CMSC427 fall 2017
Texture Mapping

Majority of slides credit to Dr. Zwicker
Today

- Basic shader for texture mapping
- Texture coordinate assignment
- Antialiasing
- Fancy textures

- Warning: side notes will include history of pyramids with Dr. Rosenfeld, revisiting bilinear interpolation,
Texture mapping

• Glue textures (images) onto surfaces

• Same triangles, much more interesting and detailed appearance

• Think of colors as reflectance coefficients
Texture mapping - quick

- Basic shading - material constant over objects

- Basic shading plus texture mapping - color varies over object

- How do?
Texture mapping in OpenGL

- Initializing and loading texture requires series of OpenGL API calls
  - `glPixelStorei`
  - `glGenTextures`
  - `glBindTexture`
  - `glTexImage2D`
  - etc...

- Look up details when you need them
- Learn from example code, GLTexture.java
- Documentation: [http://www.opengl.org/documentation/](http://www.opengl.org/documentation/)
Basic shaders for texturing

// Need to initialize texture using OpenGL API call

// Vertex shader
uniform mat4 modelview;
uniform mat4 projection;
in vec2 texcoords;
in vec4 position;
out frag_texcoords;

void main()
{
    gl_Position = projection * modelview * position; // predefined output
    frag_texcoords = texcoords; // pass texture coords. to fragment shader
}

// Fragment shader
uniform sampler2D tex; // “tex” is reference to texture, set by host
in frag_texcoords;
out frag_color;
void main()
{
    frag_color = texture(tex, frag_texcoords); // “texture” is a GLSL fnct.
}
Getting a texture sampler

```java
import com.jogamp.opengl.util.texture.*;

// Declare texture objects at class level
private int earthTexture;  // Index to OpenGL texture unit
private Texture joglEarthTexture;  // Actual texture data

// Read texture - as part of initialization
joglEarthTexture = loadTexture("earth.jpg");
earthTexture = joglEarthTexture.getTextureObject();

// Bind texture to GPU - as part of display
gl.glActiveTexture(GL_TEXTURE0);
gl.glUniform1i(gl.glGetUniformLocation(rendering_program, "tex"), 0);
gl.glBindTexture(GL_TEXTURE_2D, earthTexture);

// Utility function
public Texture loadTexture (String textureFileName) {
    Texture tex = null;
    try {
        tex = TextureIO.newTexture(new File(textureFileName), false);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return tex;
}
```
Adding texture coordinates

// From Gordon program 5.1 Texture mapping on a pyramid
// From setVertices, one time initialization operation
float[] pyramid_positions = { // Vertices
   -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 0.0f, 1.0f, 0.0f, //front
   1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.0f, //right
   1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.0f, //back
  -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 1.0f, 0.0f, //left
   -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 1.0f, //LF
   -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, //RR
};
float[] texture_coordinates = { // Range 0-1
   0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f, // Range 0-1
   0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f,
   0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f,
   0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f,
   0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f,
   1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f
};

// Create vertex Vertex Buffer Object
gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
FloatBuffer pyrBuf = Buffers.newDirectFloatBuffer(pyramid_positions);
gl.glBufferData(GL_ARRAY_BUFFER, pyrBuf.limit()*4, pyrBuf, GL_STATIC_DRAW);

// Create texture coordinate Vertex Buffer Object
gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
FloatBuffer texBuf = Buffers.newDirectFloatBuffer(texture_coordinates);
gl.glBufferData(GL_ARRAY_BUFFER, texBuf.limit()*4, texBuf, GL_STATIC_DRAW);
Using texture coordinates

// From Gordon program 5.1  Texture mapping on a pyramid
// From display, repeated for each frame

// Bind vertices
gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);	gl.glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);	gl.glEnableVertexAttribArray(0);

// Bind texture coordinates
gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);	gl.glVertexAttribPointer(1, 2, GL_FLOAT, false, 0, 0);	gl.glEnableVertexAttribArray(1);

// Activate texture 0
gl.glActiveTexture(GL_TEXTURE0);	gl.glBindTexture(GL_TEXTURE_2D, earthTexture);

// Active z-buffer
gl.glEnable(GL_DEPTH_TEST);	gl.glDepthFunc(GL_LEQUAL);

// Draw
gl.glDrawArrays(GL_TRIANGLES, 0, 18);
Side note: Java-Shader binding

// Option 1: using preset location index

// Option 1 in Java program
gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
gl.glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
gl.glEnableVertexAttribArray(0); // We're connecting to location 0

// Option 2: query shader program

// Option 2 in Java program
gl.glActiveTexture(GL_TEXTURE0);
gl.glUniform1i(gl.glGetUniformLocation(rendering_program, "tex"), 0); // Query!

