 \quad or\ N_{i,1}(u) = 0\ otherwise$$

$$N_{i+2,1}(u) = 1 \quad for\ i-2 \le u < i-1 \quad or\ N_{i,1}(u) = 0\ otherwise$$

$$N_{i+3,1}(u) = 1 \quad for\ i-1 \le u < i \quad or\ N_{i,1}(u) = 0\ otherwise$$

$$N_{i+4,1}(u) = 1 \quad for\ i \le u < i+1 \quad or\ N_{i,1}(u) = 0\ otherwise$$

$$N_{i+5,1}(u) = 1 \quad for\ i+1 \le u < i+2 \quad or\ N_{i,1}(u) = 0\ otherwise\ \ldots$$

Then the formula $N_{i,k}(u) = \frac{(u-t_i)N_{i,k-1}(u)}{t_{i+k-1}-t_i} + \frac{(t_{i+k}-u)N_{i+1,k-1}(u)}{t_{i+k}-t_{i+1}}$ is applied for: $N_{i,5}(u)$, $N_{i+1,5}(u)$, $N_{i+2,5}(u)$, $N_{i+3,5}(u)$, $N_{i+4,5}(u)$ where the N are evaluated using the following Mathematica commands:

```
t[i_] := i - 5 + 1;
NC2[i_, k_] :=
 (((u - t[i]) * (NC2[i, k - 1])) / (t[i + k - 1] - t[i])) +
  (((t[i + k] - u) NC2[i + 1, k - 1]) / (t[i + k] - t[i + 1]))
NC2[i_, 1] := MC[i, 1]
```

Using these commands, derived again from Mortenson's discussion, allows us to evaluate any $N_{i,k}(u)$ in terms of the base $N_{i,1}$, $N_{i+1,1}$, $N_{i+2,1}$ …
Then we choose a representative interval for one of the function to be 1 over. For this case, the interval i ≤ u < i + 1 was chosen which means that only the terms with $N_{i+4,1}$ had to be evaluated. The evaluation was done with the following Mathematica command:

```
Expand[24 * NC2[i, 5] /.
  {MC[i, 1] → 0, MC[i + 1, 1] → 0, MC[i + 2, 1] → 0,
   MC[i + 3] → 0, MC[i + 4, 1] → 1, MC[i + 5, 1] → 0,
   MC[i + 6, 1] → 0, i → 0}]
```

This provides the coefficients for one column of the matrix. The other columns are determined by evaluating NC2[i+1, 5], NC2[i+2,5], NC2[i+3,5], and NC2[i+4,5]. The 24 comes from empirically looking at the output of the unexpanded form. Just as the cubic has a 1/6th and the quadratic has a ½, the quartic has its own constant factor, namely 1/24.

3. Given a m x n tensor product patch, P(u,w), there are three possible ways to evaluate a point. Work out the operation count for each of these cases.

Since in class we covered on the evaluations necessary for a Bezier Tensor Product Patch, I will consider only Bezier patches for the operation counts.

* Use the recursive de Casteljau algorithm

There are two cases for the de Casteljau algorithm, one in which the patch has equal control points in each dimension and one in which there is an addition bilinear interpolation.

As an example of a 2 point interpolation, we have a form such as:

$$(1-t)p_{0,0} + tp_{0,1}$$

This evaluation takes 1 subtraction, 1 addition of (x,y,z) values = 3 additions, 2 multiplications of (x,y,z) values = 6 multiplications. So S = 1, A = 3, M = 6.

As an example, a 4 point application of the direct de Casteljau algorithm will result in a formula such as:

$$p(s,t) = (1-s)((1-t)p_{0,0} + tp_{0,1}) + s((1-t)p_{1,0} + tp_{1,1})$$

This evaluation requires 2 subtractions ((1-s) and (1-t)), 9 additions ($p_{i,j} = \lfloor x_{i,j} \quad y_{i,j} \quad z_{i,j} \rfloor$) so any time we add two p's we do 3 additions and since we have $p_{0,0}$+$p_{0,1}$ and $p_{1,0}$ + $p_{1,1}$ and then sum those we get (3+3+3 = 9), finally there are 18 multiplications. (1-t)*$p_{0,0}$ = 3 and each p has one so 3*4=12 and then we multiply by the s's for each so 3+3 = 6 and 12 + 6 = 18 multiplications. S = 2, A = 9, M = 18.

