Today

- Bézier surfaces
- Advanced surface modeling
- Shader programming
- Environment mapping
- Bump mapping
- Toon shading

Bilinear patch

- Control mesh with four points \( p_0, p_1, p_2, p_3 \)
- Compute \( x(u, v) \) using a two-step construction

Bilinear patch (step 1)

- For a given value of \( u \), evaluate the linear curves on the two \( u \)-direction edges
- Use the same value \( u \) for both

\[
q_0 = \text{Lerp}(u, p_0, p_1) \\
q_1 = \text{Lerp}(u, p_2, p_3)
\]

Bilinear patch (step 2)

- Consider that \( q_0, q_1 \) define a line segment
- Evaluate it using \( v \) to get \( x \)

\[
x = \text{Lerp}(v, q_0, q_1)
\]

Bilinear patch

- Combining the steps, we get the full formula

\[
x(u, v) = \text{Lerp}(v, \text{Lerp}(u, p_0, p_1), \text{Lerp}(u, p_2, p_3))
\]
Bilinear patch

- Visualization

Bicubic Bézier patch

- Grid of 4x4 control points, \(p_0\) through \(p_{15}\)
- Four rows of control points define Bézier curves along \(u\)
  \(p_0p_1p_2p_3; p_4p_5p_6p_7; p_8p_9p_{10}p_{11}; p_{12}p_{13}p_{14}p_{15}\)
- Four columns define Bézier curves along \(v\)
  \(p_0p_4p_8p_{12}; p_1p_5p_9p_{13}; p_2p_6p_{10}p_{14}; p_3p_{15}p_{15}p_{15}\)

Bézier patch (step 1)

- Evaluate four \(u\)-direction Bézier curves at \(u\)
- Get points \(q_0, q_1\)

\[
q_0 = \text{Béz}(p_0, p_4, p_8, p_{12})
q_1 = \text{Béz}(p_1, p_5, p_9, p_{13})
q_2 = \text{Béz}(p_2, p_6, p_{10}, p_{14})
q_3 = \text{Béz}(p_3, p_7, p_{11}, p_{15})
\]

Bézier patch (step 2)

- Points \(q_0, q_1\) define a Bézier curve
- Evaluate it at \(v\)

\[
x(u, v) = \text{Béz}(v, q_0, q_1, q_2, q_3)
\]

Bézier patch

- Same result in either order (evaluate \(u\) before \(v\) or vice versa)

\[
q_0 = \text{Béz}(p_0, p_4, p_8, p_{12})
q_1 = \text{Béz}(p_1, p_5, p_9, p_{13})
q_2 = \text{Béz}(p_2, p_6, p_{10}, p_{14})
q_3 = \text{Béz}(p_3, p_7, p_{11}, p_{15})
\]

\[
x(u, v) = \text{Béz}(v, q_0, q_1, q_2, q_3)
\]

Tensor product formulation

- Corresponds to weighted average formulation
- Construct two-dimensional weighting function as product of two one-dimensional functions

\[
x(u, v) = \sum_i \sum_j p_{ij}B_i(u)B_j(v)
\]
- Bernstein polynomials \(B_i, B_j\) as for curves
**Properties**

- Convex hull: any point on the surface will fall within the convex hull of the control points
- Interpolates 4 corner points
- Approximates other 12 points, which act as “handles”
- The boundaries of the patch are the Bézier curves defined by the points on the mesh edges
- The parametric curves are all Bézier curves

**Tangents of Bézier patch**

- Remember parametric curves $x(u,v), y(u,v)$ where $u_0, v_0$ is fixed
- Tangents to surface = tangents to parametric curves
- Tangents are partial derivatives of $x(u,v)$
- Normal is cross product of the tangents

**Tessellating a Bézier patch**

- Uniform tessellation is most straightforward
  - Evaluate points on a grid of $u_i, v_i$ coordinates
  - Compute tangents at each point, take cross product to get per-vertex normal
  - Draw triangle strips (several choices of direction)
- Adaptive tessellation/recursive subdivision
  - Potential for “cracks” if patches on opposite sides of an edge divide differently
  - Tricky to get right, but can be done

**Piecewise Bézier surface**

- Lay out grid of adjacent meshes of control points
- For $C^0$ continuity, must share points on the edge
  - Each edge of a Bézier patch is a Bézier curve based only on the edge mesh points
  - So if adjacent meshes share edge points, the patches will line up exactly
- But we have a crease...

