Textures II

Logistics

- Programming Assignment 1 – Line Drawing
- Programming Assignment 2 – Polygon Fill
- Programming Assignment 3 – Clipping
- Programming Assignment 4 – Hello OpenGL
- Programming Assignment 5 - Tesselation
  - All done including resubmits
- Programming Assignment 6 – Transformations / Viewing
  - Resubmits due by 2/15
- Programming Assignment 7 – Shading
  - Due Thursday (tioday)

Logistics

- Grad report
  - List of papers due Friday, Jan 11th
    - All done
  - Actual Report due February 15th

Plan for this week

- Texture Mapping
- Today
  - Concepts
  - About the midterm
- Thursday
  - Doing it in OpenGL
  - + Final exam.
- But first..

Final Exam

- Like with the midterm:
  - Option of programming or written exam
- Written exam:
  - Monday, Feb 18th @ 12:30 – 2:30 (This room 70-3560)
  - Format of questions much like midterm
- Programming:
  - To be discussed in 2nd half.

Plan for today

- Doing textures in shader based OpenGL
- Programming Assignment 8 (and final)

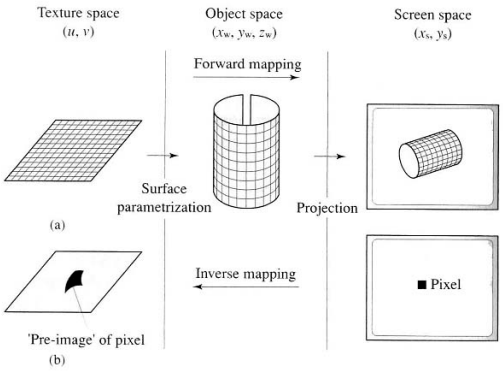
Texture Mapping

- Developed in 1974 by Ed Catmull
  - Currently, president of Pixar (head of Disney Animation)
- Goal: to make Phong shading less plastic-looking
- Uses of texturing
  - Simulating materials
  - Reducing geometric complexity
  - Image warping

Texture Mapping

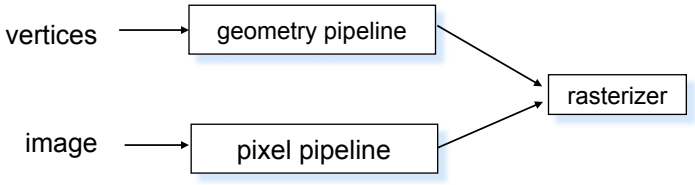
- Two basic types of textures:
  - Image files
  - Mathematically-generated (e.g., checkerboard pattern)
- Textures made up of *texture elements* (texels)
- Can be 1D, 2D, or 3D
  - 1D textures are images with dimension 1xN or Nx1
  - 2D textures have width and height
  - 3D textures describe volumes
    - Used, e.g., in CT/MRI scans
    - Huge – a 256x256x256 grayscale-alpha texture uses 32MB of memory
- For simplicity, we'll mostly look at 2D textures
  - Same principles apply to 1D and 3D

Texture Mapping



Texture Mapping and the OpenGL Pipeline

- Images and geometry flow through separate pipelines that join at the rasterizer
- Advantage: visual detail is in the image, not the geometry
  - "Complex" textures do not affect geometric complexity



Applying Textures I

- Four steps to applying a texture
  1. specify the texture
     - read or generate image
     - assign to texture
  2. specify texture parameters
     wrapping, filtering
  3. assign texture coordinates to vertices
  4. Bind texture to shader and use in shaders

Texture Objects

- Have OpenGL store your images
  - one image per texture object
  - may be shared by several graphics contexts
- Generate texture names

```
glGenTextures( n, *texIds );
```

Texture Objects (cont'd.)

- Bind textures before using

```
glActiveTexture (0);  // more on this later
glBindTexture( target, id );

target =  GL_TEXTURE_1D,
          GL_TEXTURE_2D,
          GL_TEXTURE_3D, or
          GL_TEXTURE_CUBE_MAP.
```

