Out[1]: (Show / Hide code)

Computing Bézier Surfaces

CSE 4303 / CSE 5365 Computer Graphics

2020 Fall Semester, Version 1.1, 2020 December 04

A large amount of derivation and theory of Bézier curves and surfaces is given in another handout. Here theory meets practice in that we concentrate on the rather prosaic need for actually computing numeric values for the points and then constructing triangles that can be drawn. We begin with a brief background statement and then jump right into the calculation. Since this is supposed to be expository, we do not do this calculation in the most efficient manner possible but emphasize instead comprehensibility.

In brief, we use the control points of the Bézier surface and the desired resolution to compute the points that will comprise the surface using the parametric description of such a surface. Since it's a surface, there are two independent variables, u and v, each ranging $[0, 1]$. The generated points are then grouped into a series of triangles that can be drawn.

Definition of a Bicubic Bézier Surface

As we have developed elsewhere, the points on a bicubic Bézier surface can be computed thus,

$$\mathbf{B}^{3,3}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3}\binom{3}{i}u^i(1-u)^{3-i}\binom{3}{j}v^j(1-v)^{3-j}\mathbf{P}_{i,j}$$

What this says is that each point u, v of the surface is a linear combination of the sixteen control points $\mathbf{P}_{i,j}$. We can write this more simply as,

$$\mathbf{B}^{3,3}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3}c_{i,j}(u,v)\mathbf{P}_{i,j}$$

where,

$$c_{i,j}(u,v) = \binom{3}{i}u^i(1-u)^{3-i}\binom{3}{j}v^j(1-v)^{3-j}$$

The $c_{i,j}(u,v)$ are the coefficients of the $\mathbf{P}_{i,j}$ in the linear combination.

By doing some algebra, these coefficients $c_{i,j}(u,v)$ can be expanded into closed forms,

Out[2]:

$$
\begin{aligned}
c_{0,0} &= (1-u)^3(1-v)^3 & c_{0,1} &= 3v(1-u)^3(1-v)^2 \\
c_{0,2} &= 3v^2(1-u)^3(1-v) & c_{0,3} &= v^3(1-u)^3 \\
\\
c_{1,0} &= 3u(1-u)^2(1-v)^3 & c_{1,1} &= 9uv(1-u)^2(1-v)^2 \\
c_{1,2} &= 9uv^2(1-u)^2(1-v) & c_{1,3} &= 3uv^3(1-u)^2 \\
\\
c_{2,0} &= 3u^2(1-u)(1-v)^3 & c_{2,1} &= 9u^2v(1-u)(1-v)^2 \\
c_{2,2} &= 9u^2v^2(1-u)(1-v) & c_{2,3} &= 3u^2v^3(1-u) \\
\\
c_{3,0} &= u^3(1-v)^3 & c_{3,1} &= 3u^3v(1-v)^2 \\
c_{3,2} &= 3u^3v^2(1-v) & c_{3,3} &= u^3v^3
\end{aligned}
$$

So for any given values of u and v, we use these sixteen formulae to obtain the values of the coefficients $c_{0,0}$ through $c_{3,3}$. Given those coefficient values, we compute the point at position (u, v) as a linear combination of the control points $\mathbf{P}_{i,j}$ as,

$$\mathbf{B}^{3,3}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} c_{i,j}\mathbf{P}_{i,j}$$

Easy!

Generating the Points of a Bézier Surface

To generate the points, we need two items,

- The sixteen control points, $\mathbf{P}_{i,j}, \quad i,j \in [0,3]$
- The desired resolution r.

Generally, we use the same resolution for both the u and v dimensions, though this is not absolutely required. For resolution r, we will have r separate values for each of the u and v ranges. For example, for resolution $r = 5$ we would have,

Out[3]:

$$u = \begin{bmatrix} 0.0 & 0.25 & 0.5 & 0.75 & 1.0 \end{bmatrix}$$

$$v = \begin{bmatrix} 0.0 & 0.25 & 0.5 & 0.75 & 1.0 \end{bmatrix}$$

For arbitrary resolution r, these values can easily be computed as $\frac{i}{r-1}$ for $0 \leq i < r$.