#version 430
layout (location=0) in vec3 pos; // Option 1 in shader
layout (location=1) in vec2 texCoord;

out vec2 tc;

uniform mat4 mv_matrix;
uniform mat4 proj_matrix;

uniform sampler2D tex;

void main(void){
  gl_Position = proj_matrix * mv_matrix * vec4(pos,1.0);
  tc = texCoord;
}

// NOTE: MACS DON'T ALLOW OPTION 1 FOR TEXTURES, MUST QUERY
Today

- Basic shader for texture mapping
- Texture coordinate assignment
- Texture filtering
- Fancy textures
Texture coordinate assignment

• Surface parameterization
  - Mapping between 3D positions on surface and 2D texture coordinates
  - In practice, defined by texture coordinates of triangle vertices

• Various options to establish a parameterization

• Note: texture coordinates are often called (s,t) or equivalently (u,v)

• Can be in range [0,1]
  or [0, width], [0, height] of image
Parametric surfaces

http://en.wikipedia.org/wiki/Parametric_surface

- Surface position $x, y, z$ given by three functions
  
  $x = f_x(u, v)$  
  $y = f_y(u, v)$  
  $z = f_z(u, v)$

  of parameters $u, v$

- Very common in computer aided design (CAD)
- Use $(u, v)$ parameters as texture coordinates
- Later in class: Bézier surfaces
Cylinder

http://en.wikipedia.org/wiki/Parametric_surface

- Vertices \((x,y,z)\) given by three functions
  \[
  x = r \cos(2\pi u) \\
  y = hv \\
  z = r \sin(2\pi u)
  \]

- Texture coordinates given by \(u, v \in [0,1]\)
Sphere

- Mercator projection
As a function of vertex positions

• In general, may compute $u$ and $v$ using two functions of vertex positions $x, y, z$

$$u = f_u(x, y, z), \quad v = f_v(x, y, z)$$

• How to define $f_u, f_v$?
Linear functions

- Simplest form: linear function (transformation) of vertex $x, y, z$ coordinates
- For example, orthographic transformation

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$
Projective transformation

- Use perspective projection of $x, y, z$ coordinates

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad u = u'/w, v = v'/w$$

- Useful to achieve “fake” lighting effects
Spherical mapping

- Use, e.g., spherical coordinates for sphere
- Place object in sphere
- “shrink-wrap” sphere to object
  - Shoot ray from center of sphere through each vertex
  - Spherical coordinates of the ray are texture coordinates for vertex
Cylindrical mapping

• Similar as spherical mapping, but with cylinder

• Useful for faces
Skin mapping

- Techniques to unfold surface onto plane
  - Minimize “distortions”
  - Preserve area, angle
- Sophisticated math
- Functionality usually provided by 3D modeling tools (Maya, Blender, etc.)
Today

• Basic shader for texture mapping
• Texture coordinate assignment
• Antialiasing
• Fancy textures
What is going on here?
Aliasing

Sufficiently sampled, no aliasing

http://en.wikipedia.org/wiki/Aliasing
Aliasing

Sufficiently sampled, no aliasing

Insufficiently sampled, aliasing

High frequencies in the input appear as low frequencies in the sampled signal

http://en.wikipedia.org/wiki/Aliasing
Antialiasing: intuition

- Pixel may cover large area on triangle in camera space
Antialiasing: intuition

- Pixel may cover large area on triangle in camera space
- Corresponds to many texels in texture space
- Should compute “average” of texels over pixel area
Antialiasing: the math

- Pixels are samples, not little squares
  [http://alvyray.com/Memos/CG/Microsoft/6_pixel.pdf](http://alvyray.com/Memos/CG/Microsoft/6_pixel.pdf)

- Use **frequency analysis** to explain sampling artifacts
  - Fourier transforms

- If you are interested
  - Heckbert, “Fundamentals of texture mapping”
  - Glassner, “Principles of digital image synthesis”
Antialiasing

- Can be achieved by „averaging“ texels over pixel area

- Problems, disadvantages?
Antialiasing using mipmaps

• Averaging over texels during rendering is expensive
  - Many texels as objects get smaller
  - Large memory access and computation cost
• Precompute and store “averaged” (filtered) textures
    - MIP stands for “multum in parvo” (Williams 1983)
• Practical solution to aliasing problem
  - Fast and simple
  - Available in OpenGL, implemented in GPUs
  - Reasonable quality
Mipmaps

Before rendering

- Precompute and store several filtered versions of textures (mipmaps)

- Filtering performs “local averaging”
  - Simplest: box filter, uniform weighting in a square window; replace each pixel by average of pixels in its neighborhood