I will take each square to take 9 adds and 18 multiplies since a recursive function (with no memory) would perform the same actions.

Looking at the matrix form, we have:

$$b_{i,j}^{r,r} = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} b_{i,j}^{r-1,r-1} & b_{i,j+1}^{r-1,r-1} \\ b_{i+1,j}^{r-1,r-1} & b_{i+1,j+1}^{r-1,r-1} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}$$

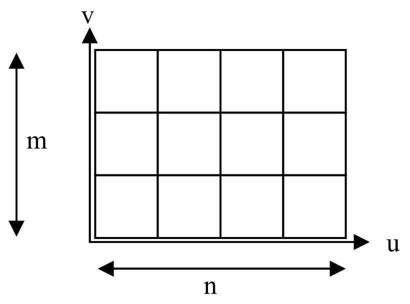
In this form, we can store the two subtractions and keep them globally so there will always be 2 subtractions. However, in evaluating $b_{i,j}^{r,r}$, we eventually come down to one or more four point cases or one or more two point cases.

Now, we can generalize this result for an m x n patch. For the special case of an n x n, we know there are not going to be any lines for interpolation and we know the number of squares (4 point cases). So since we know we will have $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ squares each of which takes 9 adds and 18 multiplies (plus our constant 2 subtracts).

For the case in which n ≠ m we have more options. We know the number of squares is given by the formula f(m,n) = m*n * f(m-1, n-1) with f(0,n) = 0 and f(m,0) = 0. So f(4,2) = 4*2 + 3*1 + 2*0= 8 + 3 = 11 squares. However, we also know that if n ≠ m we will have to interpolate some lines. In fact, we must interpolate $\sum_{i=1}^{m-n} i = \frac{(m-n)(1+m-n)}{2}$ lines. So we have 2 subtractions, f(m,n)*9 additions, f(m,n) * 18 multiplies + ((m-n)(1+m-n)/2) * 3 additions + ((m-n)(1+m-n)/2) * 6 multiplies in total.

* Use the tensor product formulation. Compute the coefficients of a u isoparametric line, and then evaluate that curve at v.

We are given an m x n patch such as:



In this problem, we evaluate all the Berstein polynomials first, multiply them by their coefficients for each curve along m and then calculate the Berstein polynomial for the line across u and multiply it by the coefficients to get the points on (u,v). This is somewhat different than the method discussed in class which essentially performs a de Casteljau iteration for each curve to get the u control points and then performs de Casteljau on that curve to get the final point on the curve.

In this method, we first calculate the Berstein polynomial. We assume that the degree of the curve is the same across the surface. The polynomials are computed for the given v and for all $0 \leq i \leq n$. The Berstein polynomial is given by:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Assume that the binomial coefficients are **precomputed** and do no factor into the operation count. Evaluating the polynomial uses 1 subtraction of (1-t) which can be stored and used over and over. Next we must compute powers of t, t^i implies that t is multiplied i times (using a simple implementation) and (1-t) is multiplied n-i times and then we multiply the binomial so in total we have i+n-i+1 = n+1 multiplies and one subtract so n+2 total operations (not counting binomials).

After doing that, we have a Berstein polynomial that can be used for all the curves on the v axis. However, we must evaluate the coefficients and multiply them against the Berstein as:

$$x(t) = \sum_{i=0}^{n} B_i^n(t) p_i$$

So we perform n multiplies and n-1 adds to compute a single point on a curve. Each add takes 3 additions and each multiply takes 3 multiplications. So the total operation count is (n+1)*(3) + 3n = 6n+3.

In our case along the m axis, we really have **m+2** operations to calculate the Berstein polynomial and 6m+3 operations to calculate a point on one of the curves. However, there are n curves so to generate all the points on all the curves takes **n*(6m+3)** operations.

Now we have all the points on a isoparametric line on u. Now we must compute the Berstein in this direction. This requires **n+2** operations. Evaluating a point on the u curve takes **6n+3** operations.

