Computer Graphics

Prof. Feng Liu

Fall 2015

http://www.cs.pdx.edu/~fliu/courses/cs447/

11/17/2015
Last time

- Lighting and Shading
Today

- Texture Mapping
- Homework 4 available, due in class November 24
- Project 2
- Will publicize several times in the final week of classes when you can get your project graded
  - Demo your program to the instructor in person
    - Bring your own laptop or on a CS Windows Lab Machine
  - Latest time to grade
    - 5:00 pm, Friday, December 4, 2015
  - No late submission!
Mapping Techniques

- Consider the problem of rendering a globe
  - The geometry is very simple - a sphere
  - But the color changes rapidly, with sharp edges
  - With the local shading model, so far, the only place to specify color is at the vertices
  - To do a globe, would need thousands of polygons for a simple shape
  - Same thing for an orange: simple shape but complex normal vectors

- Solution: Mapping techniques use *simple geometry* modified by a *detail map* of some type
Globe and Orange
Texture Mapping

3D object + Texture = Textured object

Image source: https://upload.wikimedia.org/wikipedia/commons/0/04/UVMapping.png
Texture Mapping

- *Texture mapping* associates the color of a point with the color in an image: the *texture*

- Question to address
  - Which point of the texture do we use for a given point on the surface?

- Establish a *mapping* from surface points to image points
  - Different mappings are common for different shapes
  - We will, for now, just look at triangles (polygons)
Example Mappings
Basic Mapping

- The texture lives in a 2D space
  - Parameterize points in the texture with 2 coordinates: \((s,t)\)
  - These are just what we would call \((x,y)\) if we were talking about an image, but we wish to avoid confusion with the world \((x,y,z)\)

- Define the mapping from \((x,y,z)\) in world space to \((s,t)\) in texture space
  - To find the color in the texture, take an \((x,y,z)\) point on the surface, map it into texture space, and use it to look up the color of the texture
  - Samples in a texture are called **texels**, to distinguish them from pixels in the final image

- With polygons:
  - Specify \((s,t)\) coordinates at vertices
  - Interpolate \((s,t)\) for other points based on given vertices
Texture Interpolation

- Specify where the vertices in world space are mapped to in texture space
- A texture coordinate is the location in texture space that corresponds to the vertex
- Linearly interpolate the mapping for other points in world space
  - Straight lines in world space go to straight lines in texture space
Linear Interpolation

\((x_1, s_1)\) \(\rightarrow\) \((x, s)\) \(\rightarrow\) \((x_2, s_2)\)
Linear Interpolation

\[ s = s_1 \left( 1 - \frac{x-x_1}{x_2-x_1} \right) + s_2 \frac{x-x_1}{x_2-x_1} \]
Interpolating Coordinates

\[ s_L = \left( 1 - \frac{y - y_2}{y_3 - y_2} \right)s_2 + \left( \frac{y - y_2}{y_3 - y_2} \right)s_3 \]

\[ s_R = \left( 1 - \frac{y - y_1}{y_3 - y_1} \right)s_1 + \left( \frac{y - y_1}{y_3 - y_1} \right)s_3 \]

\[ s = \left( 1 - \frac{x - x_L}{x_R - x_L} \right)s_L + \left( \frac{x - x_L}{x_R - x_L} \right)s_R \]
Barycentric Coordinates

- An alternate way of describing points in triangles
- These can be used to interpolate texture coordinates
  - Gives the same result as previous slide
  - Method in textbook (Shirley)

\[ x = \alpha x_1 + \beta x_2 + \gamma x_3 \]

\[ \alpha = \frac{\text{Area}(x, x_2, x_3)}{\text{Area}(x_1, x_2, x_3)} \]

\[ \beta = \frac{\text{Area}(x_1, x, x_3)}{\text{Area}(x_1, x_2, x_3)} \]

\[ \delta = 1 - \alpha - \beta \]
Steps in Texture Mapping

- Polygons (triangles) are specified with texture coordinates at the vertices
  - A modeling step, but some ways to automate it for common shapes

- When rasterizing, interpolate the texture coordinates to get the texture coordinate at the current pixel
  - Previous slides

- Look up the texture map using those coordinates
  - Just round the texture coordinates to integers and index the image

- Take the color from the map and put it in the pixel
  - Many ways to put it into a pixel (more later)
Basic OpenGL Texturing

- Specify texture coordinates for the polygon:
  - Use `glTexCoord2f(s,t)` before each vertex:
    - Eg: `glTexCoord2f(0,0); glVertex3f(x,y,z);

- Create a texture object and fill it with texture data:
  - `glGenTextures(num,&identifier)` to get identifiers for the objects
  - `glBindTexture(GL_TEXTURE_2D, identifier)` to bind the texture
    - Following texture commands refer to the bound texture
  - `glTexParameteri(GL_TEXTURE_2D, ..., ...)` to specify parameters for use when applying the texture
  - `glTexImage2D(GL_TEXTURE_2D, ..., ...)` to specify the texture data (the image itself)

MORE...
Basic OpenGL Texturing (cont)

- Enable texturing: `glEnable(GL_TEXTURE_2D)`
- State how the texture will be used:
  - `glTexEnvf(...)`
- Texturing is done *after* lighting
- You’re ready to go...
Nasty Details

- There are a large range of functions that control the layout of texture data:
  - You must state how the data in your image is arranged
  - Eg: `glPixelStorei(GL_UNPACK_ALIGNMENT, 1)` tells OpenGL not to skip bytes at the end of a row
  - You must state how you want the texture to be put in memory: how many bits per “pixel”, which channels, ...