Putting all of this together, we can express the generation of the points of a bicubic Bézier surface with this pseudo-code,

```
pointList = array[ resolution*resolution ] of Point
index = 0

for m = 0 to resolution-1
    u = m/(resolution-1)

    for n = 0 to resolution-1
        v = n/(resolution-1)

        point = ( 0.0, 0.0, 0.0 )

        for i = 0 to 3
            for j = 0 to 3
                compute c[i,j]
                point = point + c[i,j] * P[i,j]
            end
        end

        pointList[index++] = point
    end
end
```

where P is the 4×4 set of control points for this Bézier surface.

> *Implementation note*: m and n are integer values, u and v are real values. Be careful when implementing to do the calculations for u and v as double-precision floating point. (The easiest way to ensure this is to subtract 1.0 from the resolution instead of 1. (Why does this work?))

> *Implementation note*: Often the control points are kept in a list rather than a two-dimensional array. In that case, the linear index of the control point (i, j) can be computed as $4i + j$. That index will then be in the range $[0, 15]$.

> *Implementation note*: This pseudo-code shows the computation of the linear combination by simply multiplying each control point by its coefficient and then adding the result to the running sum in `point`. This is usually accomplished by doing the calculations for each of the `point`'s x, y, and z coordinates. The w coordinate will always be 1 after the linear combination so it can be set to that value directly instead of being calculated.
>
> (Why is the value of the w coordinate of the linear combination result guaranteed to be 1?)

A bicubic Bézier surface of resolution r along each of the u and v dimensions will consist of r^2 separate points. For convenience we keep these points in a one-dimensional array `pointList`. As the points are generated in order, they are placed in successive locations in the array.

Converting the Points into Triangles

The generated points for the Bézier surface of resolution r are in a $r \times r$ square array. From these r^2 points we have to generate the triangular faces that we know how to draw. This isn't difficult if it's gone about in an orderly dashion.

Individual triangles are constructed by considering the points in order. We simply run a row counter from 0 to $r - 2$ and inside that loop a column counter from 0 to $r - 2$. (It's $r - 2$ because the rows and columns are numbered from 0 to $r - 1$. We stop one unit before that end because the last row / column is processed with the previous row / column.)

```
for row = 0 to resolution-2
    rowStart = row * resolution

    for col = 0 to resolution-2
      here  = rowStart + col
      there = here + resolution

      // Triangle A comprises the vertices here, there, and there+1
      // Triangle B comprises the vertices there+1, here+1, and here
    end
end
```

What we do with the vertex numbers depends on how the processing code is structured. Depending on the design, the triangles may be generated as faces, directly drawn, or used in some other fashion.

Consider the following diagram showing a Bézier patch at resolution $r = 5$. The individual vertices are labelled with their row, column numbers, ranging from `00` for row 0, column 0 to `04` for row 0, column 4 through to `44` for row 4, column 4 for a total of 25 vertices $= r^2$.