Therefore, in total we have: 1) calculate Berstein: m+2 operations 2) calculate points on curves: n*(6m+3) operations 3) calculate other Berstein: n+2 4) evaluate a point on there: 6n+3 for a grand total of:
# ops total = m+2 + n*(6m+3) + n+2 + 6n+3.

* Use the tensor product formulation. Compute the coefficients of a v isoparametric line, and then evaluate that curve at u.

This is just the same as the previous argument with n and m swapped. Therefore the operation count is:

# ops total = n+2 + m*(6n+3) + m+2 + 6m+3

4. Given a **rational Bezier** patch of degree n, P(u,v), give an algorithm to compute the control points (and associated weight) of its normal or hodograph patch, N(u,w). Given the parameter values (u0, w0), N(u0, w0) denotes the normal at P(u0, w0). N(u0, w0) doesn't represent a normalized vector. I am not expecting a closed form solution in terms of weights and control points of P(u,w), but show all the steps used to compute a representation of the normal patch in terms of Bernstein polynomials.

Given a rational Bezier patch, I will show how to convert it to a normal patch (hopefully).

First, a rational Bezier patch is given by: $P(s,t)=\sum_{i=0}^{n}\sum_{j=0}^{m} b_{i,j}\frac{w_{i,j}B_i^n(s)B_j^m(t)}{\sum_{k=0}^{n}\sum_{l=0}^{m} w_{k,l}B_k^n(s)B_l^m(t)}$

We will proceed as follows:
1) Show that the addition and multiplication (or really any basic operation) on a Bezier curve results in another Bezier curve. This is needed when partial derivatives are calculated to provide some argument that they are also Bezier curves. Also for cross products.
2) Compute the partial derivative P_s and P_t and argue that they are rational Bezier curves still
3) Compute P_s x P_t (abstractly) and argue that that is also a rational Bezier curve

Since n = P_s x P_t then we have constructed the normal patch for the rational Bezier patch and so have new weights and control points to use.

1) Addition and multiplication on a Bezier curve results in another Bezier curve
Any two Bezier curves have the form:

$$A(t)=\sum a_iB_i^n=\left(a_0(1-t)^n+....+a_nt^n\right)$$

$$B(t)=\sum b_iB_i^m=\left(b_0(1-t)^m+b_1(1-t)^{m-1}t....+b_mt^m\right)$$

When we multiply two of them together, we get something like:

$A(t)B(t)=a_0b_0(1-t)^{n+m}+....(...)(1-t)^{n-k+m-j}t^{k+j}+...+(a_nb_m)t^{n+m}$ where k+j≤m+n and k,j≥0

The main point being that the terms are still powers of (1-t) and t together which means that they correspond still to a Bezier curve formulation. Adding also just intermingles terms without altering the basic Bezier properties, we have just altered like terms by summing coefficients. For *rational* Bezier, the a and b terms can be seen as $a_i^{'}=a_iw_i$. In other words, the coefficients also contain the weights.

2) Compute partial derivatives P_s and P_t
To make the calculation simpler, we divide the fraction into its numerator and denominator. Namely:

$N(s,t)=\sum_{i=0}^{n}\sum_{j=0}^{m} b_{i,j}w_{i,j}B_i^n(s)B_j^m(t)$ and $D(s,t)=\sum_{k=0}^{n}\sum_{l=0}^{m} w_{k,l}B_k^n(s)B_l^m(t)$

Then, consulting Farin 1990, the derivatives can be computed using the following formulas:

$$\frac{\partial}{\partial u}b^{m,n}(u,v)=m\sum_{j=0}^{n}\sum_{i=0}^{m-1}\Delta^{1,0}b_{i,j}B_i^{m-1}(u)B_j^n(v)$$

where $\Delta^{1,0}b_{i,j}=b_{i+1,j}-b_{i,j}$

$$\frac{\partial}{\partial v}b^{m,n}(u,v)=n\sum_{i=0}^{m}\sum_{j=0}^{n-1}\Delta^{0,1}b_{i,j}B_j^{n-1}(u)B_i^m(v)$$

where $\Delta^{0,1}b_{i,j}=b_{i,j+1}-b_{i,j}$

Applying these formulas to N and D result in:

