Foundations of Computer Graphics (Spring 2010)

CS 184, Lecture 18: Shading & Texture Mapping
http://inst.eecs.berkeley.edu/~cs184

Many slides from Greg Humphreys, UVA and Rosalie Wolfe, DePaul tutorial teaching texture mapping visually.
Chapter 11 in text book covers some portions.

To Do

- Work on HW4 milestone
- Prepare for final push on HW 4, HW 5

Outline

- Brief discussion on Phong illumination model and Gouraud, Phong shading
- Texture Mapping (bulk of lecture)

These are key components of shading

Phong Illumination Model

- Specular or glossy materials: highlights
  - Polished floors, glossy paint, whiteboards
  - For plastics highlight is color of light source (not object)
  - For metals, highlight depends on surface color
- Really, (blurred) reflections of light source

Idea of Phong Illumination

- Find a simple way to create highlights that are view-dependent and happen at about the right place
- Not physically based
- Use dot product (cosine) of eye and reflection of light direction about surface normal
- Alternatively, dot product of half angle and normal
- Raise cosine lobe to some power to control sharpness or roughness

Phong Formula

\[ I = (R \cdot E)^\rho \]

\[ R = -L + 2(L \cdot N)N \]
Alternative: Half-Angle (Blinn-Phong)

\[ I - (N \cdot H)^p \]

- In practice, both diffuse and specular components for most materials

Flat vs. Gouraud Shading

- Flat: Determine that each face has a single normal, and color the entire face a single value, based on that normal.
- Gouraud: Determine the color at each vertex, using the normal at that vertex, and interpolate linearly for pixels between vertex locations.

Gouraud Shading – Details

- Gouraud Shading Details
- Scan line
- Actual implementation efficient: difference equations while scan converting

Gouraud and Errors

- \( I_1 = 0 \) because (N dot E) is negative.
- \( I_2 = 0 \) because (N dot L) is negative.
- Any interpolation of \( I_1 \) and \( I_2 \) will be 0.

2 Phongs make a Highlight

- Besides the Phong Reflectance model (cos^n), there is a Phong Shading model.
- Phong Shading: Instead of interpolating the intensities between vertices, interpolate the normals.
- The entire lighting calculation is performed for each pixel, based on the interpolated normal. (OpenGL doesn’t do this, but you can with current programmable shaders)

Problems with Interpolated Shading

- Silhouettes are still polygonal
- Interpolation in screen, not object space: perspective distortion
- Not rotation or orientation-independent
- How to compute vertex normals for sharply curving surfaces?
- But at end of day, polygons is mostly preferred to explicitly representing curved objects like spline patches for rendering
Outline

- Brief discussion on Phong illumination model and Gouraud, Phong shading
- Texture Mapping (bulk of lecture)

These are key components of shading

Texture Mapping

- Important topic: nearly all objects textured
  - Wood grain, faces, bricks and so on
  - Adds visual detail to scenes
- Meant as a fun and practically useful lecture

Adding Visual Detail

- Basic idea: use images instead of more polygons to represent fine scale color variation

Parameterization

- Q: How do we decide where on the geometry each color from the image should go?

Option: Varieties of projections

- [Paul Bourke]

Option: unfold the surface

- [Piponi2000]
### Option: make an atlas

- charts
- atlas
- surface

[Sander2001]

### Option: it’s the artist’s problem

- charts
- atlas
- surface

### Outline

- Types of projections
- Interpolating texture coordinates
- Broader use of textures

### How to map object to texture?

- To each vertex \((x, y, z)\) in object coordinates, must associate 2D texture coordinates \((s, t)\)
- So texture fits “nicely” over object

### Idea: Use Map Shape

- Map shapes correspond to various projections
  - Planar, Cylindrical, Spherical
- First, map (square) texture to basic map shape
- Then, map basic map shape to object
  - Or vice versa: Object to map shape, map shape to square
- Usually, this is straightforward
  - Maps from square to cylinder, plane, sphere well defined
  - Maps from object to these are simply spherical, cylindrical, cartesian coordinate systems

