Shading 2: Textures

Christian Miller
CS 354 - Fall 2011
Priorities

• If you learn one thing from this class
  • Make it transforms
• If you learn two things from this class
  • Make the second one texture mapping
Texture mapping

- Take all your flat surfaces and stretch images over them
Texture mapping

- Allows you to take low-resolution geometry and make it look much more detailed
  - Does not actually increase resolution, polygon edges are still there
- Absolutely ubiquitous, crucial for realism
- Well-supported by graphics hardware
  - Specially designed memory units and layouts dedicated to textures
Texture mapping setup

- A mesh, a texture, and texture coordinates
For each vertex of the mesh, specify the 2D coordinates that it maps to in the texture.

Coordinates are usually \((u, v) \in [0,1]^2\)

- Sometimes called \((s, t)\) instead

Think of wrapping the image around the mesh.

Or of unwrapping the mesh onto the flat texture.
Texture coordinates
Texture coordinates

3-D Model

\[ p = (x, y, z) \]

UV Map

\[ p = (u, v) \]

Texture
• More complicated mappings are possible by splitting up the mesh into several regions
Rendering textures

- Interpolate texture coordinates from vertices (just like colors and normals)
- Use those coordinates to look up the right pixel in the texture (called a texel)
Perspective correct interpolation

- Remember, linear screen-space (affine) interpolation doesn’t cut it when perspective is used.
- See lecture 8, from Sept. 20
Generating texcoords

- The usual way is to have an artist manually unwrap the mesh and place the texcoords in a UV editor
  - This is tedious and time consuming
- Sometimes you can get away with generating them automatically
  - Often used as an initial guess, which the artist then edits to get the final result
Computing texcoords

- Spherical, cylindrical, planar, parametric
Projective textures

- Texcoords can have a \( w \) coordinate too
- Gives you textures that look like they were cast by a projector
Texture edge modes

- What happens when texcoords go outside $[0,1]$?
- Repeat, mirror, clamp, border
Aside: Map Projections

- One application of picking texture coordinates is figuring how to display a map of the world.
- This is Mercator, which is a variant of cylindrical projection.
Aside: Map Projections

- This is Buckminster Fuller’s Dymaxion projection
- Projects onto an icosahedron, then unwraps it
- Has all sorts of beautiful properties but is basically unintelligible
Automatic texcoords

• Automatically coming up with optimal texcoords for a given mesh is very, very hard

• Open research: mesh parameterization and texture atlases
Filtering

- Problem: what if the texture gets scaled or stretched?
- Simply picking the closest texel (nearest neighbor) looks hideous due to aliasing
The basic problem

- A pixel in screen space can be mapped to an arbitrary quad in texture space.
- Can cover less than one texel, or dozens of texels.
Magnification

- When pixel < texel, you can use bilerp
- Makes things look blurry instead of blocky
Bilerp doesn’t always look the best, especially if you want sharp edges
Minification

• When pixel > texel, you want to average over an area of texels to get the final value

• This is difficult, since the area you want to average over changes depending on how far away or distorted the texture is

• Dynamically figuring out the area and averaging is way too expensive
MIP-MAPPING

• Create a pyramid of different resolutions of a texture, each 1/4 the size of the last, down to 1x1

• This pyramid is created as a preprocess, where you can apply fancy filtering to make it look good

• Costs an additional 1/3 memory size

• At runtime, figure out the most appropriate mipmap level (usually based on Z) and sample that
Mipmap pyramid

$d$ axis
Mipmap results

- Looks much better, though has some overblurring
- Can be mitigated a bit with trilinear filtering
Anisotropic filtering

- Mipmaps look too blurry because they treat the projection of a pixel as a square in texture space.
- Anisotropic filtering treats it as a line segment, and takes a number of samples along it.
MIPMAP VS. ANISOTROPIC

- Mipmaps: require 1/3 extra memory, extremely cheap to compute, very cache coherent
- Anisotropic filtering: requires no extra memory, much more expensive, takes several unordered samples of a larger chunk of memory
3D textures

- 3D volumetric textures exist as well, though you can only render slices of them in OpenGL
- Generate a full image by stacking up slices in Z
- Used all the time for visualization
Aside: texture sizes

- Most of the time, graphics cards prefer textures to be square and powers of 2
- 512 x 512, 256 x 256, 32 x 32, etc.
- The largest textures that are often used at the moment are 2048 x 2048, but even those are 16 MB w/ mipmaps
- Texture compression is very important on graphics cards, and there are several standards in use
OpenGL texture notes

- Use `glEnable(GL_TEXTURE_2D)`
- GL has texture objects:
  - Generate with `glGenTextures()`
  - Choose current texture with `glBindTexture()`
  - Load an image with `glTexImage2D()`
  - Submit texcoords with `glTexCoordXD()`
- Several other functions to change options
Textures and lighting

- We’ve basically used textures for only diffuse lighting so far
- Clever use of texture coordinates and manipulation of lighting terms can generate a staggering variety of cool effects
- The Shading 4 (Oct 20) lecture is all about this
FIGURES COURTESY...

- Real-Time Rendering, 3rd ed. [RTR]
  - Tomas Akenine-Moller, Eric Haines, Naty Hoffman
  - Eric Lengyel
  - Edward Angel, Dave Shreiner
- Wikipedia [WP]
- Nvidia Projective Texture Mapping notes [PTM]