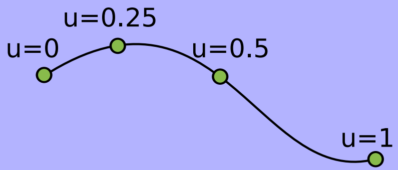
Tesselation

Review

- We’ve seen the basics of tessellation shaders
- Now we’ll see how they can contribute to better rendering
- Our motivating example for using tessellation: *spline surfaces*

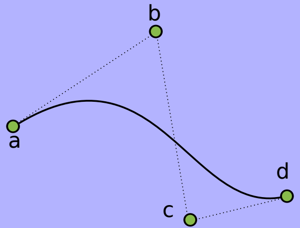
Bezier Splines

- Invented by P. Bezier at Renault in 1960's
- Define curve with a *parametric equation*
  - u=0 $\longrightarrow$ At start of curve
  - u=1 $\longrightarrow$ At end of curve



Specification

- *Control points* influence where curve goes
  - Example: Suppose we have four control points's: a,b,c,d
  - At start of curve, a has most influence, b a little less, c much less, and d almost none.
  - Near middle of curve, b and c influence the most; a and d somewhat less
  - Near end of the curve, d has most influence, a almost none; b a little, and c has more
  - Curve always tangent to first and last segment of control polygon



Example

- Spline explorer: spline.html

Terminology

- Three control points = quadratic bezier curve
- Four control points = cubic curve
- Five control points = quartic curve
- Six control points = quintic curve
- Seven control points = sextic curve
- Eight control points = septic curve
- Nine control points = octic curve
- Ten control points = nonic curve
- Eleven control points = decic curve
- 101 control points = hectic curve

Influence

- To find spline location: Let $0 \leq u \leq 1$: Point p on spline curve:

$$p(u) = \sum_{i=0}^{n} p_i B_{i,n}(u)$$

  - p_i= Control points, numbered 0...n
    - So: Total of n+1 control points
  - $B_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i}$
    - Called *Bernstein polynomials*
    - Define: All of these are 1: $\frac{0}{0}$, 0^0 , $0!$

Coefficients

- Cubic Bezier curves are commonly used; here, we have 4 control points
  - $B_{0,3}(u) = (1-u)^3$
  - $B_{1,3}(u) = 3u(1-u)^2$
  - $B_{2,3}(u) = 3u^2(1-u)$
  - $B_{3,3}(u) = u^3$
- So we get:
  $p(u) = (1-u)^3 p_0 + 3u(1-u)^2 p_1 + 3u^2(1-u) p_2 + u^3 p_3$

Note

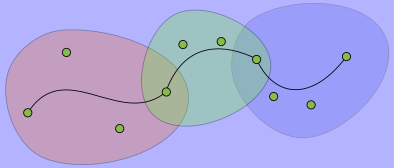
- We can use matrices to create a more compact notation
- Ex: For cubic curve:

$$p_x = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{0x} \\ p_{1x} \\ p_{2x} \\ p_{3x} \end{bmatrix}$$

- Repeat for y and z components
- You can verify this gives the same results as previous equation

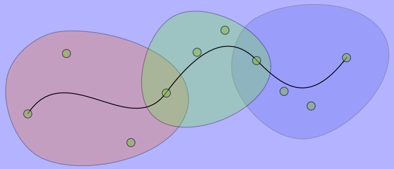
Joining Curves

- We can make longer curve by joining several smaller Bezier curves
- Ex: Join 3 cubic curves (4 control points each):
- Last point of curve i and first point of curve i+1 are coincident



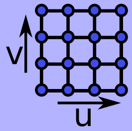
Joining Curves

- If next-to-last point of curve i and last point of curve i (= first point of curve i+1) and second point of curve i+1 are all collinear: Curve will look smooth (C1 continuity)
  - If end point of curve i is midpoint of segment: Looks smoother yet (C2 continuity)



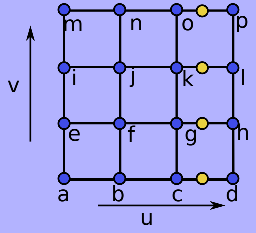
Surfaces

- Bicubic Bezier patch: A surface
  - Net of 4x4 control points
  - Two parameters: u and v: 0...1
- Formula: $p(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{n} p_{i,j}B_{i,n}(u)B_{j,n}(v)$