- Textures must be square with width/height a power of 2
  - Common sizes are 32x32, 64x64, 256x256
  - Smaller uses less memory, and there is a finite amount of texture memory on graphics cards
  - Some extensions to OpenGL allow arbitrary textures
Controlling Different Parameters

- The “pixels” in the texture map may be interpreted as many different things. For example:
  - As colors in RGB or RGBA format
  - As grayscale intensity
  - As alpha values only

- The data can be applied to the polygon in many different ways:
  - Replace: Replace the polygon color with the texture color
  - Modulate: Multiply the polygon color with the texture color or intensity
  - Similar to compositing: Composite texture with base color using operator
Example: Diffuse shading and texture

- Say you want to have an object textured and have the texture appear to be diffusely lit
- Problem: Texture is applied after lighting, so how do you adjust the texture’s brightness?
- Solution:
  - Make the polygon white and light it normally
  - Use `glTexEnvi(GL_TEXTURE_2D, GL_TEXTURE_ENV_MODE, GL_MODULATE)`
  - Use `GL_RGB` for internal format
  - Then, texture color is multiplied by surface color, and alpha is taken from fragment
Specular Color

- Typically, texture mapping happens *after* lighting
  - More useful in general
- Recall plastic surfaces and specularities: the highlight should be the color of the light
- But if texturing happens after the lighting, the color of the specularity will be modified by the texture - the wrong thing
- OpenGL lets you do the specular lighting after the texture
  - Use `glLightModel()`
We must reconstruct the texture image at the point \((s,t)\).

Time to apply the theory of sampling and reconstruction.
Textures and Aliasing

Textures are subject to aliasing:
- A polygon pixel maps into a texture image, essentially sampling the texture at a point
- The situation is essentially an image warp, with the warp defined by the mapping and projection

Standard approaches:
- Pre-filtering: Filter the texture down before applying it
  - Useful when the texture has multiple texels per output image pixel
- Post-filtering: Take multiple pixels from the texture and filter them before applying to the polygon fragment
  - Useful in all situations
Mipmapping (Pre-filtering)

- If a textured object is far away, one screen pixel (on an object) may map to many texture pixels
  - The problem is: how to combine them

- A mipmap is a low resolution version of a texture
  - Texture is filtered down as a pre-processing step:
    - `gluBuild2DMipmaps(...)`
  - When the textured object is far away, use the mipmap chosen so that one image pixel maps to at most four mipmap pixels
  - Full set of mipmaps requires at most 1.3333 the storage of the original texture (in the limit)
    - `1 + 0.25 + 0.25*0.25 + 0.25*0.25*0.25 + ...`
Mipmaps

For far objects

For middle objects

For near objects
Mipmap Math

- Define a scale factor, $\rho = \text{texels/pixel}$
  - A texel is a pixel from a texture
  - $\rho$ is actually the maximum from $x$ and $y$
  - The scale factor may vary over a polygon
  - It can be derived from the transformation matrices

- Define $\lambda = \log_2 \rho$

- $\lambda$ tells you which mipmap level to use
  - Level 0 is the original texture, level 1 is the next smallest texture, and so on
  - If $\lambda < 0$, then multiple pixels map to one texel: magnification
Post-Filtering

- You tell OpenGL what sort of post-filtering to do
- Magnification: When $\lambda < 0$ the image pixel is smaller than the texel:
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, type)`
  - Type is `GL_LINEAR` or `GL_NEAREST`
- Minification: When $\lambda > 0$ the image pixel is bigger than the texel:
  - `GL_TEX_MIN_FILTER`
  - Can choose to:
    - Take nearest point in base texture, `GL_NEAREST`
    - Linearly interpolate nearest 4 pixels in base texture, `GL_LINEAR`
    - Take the nearest mipmap and then take nearest or interpolate in that mipmap, `GL_NEAREST_MIPMAP_LINEAR`
    - Interpolate between the two nearest mipmaps using nearest or interpolated points from each, `GL_LINEAR_MIPMAP_LINEAR`
Filtering Example

NEAREST_MIPMAP_NEAREST:
level 0, texel (0,0)

LINEAR_MIPMAP_NEAREST:
texel 0, texel (0,0) * 0.51
+ level 1, texel (0,0) * 0.49

NEAREST_MIPMAP_LINEAR:
level 0, combination of
texels (0,0), (1,0), (1,1), (0,1)

LINEAR_MIPMAP_LINEAR:
Combination of level 0
and level 1, 4 texels
from each level, using 8
texels in all

s=0.12, t=0.1
ρ=1.4
λ=0.49
Other Texture Stuff

- Texture must be in fast memory - it is accessed for every pixel drawn
  - If you exceed it, performance will degrade horribly
  - Skilled artists can pack textures for different objects into one image

- Texture memory is typically limited, so a range of functions are available to manage it

- Sometimes you want to apply multiple textures to the same point: *Multitexturing* is now in most new hardware
Next Time

☐ Mesh and Modelling