Out[1]: (Show / Hide code)

Computing Bézier Surfaces

CSE 4303 / CSE 5365 Computer Graphics

Brian A. Dalio, 2019 Spring

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. 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.

Definition of a Bicubic Bézier Surface

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

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

Doing the algebra, we can compute the sixteen coefficients in terms of (u, v) as follows,

$$c_{i,j} = \binom{3}{i}u^i(1-u)^{3-i}\binom{3}{j}v^j(1-v)^{3-j}, \quad i,j \in [0,3]$$

Or, expanded,

Out[2]:

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

And, finally, the point at position (u, v) is,

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

Generating the Points of a Bézier Surface

To generate the points, we need two items,

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

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

Out[3]:

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

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

These values can easily be computed using the `numpy.linspace()` function. Each of u and v in the previous example were generated with the call,

```
numpy.linspace( 0.0, 1.0, 5 )
```

Remember, $u, v \in [0.0, 1.0]$.

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

```
pointList = []
for u in numpy.linspace( 0.0, 1.0, resolution )
    for v in numpy.linspace( 0.0, 1.0, resolution )
        point = ( 0.0, 0.0, 0.0 )
        for i in 0 .. 3
            for j in 0 .. 3
                compute c[i,j]
                point = point + c[i,j] * P[i,j]
            end
        end

        pointList.append( point )
    end
end
```

Once all of the points have been generated, individual triangles may be constructed by traversing the points in order,

```
for row in 0 .. resolution-2
    rowStart = row * resolution

    for col in 0 .. resolution-2
      here  = rowStart + col
      there = here + resolution

      triangleA = ( pointList[here], pointList[there], pointList[there+1] )
      triangleB = ( pointList[there+1], pointList[here+1], pointList[here] )

      drawTriangle( triangleA )
      drawTriangle( triangleB )
    end
end
```

The `drawTriangle()` routine can take care of transforming, projecting, and clipping the points and lines as it draws the triangle.