**C$^1$ continuity**

- We want the parametric curves that cross each edge to have C$^1$ continuity
  - So the handles must be equal-and-opposite across the edge:

http://www.spiritone.com/~english/cyclopedia/patches.html
Modeling with Bézier patches
• Original Utah teapot specified as Bézier Patches

Today
• Bézier surfaces
• Advanced surface modeling
  Shader programming
  • Environment mapping
  • Bump mapping
  • Toon shading

Advanced surface modeling
• B-spline/NURBS patches
• For the same reason as using B-spline/NURBS curves
  - More flexible (can model spheres)
  - Better mathematical properties, continuity

Advanced surface modeling
• Trim curves: cut away part of surface
  - Implement as part of tessellation/rendering

Modeling headaches
• Original Teapot isn’t “watertight”
  - Spout & handle intersect with body
  - No bottom
  - Hole in spout
  - Gap between lid and body

Modeling headaches
NURBS surfaces are flexible
  - Conic sections
  - Can blend, merge, trim…
  ...but
  • Any surface will be made of quadrilateral patches (quadrilateral topology)
**Quadrilateral topology**

Makes it hard to

- join or abut curved pieces
- build surfaces with awkward topology or structure

**Subdivision surfaces**

- Arbitrary mesh of control points, not quadrilateral topology
  - No global \( u,v \) parameters
  - Can make surfaces with arbitrary topology or connectivity
  - Work by recursively subdividing mesh faces
    - Per-vertex annotation for weights, corners, creases
  - Used in particular for character animation
    - One surface rather than collection of patches
    - Can deform geometry without creating cracks

**Questions?**

**Today**

- Bézier surfaces
- Advanced surface modeling

**Shader programming**

- Environment mapping
- Bump mapping
- Toon shading

**Environment maps**

- Represent illumination arriving at a point from all directions
- “360 degrees” panoramic image
- Instead of 360 degrees panoramic image, take picture of mirror ball (light probe)

**Environment maps applications**

- Use environment map as “light source”

[Light probes, Paul Debevec, http://www.debevec.org/Probes/]

[Global illumination, Sloan et al.]

[Reflection mapping]
Simplifying assumption

- Assume light captured by environment map is emitted from infinitely far away.
- Environment map is a directional light source.
  - Value of environment map is defined for each direction, independent of position.
- Use single environment map at each point in the scene.
- Approximation!

Cube environment maps

- Store incident light on six faces of a cube instead of on sphere.

Cube map look-up

- Given direction \((x, y, z)\)
- Largest coordinate component determines cube map face.
- Dividing by magnitude of largest component yields coordinates within face.
- In GLSL
  - Use \((x, y, z)\) direction as texture coordinates to `samplerCube`.

Reflection mapping

- Simulate mirror reflection.
- Compute reflection vector at each pixel.
- Use reflection vector to look up cube map.

Reflection mapping in GLSL

Application setup

- Load, bind a cube environment map.
  ```
  glBindTexture(GL_TEXTURE_CUBE_MAP, ...);
  glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X,...);
  glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X,...);
  glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y,...);
  ...
  glEnable(GL_TEXTURE_CUBE_MAP);
  ```

Vertex shader

- Compute viewing direction.
- Reflection direction
  - Use `reflect` function.
- Pass reflection direction to fragment shader.

Fragment shader

- Look-up cube map using interpolated reflection direction.
  ```
  varying float3 refl;
  uniform samplerCube envMap;
  textureCube(envMap, refl);
  ```
Reflection mapping examples

- Approximation, reflections are not accurate

Demo

- Cg c7 reflection

Today

- Bézier surfaces
- Advanced surface modeling
- Shader programming
- Environment mapping
- Bump mapping
- Toon shading

Bump mapping

- Surface detail is often the result of small perturbations in the surface geometry
- Modeling detailed surfaces would lead to impractical number of triangles
- Bump mapping alters the surface normal to provide the illusion of small scale surface detail
- Normals are encoded in texture maps

Bump mapping

- Generating and storing bump maps
- Rendering with bump maps
Generating bump maps

- Usually done in a **pre-process**
- **Input**
  - Texture map that encodes small surface displacements
- **Output**
  - Texture map that encodes normals of displaced surface
  - This texture will be stored as an image, read by the application

Storing bump maps

- Encode normal direction in RGB color channels
  - Coordinates of unit normal are in \([-1..1]^3\]
  - Need to map range \([-1..1]\) to \([0..255]\) for all channels

Rendering with bump maps

- Bump map normals are defined in **tangent space**
- Will define tangent space for each triangle
  - Texture coordinates provide parameterization of each triangle
  - Compute tangent space using partial derivatives of parameterization
  - Will need to transform normals from tangent space to camera space

Tangent space

- Triangle with texture coordinates can be expressed as parametric surface \(\mathbf{x}(u, v)\)
- Triangle vertices in object space \(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2\)

\[
\begin{align*}
\mathbf{v}_0 &= (u_0, v_0) \\
\mathbf{v}_1 &= (u_1, v_1) \\
\mathbf{v}_2 &= (u_2, v_2)
\end{align*}
\]

- We know

\[
\mathbf{x}(u_0, v_0) = \mathbf{v}_0, \quad \mathbf{x}(u_1, v_1) = \mathbf{v}_1, \quad \mathbf{x}(u_2, v_2) = \mathbf{v}_2
\]
**Tangent space**

- Solve for affine function
  \[ \mathbf{x}(u, v) = \begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \\ m_{2,0} & m_{2,1} & m_{2,2} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \]