- Use higher quality filter to avoid aliasing

- Precompute several filtered textures with different sizes of filtering window
Mipmaps

Double the size of the filtering window from level to level!
Computing mipmaps

- Filtering implemented using convolution
  http://en.wikipedia.org/wiki/Convolution

  - Input function $f$, convolution kernel (filter) $g$
  - Continuous formulation
    $$ (f * g)(t) \eqdef \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) \, d\tau $$
  - Discrete formulation
    $$ (f * g)[n] \eqdef \sum_{m=-\infty}^{\infty} f[m] \cdot g[n - m] $$

- Two-dimensional convolution is a straightforward extension
Computing mipmaps

• Filtered textures are blurry
  - Reduce resolution by factor 2 successively without losing information

• Increases memory cost only by 1/3
  - \( \frac{1}{3} = \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \ldots \)

• Width, height of texture needs to be power of two
Example

- Resolutions 512x512, 256x256, 128x128, 64x64, 32x32

“multum in parvo”
Example

- 1 texel in level 4 is an average of $4^4 = 256$ texels in level 0

“multum in parvo”
Rendering with mipmaps

• Interpolate texture coordinate of each pixel as before

• Compute approximate size of pixel in texture space

• Look-up color in nearest mipmap
  - E.g., if pixel corresponds to 10x10 texels use mip-map level 3
  - Use nearest neighbor or bilinear interpolation as before
Mipmapping

- Image plane
- Camera space
- Texture space

- Mip-map level 0
- Mip-map level 1
- Mip-map level 2
- Mip-map level 3

- Pixel area

- Texels
Size of a pixel in texture space

- Given by partial derivatives of mapping $u(x, y), v(x, y)$

Image plane

Camera space

Texture space

$\begin{pmatrix} \frac{\partial u}{\partial x'} & \frac{\partial v}{\partial x'} \\ \frac{\partial u}{\partial y'} & \frac{\partial v}{\partial y'} \end{pmatrix}$
Nearest mipmap, nearest neighbor

- Visible transition between mipmap levels
Nearest mipmap, bilinear

- Visible transition between mipmap levels
Trilinear mipmapping

http://en.wikipedia.org/wiki/Trilinear_filtering

- Use two nearest mipmap levels
  - E.g., if pixel corresponds to 10x10 texels, use mipmap level 3 and 4
- Perform bilinear interpolation in both mipmaps
- Linearly blend between the results
- Requires access to 8 texels for each pixel
- Standard method, supported by hardware with no performance penalty
Trilinear mipmapping

- Smooth transition between mipmap levels
Note on OpenGL

• Distinguishes between minification and magnification
  - Minification: a texel is smaller than a pixel
  - Magnification: a texel is larger than a pixel
  - Minification, magnification may vary across pixels of individual triangles

• OpenGL allows you to specify different interpolation techniques separately
  – glTexParameter
Are we satisfied?

Trilinear mipmapping
Mipmapping limitations

- Mipmap texels always represent square areas
- Pixel area is not always square in texture space
- Mipmapping makes trade-off between aliasing and blurriness

A circular pixel is back-projected to an ellipse
Anisotropic texture filtering

https://en.wikipedia.org/wiki/Anisotropic_filtering

- Average texture over elliptical area
  - Higher quality than trilinear mip-mapping
  - More expensive

- Anisotropic filtering in hardware
  - Take several bilinear probes approximating the ellipse
  - Reduces rendering performance on current GPUs
Comparison

- Animation
- OpenGL 3D Game Tutorial 41: Antialiasing and Anisotropic Filtering
- First 3 minutes
- https://www.youtube.com/watch?v=Pdn13TRWEM0
Today

- Basic shader for texture mapping
- Texture coordinate assignment
- Antialiasing
- Fancy textures
Fancy textures

- Textures most commonly used to modulate ambient and diffuse reflection

- E.g., diffuse fragment shader with texture

```c
in vec3 normal, lightstrength, lightDir;
uniform sampler2D tex;
out fragColor;

void main()
{
    fragColor = lightstrength * 
    max(dot(normal, normalize(lightDir)),0.0) * 
    texture(tex, texcoords); // texture as diffuse coeff.
}
```

- Other applications?
Bump mapping

- Texture map contains normal perturbations
- No modification of geometry
  - Visible mostly at silhouettes
- Render using per-pixel shading, fragment shader
  - Normal in each pixel is modified using texture map (later in course)
Displacement mapping

- Texture map contains local height field
- Modifies geometry
  - Correct silhouettes, shadows
- Requires complicated fragment shader
Other effects

Multi-texturing

• Several layers of textures for different effects
  - Scratches, dents, rust, ...
  - Illumination textures

Animated textures

• Raindrops

• A TV screen, projector in a 3D scene