Specifying a Texture Image

- Define a texture image from an array of texels in CPU memory

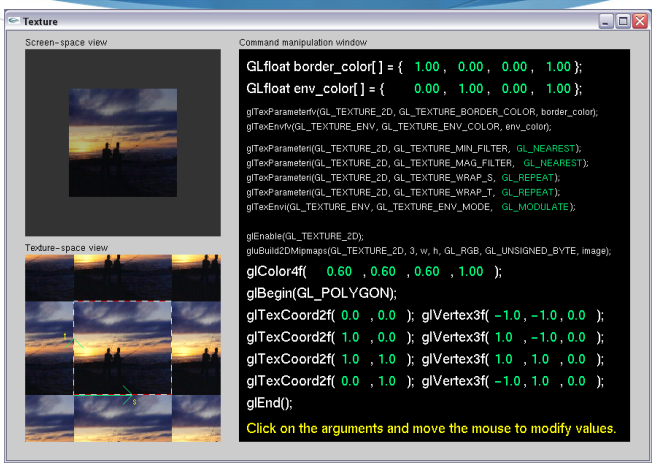
```
glTexImage2D( target, level, components,
    w, h, border, format, type, *texels );
```

- Texel colors are processed by pixel pipeline
  - pixel scales, biases and lookups can be done

Specifying a Texture Image

```
glTexImage2D( target,   // GL_TEXTURE_2D, etc.
              level,    // LOD (mip-map), usually 0
              components, // number of elems per color
              w, h,     // width and height…power of 2?
              border,   // 0 or 1 border or no border
              format,   // e.g. GL_RGB
              type,     // data type of data
              *texels );  // array of data
```

Texture Parameters



Assigning Texture parameters

- Things like wrapping, filtering, etc.

```
glTexParameter[fi](
    target, // GL_TEXTURE_2D, etc.
    paramname, // param being set
    param) // parameter value
```

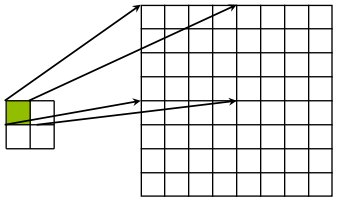
Texture parameters

- Filter Modes (glTexParameter())
  - Minification or magnification
  - Special *mipmap* minification filters
- Wrap Modes (glTexParameter())
  - Clamping or repeating

Filter Modes

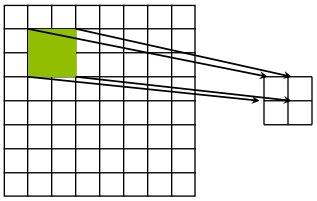
Example:

```
glTexParameteri( target, type, mode );
```



Texture Polygon

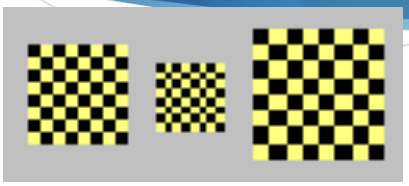
Magnification



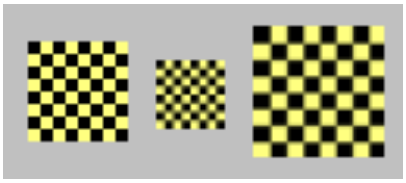
Texture Polygon

Minification

Filtering



Using GL_NEAREST for both the GL_TEXTURE_MAG_FILTER and the GL_TEXTURE_MIN_FILTER. First, the original 32x32 texture; then, the texture scaled to 66%; then, the texture scaled to 133%. Notice how the textures appear "blocky", especially in the smaller texture.



Using GL_LINEAR for both the GL_TEXTURE_MAG_FILTER and the GL_TEXTURE_MIN_FILTER. Notice how the textures appear much smoother thanks to the linear interpolation.

Texture wrapping Modes

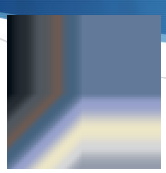
- Texture parameter options for a currently-bound texture:
  - **GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T**
    - What should be done if the horizontal (***_S**) or vertical (***_T**) texture coordinates ever go beyond the [0,1] range
    - **GL_CLAMP** clamps the texture
    - **GL_REPEAT** tiles it
  - **GL_TEXTURE_MAG_FILTER**, **GL_TEXTURE_MIN_FILTER**
    - What kind of stretching should be done if a texture-mapped polygon is rendered larger (***_MAG_***) or smaller (***_MIN_***) than the texture mapped to it is (pixel-wise)
    - **GL_NEAREST** uses simple "blocky" stretching for textures
    - **GL_LINEAR** uses linear-interpolated stretching

Wrapping Mode

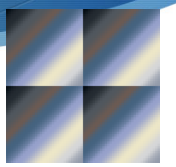
- Example:

```
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_REPEAT )
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_CLAMP )
```