### Planar mapping

- Like projections, drop z coord \((s, t) = (x, y)\)
- Problems: what happens near \(z = 0\)?
Cylindrical Mapping
- Cylinder: \( r, \theta, z \) with \((s,t) = (\theta/(2\pi), z)\)
- Note seams when wrapping around \((\theta = 0 \text{ or } 2\pi)\)

Spherical Mapping
- Convert to spherical coordinates: use latitude/long.
- Singularities at north and south poles

Outline
- Types of projections
- Interpolating texture coordinates
- Broader use of textures

1st idea: Gouraud interp. of texcoords
\[
\begin{align*}
I_1 &= f(x_1-x) + f(x_1-y) \\
I_2 &= f(x_1-x) + f(x_2-y) \\
I_3 &= f(x_1-x) + f(x_1-y) \\
I_4 &= f(x_1-x) + f(x_2-y) \\
\end{align*}
\]
Actual implementation efficient: difference equations while scan converting
Artifacts

- McMillan’s demo of this is at http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide05.html
- Another example http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide06.html
- What artifacts do you see?
- Why?
- Why not in standard Gouraud shading?
- Hint: problem is in interpolating parameters

Interpolating Parameters

- The problem turns out to be fundamental to interpolating parameters in screen-space
  - Uniform steps in screen space = uniform steps in world space

Texture Mapping

- Linear interpolation of texture coordinates
- Correct interpolation with perspective divide

Perspective-Correct Interpolation

- Skipping a bit of math to make a long story short…
  - Rather than interpolating u and v directly, interpolate u/z and v/z
    - These do interpolate correctly in screen space
    - Also need to interpolate 1/z and multiply per-pixel
  - Problem: we don’t know 1/z anymore
  - Solution: we do know w = 1/z
  - So…interpolate uw and vw and w, and compute
    - \( u = uw/w \) and \( v = vw/w \) for each pixel
  - This unfortunately involves a divide per pixel

Texture Map Filtering

- Naive texture mapping aliases badly
- Look familiar?
  ```c
  int uval = (int) (u * denom + 0.5f);
  int vval = (int) (v * denom + 0.5f);
  int pix = texture.getPixel(uval, vval);
  ```
- Actually, each pixel maps to a region in texture
  - \(|\text{PIX}| < |\text{TEX}|\)
  - Easy: interpolate (bilinear) between texel values
  - \(|\text{PIX}| > |\text{TEX}|\)
    - Hard: average the contribution from multiple texels
  - \(|\text{PIX}| = |\text{TEX}|\)
    - Still need interpolation!
Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, linearly interpolate between two closest levels (e.g., trilinear filtering)
  - Fast, easy for hardware

- Why “Mip” maps?

MIP-map Example

- No filtering:
  - AAAAAGH MY EYES ARE BURNING
- MIP-map texturing:
  - Where are my glasses?

Outline

- Types of projections
- Interpolating texture coordinates
- Broader use of textures

Texture Mapping Applications

- Modulation, light maps
- Bump mapping
- Displacement mapping
- Illumination or Environment Mapping
- Procedural texturing
- And many more

Modulation textures

Map texture values to scale factor

Bump Mapping

- Texture = change in surface normal!
Displacement Mapping

- Quake introduced illumination maps or light maps to capture lighting effects in video games

Illumination Maps

- Texture map + light map:

Environment Maps

- Images from Illumination and Reflection Maps:
  - Simulated Objects in Simulated and Real Environments
  - Gene Miller and C. Robert Hoffman
  - SIGGRAPH 1984 "Advanced Computer Graphics Animation" Course Notes

Solid textures

- Texture values indexed by 3D location (x,y,z)
  - Expensive storage, or
  - Compute on the fly, e.g., Perlin noise

Procedural Texture Gallery