$$N_s(s,t) = \sum_{j=0}^{n}\sum_{i=0}^{m-1}(w_{i+1,j}p_{i+1,j} - w_{i,j}p_{i,j})B_i^{m-1}(s)B_j^n(t)$$

$$D_s(s,t) = \sum_{j=0}^{n}\sum_{i=0}^{m-1}(w_{i+1,j} - w_{i,j})B_i^{m-1}(s)B_j^n(t)$$

$$N_t(s,t) = \sum_{j=0}^{n-1}\sum_{i=0}^{m}(w_{i,j+1}p_{i,j+1} - w_{i,j}p_{i,j})B_i^{m}(s)B_j^{n-1}(t)$$

$$D_t(s,t) = \sum_{j=0}^{n-1}\sum_{i=0}^{m}(w_{i,j+1} - w_{i,j})B_i^{m}(s)B_j^{n-1}(t)$$

Since we are adding and multiplying to achieve the differentiation, the partials are also Bezier curves. Now we can use these terms to compute the partials for the whole expression. We have:

$$P_s(s,t) = \left(\frac{N}{D}\right)_s = \frac{N_s(s,t)D(s,t) - N(s,t)D_s(s,t)}{D(s,t)^2}$$

where the quotient rule was used since N/D form a quotient.

$$P_t(s,t) = \left(\frac{N}{D}\right)_t = \frac{N_t(s,t)D(s,t) - N(s,t)D_v(s,t)}{D(s,t)^2}$$

So now that we have the partials (in Bezier form since they are formed from basic operations), then we can try and phrase the normal map as a composition of these elements.

3) Compute P_s x P_t

If we take the results from part 2, we can say that the unnormalized normal map is:

$$NM(s,t) = \frac{\left(N_s(s,t)D(s,t) - N(s,t)D_s(s,t)\right)\times(N_v(s,t)D(s,t) - N(s,t)D_t(s,t))}{D(s,t)^2}$$

The cross product can be calculated by breaking the terms into their components. We know that we can rephrase P(s,t) as P(s,t) = [x(s,t) y(s,t) z(s,t)]. So $P_s(s,t) \times P_t(s,t)$= [$x_s(s,t)$ $y_s(s,t)$ $z_s(s,t)$] × [$x_t(s,t)$ $y_t(s,t)$ $z_t(s,t)$] where the patch has been split into its x, y and z components Then the cross product becomes:

$$\begin{aligned} n &= \left[x_s(s,t) \quad y_s(s,t) \quad z_s(s,t)\right]\times\left[x_t(s,t) \quad y_t(s,t) \quad z_t(s,t)\right] \\ &= \left[(y_s z_t - y_t z_s) \quad (z_s x_t - x_s z_t) \quad (x_s y_t - y_s x_t)\right] \end{aligned}$$

Since these components are all multiplications and subtractions (negative additions) then we can also call the normal patch generated by the NM formula a Bezier patch as well.

5. Degree elevation of a triangular patch. Given a triangular patch of degree n, the degree elevation formula for this patch is given as:

$$\sum_{|i|=n} b_i B_i^n(u) = \sum_{|i|=n+1} b_i^{(1)} B_i^{n+1}(u)$$

show that the control points $b_i^{(1)}$ are given as:

$$b_i^{(1)} = \frac{1}{n+1}\left[ib_{i-e1} + jb_{i-e2} + kb_{i-e3}\right]$$

There is a lot of notation introduced by these equations which is worthwhile to explain. |i|=i+j+k. So saying |i|=n means that i+j+k add up to n with none of them going negative. Therefore, the summations are more complicated than a straight indexing scheme. The i subscript refers to (i,j,k) as a way of identifying control points on the triangle.E1 = (1,0,0), e2 = (0,1,0), e3 = (0,0,1) are representative of the unit axes. U = (u,v,w) which are coordinates on the triangle patch. The Berstein polynomial is also redefined as:

$$B_{\vec{i}}^n(\vec{u}) = \binom{n}{\vec{i}} u^i v^j w^k = \frac{n!}{i!j!k!} u^i v^j w^k \quad |\vec{i}| = n$$

Also $B_{\vec{i}}^n(\vec{u}) = 0 \quad if\ i,j,k < 0\ or\ i,j,k > n$

$\vec{u} = (u,v,w)$ but u+v+w=1 so the function is bivariate.