Wrapping Mode

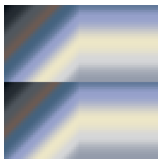


The bound texture

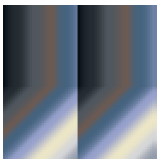


```
GL_TEXTURE_WRAP_S = GL_CLAMP
GL_TEXTURE_WRAP_T = GL_CLAMP
```

```
GL_TEXTURE_WRAP_S = GL_REPEAT
GL_TEXTURE_WRAP_T = GL_REPEAT
```



```
GL_TEXTURE_WRAP_S = GL_CLAMP
GL_TEXTURE_WRAP_T = GL_REPEAT
```



```
GL_TEXTURE_WRAP_S = GL_REPEAT
GL_TEXTURE_WRAP_T = GL_CLAMP
```

Reading image data

- OpenGL does not natively support reading of any image file format.
- Get the data in an array…doesn't care how you get it there.
- However…

Simple OpenGL Image Library (SOIL)

- Easy to use library that not only will read image files but will also set up OpenGL textures.
- http://www.lonesock.net/soil.html
- Supports multiple image formats including:
  - BMP, PNG, JPG, TGA, PSD, HDR
- Can also automatically create MIP Maps for you.
- But only for C++.

Simple OpenGL Image Library (SOIL)

- Example call:

```
/* load an image file directly as a new OpenGL texture */
GLuint tex_2d = SOIL_load_OGL_texture
      ( "img.png",
         SOIL_LOAD_AUTO,
         SOIL_CREATE_NEW_ID,
         SOIL_FLAG_MIPMAPS | SOIL_FLAG_TEXTURE_REPEATS);

/* check for an error during the load process */
 if ( 0 == tex_2d ) {
       printf( "SOIL loading error: '%s'\n",
       SOIL_last_result() );
}
```

Textures in Java

- JOGL `TextureIO` class
  - Set of convenience routines for reading and setting up openGL textures

https://jogamp.org/deployment/jogamp-next/javadoc/jogl/javadoc/com/jogamp/opengl/util/texture/TextureIO.html

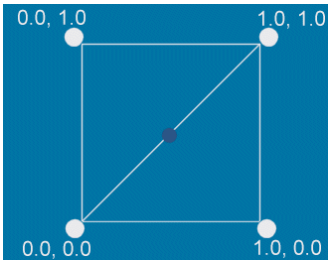
Textures in Java

- Can set texture parameters on returned `Texture` object

```
Texture tex_id;
try {
  InputStream stream = getClass().getResourceAsStream("texture.jpg");
  TextureData data = TextureIO.newTextureData(GLProfile.getDefault(),
                        stream, false, "jpg");
  tex_id = TextureIO.newTexture(data);
 }
 catch (IOException exc) {
      // Do error handling here
 }
```

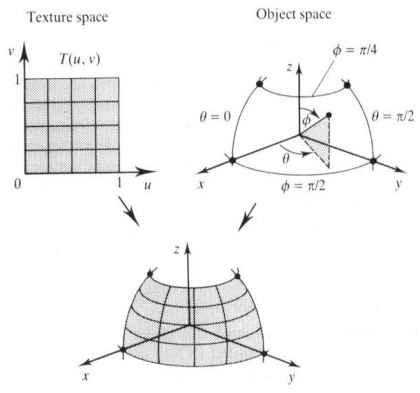
Texture Coordinates

- Stretching and shrinking of the texture is based on the premise of sampling
- Texture coordinates become necessary
  - Arbitrary mapping of texture coordinates to polygon coordinates

