Objects 3: Surfaces

Christian Miller
CS 354 - Fall 2011
Bilinear interpolation

\[ p(u, v) = (1 - u)(1 - v)a + (u)(1 - v)b + (1 - u)(v)c + (u)(v)d \]

- Lerp along two opposite edges, then lerp between those results (requires 4 control points)
Bicubic surfaces

- You can do the same thing with cubic curves
- Requires $4^2 = 16$ control points
- Interpolate along 4 parallel rows, then interpolate between the results
**Geometric form**

\[ p^v(s, t) = S^T(s)M^T G^v M T(t) \]

\[
= \begin{bmatrix}
1 & s & s^2 & s^3
\end{bmatrix}
M^T \begin{bmatrix}
g^v_{11} & g^v_{12} & g^v_{13} & g^v_{14} \\
g^v_{21} & g^v_{22} & g^v_{23} & g^v_{24} \\
g^v_{31} & g^v_{32} & g^v_{33} & g^v_{34} \\
g^v_{41} & g^v_{42} & g^v_{43} & g^v_{44}
\end{bmatrix}
M \begin{bmatrix}
1 \\
t \\
t^2 \\
t^3
\end{bmatrix}
\]

- \( s, t \) are the 2D parameters (called \( u \) and \( v \) elsewhere)
- \( v \) stands for the x, y, or z coordinate
- \( M \) is the same basis matrix used in the cubic case
Bézier surfaces

- Just use the Bézier basis matrix for $M$
- Interpolates on the corners
Bézier control points
2D de Casteljau

- Do the same thing as in 1D case, using bilerp for each rectangle
JOINING BÉZIER PATCHES

- To get $C^0$ continuity, match all points along edges.
- To get $C^1$, linearly match second row of points on each side as well.
NURBS surfaces

- Extends NURBS to 2D surfaces
- An ubiquitous modeling primitive
- Flexible: can change number / placement of control points, parameter spacings, and weights
Rendering curves and surfaces

- Easy way: just step along $u$ and $v$ at some resolution, evaluate and spit out quads as you go.
- Better way: take into account visible size of surface and curvature of different regions, and adapt resolution accordingly.
  - Bézier curves can be recursively subdivided (see section 10.9.2 in your book).
• Everything so far has been rectangular, but it’s possible to do triangular patches as well
  • Gets a bit more complicated
• Splines and patch surfaces are great representations, but there are other ways to do curved surfaces...
Subdivision surfaces

• Subdivision surfaces: take an existing mesh, increase its resolution (refine), and guess how to add curvature to the new regions (smooth)

• Two flavors: approximating and interpolating
Chaikin’s subdivision

- The “corner cutting” scheme (approximating)
- Each iteration smooths the mesh
The 4-point scheme

- Doesn’t change existing points (interpolating)
- Each iteration adds detail, doesn’t smooth as much

\[
p_{2i}^{k+1} = p_i^k \quad p_{2i+1}^{k+1} = \left( \frac{1}{2} + w \right) (p_i^k + p_{i+1}^k) - w(p_{i-1}^k + p_{i+2}^k)
\]
4^n Surface refinement

- A very common way to refine triangle meshes
- Add new vertices halfway along each edge
- Creates 4x the number of triangles
Loop subdivision

- Use $4^N$ surface refinement
- Update positions of existing vertices
- New vertex locations are a weighted average of neighbors
Computing $\beta$

$$\beta(n) = \frac{1}{n} \left( \frac{5}{8} - \frac{(3 + 2 \cos(2\pi/n))^2}{64} \right)$$

$$\beta_{simp}(n) = \frac{3}{n(n + 2)}$$

- Here, $n = \#$ neighbors of vertex to be updated
- Two methods, full and simplified
Loop subdivision

• Approximating, since it alters original vertices
Modified butterfly subdivision

• Also uses $4^N$ refinement
• Interpolating, doesn’t change original vertices
• Much more complicated, has several cases
Modified butterfly

- Keeps original vertices, can cause distortion
Comparison

- Original / Loop / Modified butterfly
Comparison

- Original / Loop / Modified butterfly
Root-3 refinement

- Different refinement scheme, generates 3x the triangles on each iteration
- Add midpoint as new vertex, connect to corners, delete original edges, connect across deleted edge
Root-3 subdivision

- Use Root-3 refinement
- Approximating, similar to Loop

\[ \beta(n) = \frac{4 - 2 \cos\left(\frac{2\pi}{n}\right)}{9n} \]
Root-3 results

- Looks close to Loop, but 3/4 the triangles
Catmull-Clark Subdivision

- Works on non-triangle meshes (others don’t)
- Generates quads each iteration
- Generate new vertex at each face centroid, then new points from edges, then update old vertices
Equations can be found in papers, online, etc.
Catmull-Clark

- Works well on basically any mesh, loses only a small amount of volume
- Totally ubiquitous in modeling software
Another example
**Spline vs. Subdivision**

- There are reasons to use both
- Splines:
  - Mathematically elegant, provable continuity properties
  - Takes up very little space, infinite resolution, easy to edit
  - Can render adaptively, scale detail as needed
- Subdivision surfaces:
  - Can generate straight from a mesh, no other work necessary
  - Also have provable continuity properties (these proofs are hard!)
  - Generate a fixed increase in triangles each iteration
Implicit surfaces (a.k.a. isosurfaces)

\[ f(\mathbf{x}) = 0 \]

- Define a surface as the level set of some equation
  - Planes are a good example
- These can be arbitrarily complicated
- Very difficult to edit
- Makes some surfaces really easy to define, however
Metaballs

- Implicit surface based on distance from particles
Metaballs

- Implicit surface based on distance from particles
Metaballs
Metaballs
Rendering implicit surfaces

• HARD

• The equation itself doesn’t really tell you anything about where the surface is likely to be

• Could be all sorts of crazy topological conditions

• We’re reduced to sampling space and looking at the results
Marching Cubes

• Evaluate the function on a grid
• For each cube, look at the signs for the corners
• Use the pattern of those signs to look up the mesh for that cube in a lookup table
  • 256 cases in 3D, though only 14 if you account for rotations and symmetries
Marching cubes

- Result very dependent on grid sampling resolution
- Higher resolution means better results, but much more expensive to evaluate!
FIGURES COURTESY...

- Real-Time Rendering, 3rd ed. [RTR]
  - Tomas Akenine-Moller, Eric Haines, Naty Hoffman
  - Eric Lengyel
  - Edward Angel, Dave Shreiner
- Wikipedia [WP]
- http://paulbourke.net/geometry/polygonise/ [PB]