Since u+v+w=1, we can place is anywhere in the equation for degree elevation and we will still have a legitimate formula (since we are essentially multiplying by 1). So we reformulate the degree elevation formula as:

$$(u+v+w)\sum_{|i|=n} b_i B_i^n(\vec{u}) = \sum_{|i|=n+1} b_i^{(1)} B_i^{n+1}(\vec{u})$$

Then we can compare like powers to define elevated control points in terms of the original control points. One thing to keep in mind when looking at the formula is that the right side has |i|=n+1 while the left side has |i|=n.

On the right hand side of the equation, we will have a term as:

$$b_i^{(1)} B_i^{n+1}(u,v,w) = b_i^{(1)} \frac{(n+1)!}{i!j!k!} u^i v^j w^k \text{ where i+j+k=n+1}$$

However, with i+j+k=n+1 we can rephrase this to find the corresponding terms on the left hand side. In particular, if i+j+k=n+1, we have three choices: (i-1)+j+k=n, i+(j-1)+k=n, or i+j+(k-1)=n. These indices correspond to specific terms in the summation. So on the right hand side we can state:

$$(u+v+w)\sum_{|i|=n} b_i B_i^n(\vec{u})$$

$$= (u+v+w)\left(\ldots + b_{i-1,j,k}\tfrac{n!}{(i-1)!j!k!}u^{i-1}v^j w^k + b_{i,j-1,k}\tfrac{n!}{i!(j-1)!k!}u^i v^{j-1} w^k + b_{i,j,k-1}\tfrac{n!}{i!j!(k-1)!}u^i v^j w^{k-1} + ..\right)$$

but then multiplying by (u+v+w) will create terms that are equal in u,v,w powers as our term on the right hand side. Multiplying out the u+v+w results in the following terms in the summation sequence:

$$\ldots + b_{i-1,j,k}\frac{n!}{(i-1)!j!k!}u^i v^j w^k + \ldots b_{i,j-1,k}\frac{n!}{i!(j-1)!k!}u^i v^j w^k + \ldots b_{i,j,k-1}\frac{n!}{i!j!(k-1)!}u^i v^j w^k + ..$$

Granted the terms will not appear together, but the distribution of + over * says that each term will eventually be multiplied by u, v, and w separately.

Now that we have terms of equal power on the left and right side, we can then solve the equality for $b_i^{(1)}$ to get our sought after result. Equating like terms (and eliminating the common $u^i v^j w^k$), we have:

$$b_{i-1,j,k}\frac{n!}{(i-1)!\,j!\,k!}+b_{i,j-1,k}\frac{n!}{i!\,(j-1)!\,k!}+b_{i,j,k-1}\frac{n!}{i!\,j!\,(k-1)!}=b_i^{(1)}\frac{(n+1)!}{i!\,j!\,k!}$$

so solving for $b_i^{(1)}$

$$b_i^{(1)}=b_{i-1,j,k}\frac{n!}{(i-1)!\,j!\,k!}\frac{i!\,j!\,k!}{(n+1)!}+b_{i,j-1,k}\frac{n!}{i!\,(j-1)!\,k!}\frac{i!\,j!\,k!}{(n+1)!}+b_{i,j,k-1}\frac{n!}{i!\,j!\,(k-1)!}\frac{i!\,j!\,k!}{(n+1)!}$$

simplifying

$$b_i^{(1)}=b_{i-1,j,k}\frac{i}{n+1}+b_{i,j-1,k}\frac{j}{n+1}+b_{i,j,k-1}\frac{k}{n+1}$$

or as the problem states it:

$$b_i^{(1)}=\frac{1}{n+1}\left(ib_{i-1,j,k}+jb_{i,j-1,k}+kb_{i,j,k-1}\right)$$

finally substituting e1=(1,0,0), e2=(0,1,0), e3=(0,0,1)

$$b_i^{(1)}=\frac{1}{n+1}\left(ib_{\bar{i}-e1}+jb_{\bar{i}-e2}+kb_{\bar{i}-e3}\right)$$

and this is the form we desired.