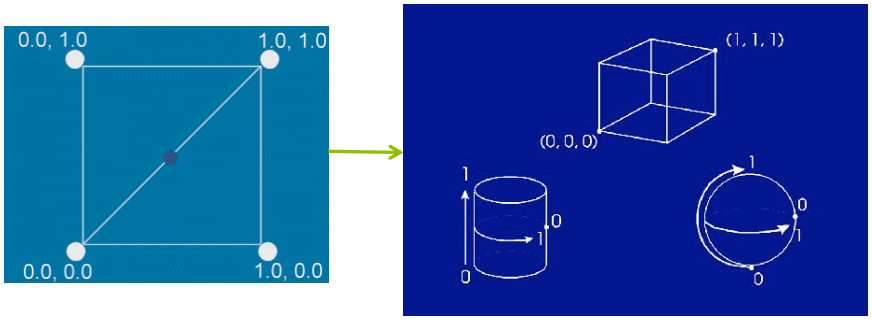


Texture Coords

- Texture is like a rubber sheet stretched to fit model



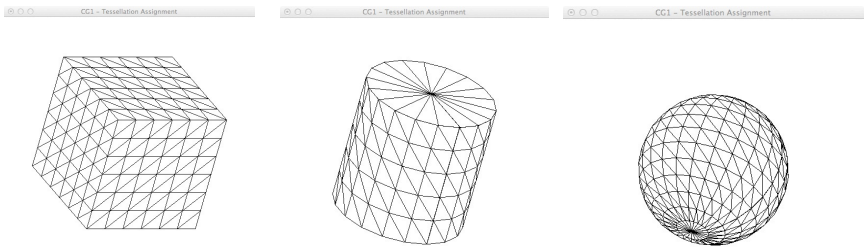
Types of Mapping



Assign Texture Coords

- Texture coords are **attributes** attached to vertices by the application.
- Would like texture coords to be attached to vertices when sent to fragment shader so they can be interpolated during rasterization.
- Interpolated texture coord will provide the (u,v) for a given pixel

Texturing and Tessellation



Using in Shaders

- `sampler` datatype
  - Used to represent a texture somewhere in GPU texture memory.
  - Wide variety of sampler types:
    - `sampler1D, sampler2D, sampler3D`
    - `samplerCube, sampler2Drect`
    - `sampler1DShadow, sampler2DShadow`
  - See http://www.opengl.org/wiki/GLSL_Sampler

Texture access in shader

- Basic access performed by `texture` function
- You're free to do whatever you like with the returned value
- Will do antialiasing as defined when setting up texture

```
vec4 texture(sampler2D sampler, vec2 texCoord[, float bias]);
Or
vec4 texture2D (sampler2D sampler, vec2 texCoord[, float bias]);
```

Vertex Shader

```
attribute vec4 vPosition;
attribute vec2 vTexCoord;

varying vec2 texCoord;

void main()
{
    texCoord    = vTexCoord;
    gl_Position = vPosition;
}
```

Fragment Shader

```
varying vec2 texCoord;

uniform sampler2D texture;

void main()
{
  gl_FragColor = texture( texture,
                          texCoord );
}
```

Binding Textures to Samples

- OpenGL maintains a number of independent "bind locations" for textures.
  - Actual number will depend on the graphics card
  - Named GL_TEXTURE0, GL_TEXTURE1, etc.
  - Use the `glActiveTexture()` function to make a bind location active.
  - To bind a texture to a sampler, you send the texture bind location unit as a value for the uniform sampler value.

Binding Textures to samplers

```
glUseProgram(program);
GLint baseImageLoc = glGetUniformLocation(program, "baseImage");
GLint normalMapLoc = glGetUniformLocation(program, "normalMap");
GLint shadowMapLoc = glGetUniformLocation(program, "shadowMap");

glUniformi(baseImageLoc, 0); //Texture unit 0
glUniformi(normalMapLoc, 2); //Texture unit 2
glUniformi(shadowMapLoc, 4); //Texture unit 4

//When rendering an objectwith this program.
glActiveTexture(GL_TEXTURE0 + 0);
glBindTexture(GL_TEXTURE_2D, object1BaseImage);
glActiveTexture(GL_TEXTURE0 + 2);
glBindTexture(GL_TEXTURE_2D, object1NormalMap);
glActiveTexture(GL_TEXTURE0 + 4);
glBindTexture(GL_TEXTURE_2D, shadowMap);
glUseProgram(program);
```

Binding Textures to samplers

- Note that in Java texture activation and binding is done by the JOGL `Texture` object
  - `public void enable(GL gl) throws GLException`
  - `public void bind(GL gl) throws GLException`

Summary

- In application:
  - Define / read texture data
  - Assign texture coords
  - Define texture parameters
  - Bind buffer to target
  - Load texture data into buffer
  - Bind texture to shader "sampler"
- In shader:
  - Use samplers to get texture access.

Questions?

Assignment 8

- Goal:
  - To gain experience using textures in shader based OpenGL,
- Draw a single object using a texture map for color data
- Type of mapping depends upon type of object
  - Cube = planar
  - Cylinder = cylindrical
  - Sphere = spherical

Assignment 8

- Tasks:
  - Modify tessellation routines to generate texture coordinates.
  - Read/setup texture in application
  - Access texture values in shader and apply.

Assignment 8

- Due next Thursday, Feb 14th @11:159pm
- If you are still in need of tessellation routines, please see
  - http://www.crackinwise.com/3d-tessellation-algorithms-in-c/

Break