Surfaces

- Example: Suppose we want to find a point on the surface. Let u=0.7, v=0.3
  - Consider curve a,b,c,d. Find point on curve for u=0.7. Call this A.
  - Consider curve e,f,g,h. Find point on curve for u=0.7. Call this B.
  - Consider curve i,j,k,l. Find point on curve for u=0.7. Call this C.
  - Consider curve m,n,o,p. Find point on curve for u=0.7. Call this D.
  - Treat (A,B,C,D) as control points for a Bezier curve.
  - Compute point on that curve for v=0.3.



Alternate Way

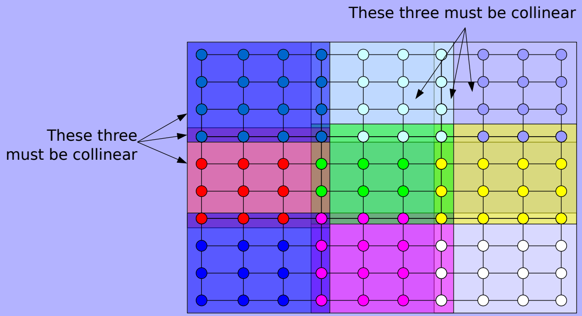
- We can also obtain via matrices:
  - Let M = $\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
  - Let P_x = 4x4 matrix of control points' x values, P_y = 4x4 matrix of control points' y values, P_z = matrix of control points' z values
  - Let $U = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$ and $V = \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$
  - Then: $p_x(u,v) = UMP_xM^TV$, $p_y(u,v) = UMP_yM^TV$, and $p_z(u,v) = UMP_zM^TV$

Normal

- To get the surface normal: We can take the cross product of the tangent and bitangent
- Tangent's x= $\frac{\partial p}{\partial u} = \left[\begin{array}{ccc} 3u^2 2u & 1 & 0 \end{array}\right] M P_x M^T V$
  - Similar for y and z: Use Py and Pz
- Bitangent's x = $\frac{\partial p}{\partial v} = U M P_x M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$
  - Similar for y and z
- Normal = tangent x bitangent

Joining

- Joining patches $\rightarrow$ Like joining curves
- Must have coincident shared control points
- Also have collinear points if we want smooth mesh



Rendering

- Tesselation shaders are tailor made for spline surfaces!
- Procedure:
  - Take 4x4 control mesh as input (i.e., set patch size to 16)
  - Tesselate according to distance from viewer
  - Draw it

Procedure

- Input: object space control points (16 of them: 4x4 grid)
- VS basically does nothing but copy data
- TCS runs 16 times (since 4x4 set of control points)
  - Sets amount of subdivision
    - Usually, we'd use distance from viewer to control this
  - Output a single control point
    - This happens 16 times so all 16 control points get sent out
- Tessellator generates quad patches
- TES runs for the generated points

TES

- In TES: How do we know where we are on the patch?
- This is what gl_TessCoord is for
- In quad mode: gl_TessCoord.x tells the u coordinate, gl_TessCoord.y tells the v coordinate
- We already have formula for determining Bezier spline point given control points, u, and v, so that's all we need
- We also compute the normal using the strategy just seen
- Fragment shader is the same as it's always been

Example

- Example: ex1.zip

Sources

- OpenGL Wiki. https://www.opengl.org/wiki/Tessellation_Control_Shader
- OpenGL Wiki. https://www.opengl.org/wiki/Tessellation_Evaluation_Shader
- OpenGL Wiki. https://www.opengl.org/wiki/Tessellation
- Philip Rideout. Triangle Tesselation with OpenGL 4.0. http://prideout.net/blog/?p=48
- Philip Rideout. Quad Tesselation with OpenGL 4.0. http://prideout.net/blog/?p=49
- http://onrendering.blogspot.com/2011/12/tessellation-on-gpu-curved-pn-triangles.html
- http://www.ludicon.com/castano/blog/2009/01/10-fun-things-to-do-with-tessellation/

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org