- Using correspondences at vertices
  \[
  \begin{bmatrix}
  m_{0,0} & m_{0,1} & m_{0,2} \\
  m_{1,0} & m_{1,1} & m_{1,2} \\
  m_{2,0} & m_{2,1} & m_{2,2}
  \end{bmatrix}
  \begin{bmatrix}
  u_0 & u_1 & u_2 \\
  v_0 & v_1 & v_2 \\
  1 & 1 & 1
  \end{bmatrix}
  =
  \begin{bmatrix}
  v_0 & v_1 & v_2
  \end{bmatrix}
  \begin{bmatrix}
  m_{0,0} & m_{0,1} & m_{0,2} \\
  m_{1,0} & m_{1,1} & m_{1,2} \\
  m_{2,0} & m_{2,1} & m_{2,2}
  \end{bmatrix}
  \begin{bmatrix}
  u_0 & u_1 & u_2 \\
  v_0 & v_1 & v_2 \\
  1 & 1 & 1
  \end{bmatrix}^{-1}
  \]

**Normal in object space**

- Normal map stores normals in tangent coordinates
  - Basis vectors \( t, b, n \)
- Can transform normal from tangent to object space
  - Values \([bm_0, bm_1, bm_2]\) from bump map
  - Unpacked from \([0..1]\) to range \([-1..1]\)
  \[
  \mathbf{n}_{objectspace} = \begin{bmatrix}
  t & b & n
  \end{bmatrix}
  \begin{bmatrix}
  bm_0 \\
  bm_1 \\
  bm_2
  \end{bmatrix}
  \]

**Storing tangent vectors**

**Before rendering**

- For each triangle, compute tangent, bi-tangent vector
- At each vertex, average tangent, bi-tangent vectors over adjacent triangles
- Store tangent, bi-tangent vectors as additional vertex attributes

**Rendering**

**Vertex shader**

- Per-vertex input
  - Vertex position, tangent, bi-tangent vector in object space
  - Bump map texture coordinates
- Transform everything to camera space using modelview matrix
- Output to fragment shader
  - Vertex position, texture coordinates, tangent, bi-tangent, normal vector in camera space
  - Bump map texture coordinates

**Rendering with bump maps**

**Fragment shader**

- Transform normal \([bm_0, bm_1, bm_2]\) stored in bump map to camera coordinates
  - Use \( t, b, n \) basis to transform to object space
  - Use modelview matrix to transform from object space to camera space
  \[
  \mathbf{n}_{camera} = \begin{bmatrix}
  \text{modelview}
  \end{bmatrix}
  \begin{bmatrix}
  t & b & n
  \end{bmatrix}
  \begin{bmatrix}
  bm_0 \\
  bm_1 \\
  bm_2
  \end{bmatrix}
  \]
  - Normalize \( \mathbf{n}_{camera} \)
- Perform lighting in camera coordinates
Variations

• Perform lighting in different coordinate system than camera space
  - Object space
  - Tangent space
• Tangent space is more efficient
  - Transform light direction to tangent space in vertex shader
  - Interpolate across triangle
  - No need to transform bump mapped normal at each pixel

Caveats

• Avoid triangles with zero area in texture space
• Avoid triangles with negative area in texture space
  - May happen when texture is mirrored
• Avoid nonuniform stretching of bump map

Combination with texture map

• Demo FXComposer bumpGloss

Combination with env. map

• “Environment mapped bump mapping” (EMBM)
• Use bump mapped normal to compute reflection vector, look up cube map

Env. mapped bump mapping

• Use additional ‘dirt’ texture to modulate strength of reflection from environment map

Tutorials

• Caution, slightly different derivation
Today
• Bézier surfaces
• Advanced surface modeling
Shader programming
• Environment mapping
• Bump mapping
• Toon shading

Toon shading
• Simple cartoon style shader
• Emphasize silhouettes
• Silhouette edge detection
  - Compute dot product of viewing direction \( \mathbf{v} \) and normal \( \mathbf{n} \)
  \[
  \text{edge} = \max(0, \mathbf{n} \cdot \mathbf{v})
  \]
  - Use 1D texture to define edge ramp
    \[
    \text{uniform sample1D edgeramp; edge=texture1D(edgeramp,edge)};
    \]

Toon shading
• Compute diffuse and specular shading
  \[
  \text{diffuse} = \mathbf{n} \cdot \mathbf{L} \quad \text{specular} = (\mathbf{n} \cdot \mathbf{h})^\tau
  \]
• Use 1D textures diffuseramp, specularramp to map diffuse and specular shading to colors
• Final color
  \[
  \text{uniform sampler1D diffuseramp;}
  \text{uniform sampler1D specularramp;}
  \text{c=edge * (texture1D(diffuse,diffuseramp)+}
  \text{texture1D(specular,specularramp));}
  \]

More shaders
• NVidia shader library
  - Most shaders are in HLSL
• NVidia Cg toolkit
  - Predecessor of GLSL
  - Lots of example shaders
    [http://developer.nvidia.com/object/cg_toolkit_1_1.html](http://developer.nvidia.com/object/cg_toolkit_1_1.html)

Next time
• Shadows