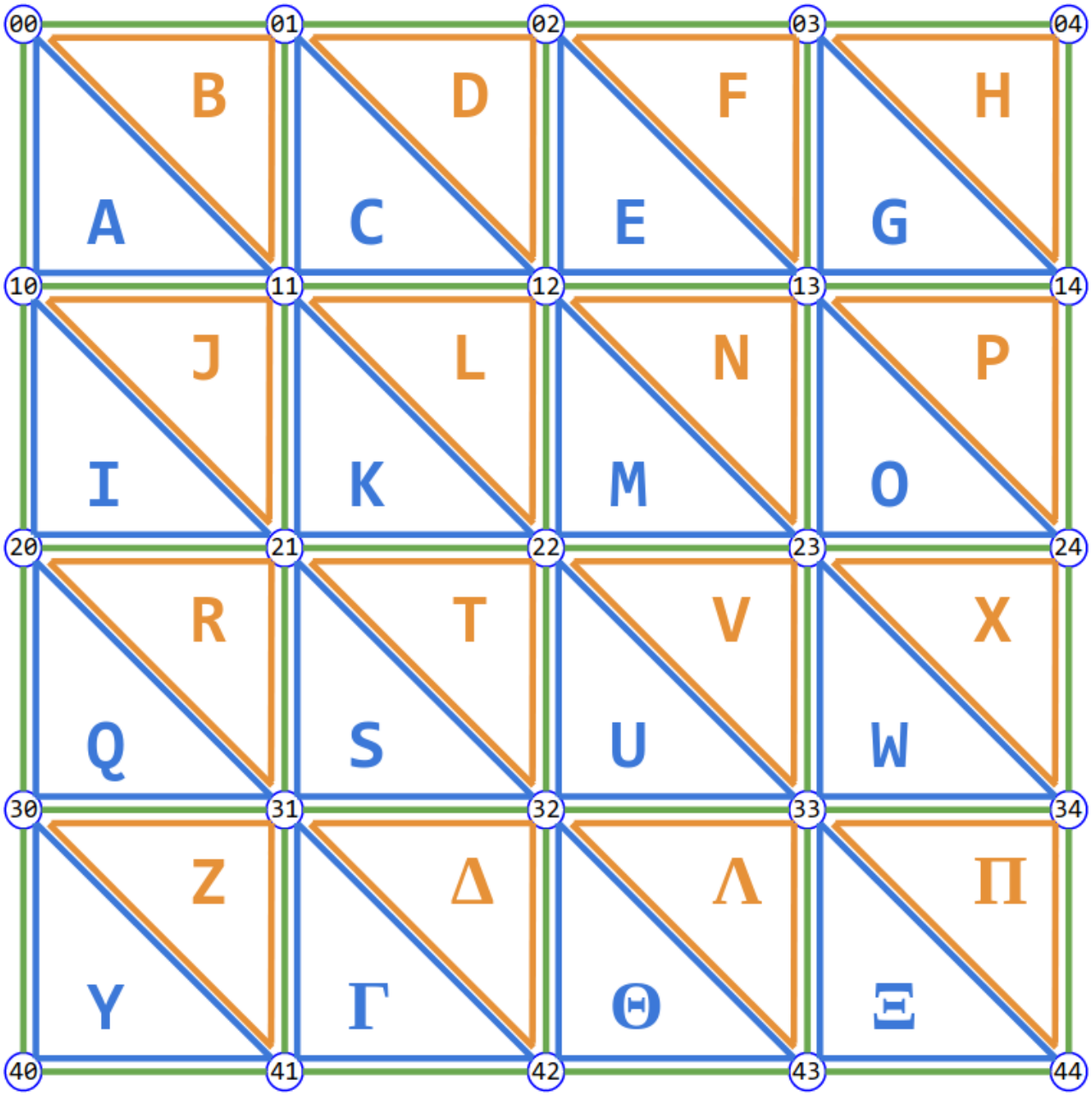
Since the vertices are kept in a linear list, the index of `00` is 0. The index of `10` is $0 + r = 5$. The index of `20` is $5 + r = 10$. The index of the vertex in the same column but one row ahead is always ahead by r units.

Tracing the pseudo-code given above, it's not difficult to see how the triangles are generated. `row` and `column` each run from 0 to $r - 2 = 3$. The index of the first vertex in each row of vertices (`rowStart`) is the row number times the resolution. `here` gets computed as `rowStart + col` each time through the inner loop. `here` sets the reference point for the two triangles that are generated. The first triangle is always the vertices `here` , `there` , and `there+1` . The second triangle is always the vertices `there+1` , `here+1` , and `here` .

For example, when `row` $= 0$ and `col` $= 0$, `rowStart` is 0, `here` is $0 =$ `rowStart` + `col` , and `there` is $5 =$ `row` + `resolution` . Therefore, $\triangle$ `A` will comprise vertices `here` $= 0$, `there` $= 5$, and `there+1` $= 6$. Counting the vertices, we see that these are vertices `00` , `10` , `11` , in that order. Similarly, $\triangle$ `B` will comprise vertices `there+1` $= 6$, `here+1` $= 1$, and `here` $= 0$. Counting out these index numbers gives us vertices `11` , `01` , `00` , in that order.

As a deeper example, when `row` $= 2$ and `col` $= 3$, `rowStart` is 10, `here` is 13, and `there` is 18. Therefore, $\triangle$ `W` will comprise vertices `here` $= 13$, `there` $= 18$, and `there+1` $= 19$. Counting the vertices, we see that these are vertices `23` , `33` , `34` , in that order. Similarly, $\triangle$ `X` will

comprise vertices `there+1` $= 19$, `here+1` $= 14$, and `here` $= 13$. Counting out these index numbers gives us vertices `34`, `24`, `23`, in that order.



Bézier Patch Triangles