Texture Mapping

Woo, Neider et Al., Chapter 9
Texture Mapping in OpenGL

- Allows you to modify the color of a polygon surface
- Textures are simply rectangular arrays of data (color, luminance, color+alpha). Individual values in a texture are called texels
Texture Mapping Modalities

• You can repeat a texture (in one or both directions) to cover a surface

• A texture can be applied to a surface in different ways:
  1) Painted directly (like a decal)
  2) Used to modulate the original surface-color
  3) Use to blend with the original surface-color
Using Texture Mapping

Steps necessary to use texture mapping:

- Create a texture object and specify the texture
- Indicate how the texture is to be applied to each pixel
- Enable texture mapping
- Draw the scene, supplying both texture and geometric coordinates
Texture Specification

• A texture is usually 2D
• It’s data can consist of 1, 2, 3, 4 elements per texel
• A method called **mipmapping** can be used to generate textures at different resolutions and increase performance
• Mipmapping prevents unnecessary and expensive mappings to small polygons
2D Texture Specification

```c
glTexImage2D(GLenum target, GLint level,
GLint internalFormat, GLsizei width, GLsizei height,
Glint border, GLenum format, GLenum type,
const Glvoid *pixels);
```

- **target**: GL_TEXTURE_2D, GL_PROXY_TEXTURE_2D
- **level**: specifies the level of detail when using multi resolution textures. “0” is the base image, “n” is the n-th mipmap reduction image
- **internalFormat**: an integer 1 to 4, or one of 38 symbolic constants
- **width, height**: the dimensions of the texture (MUST BE power of 2)
- **format**: the kind of pixel-data elements
- **type**: the data-type of each element
- **pixels**: array containing the texture image data
Values for Format and Type

• Format Constants:
  - GL_COLOR_INDEX  A single color index
  - GL_RGB          A red component, followed by green & blue components
  - GL_RGBA         Like GL_RGB, followed by an alpha component.
  - GL_RED          A single red-color component
  - GL_GREEN        A single green-color component
  - GL_BLUE         A single blue-color component
  - GL_ALPHA        A single alpha-color component

• Type Constants:
  - GL_UNSIGNED_BYTE unsigned 8-bit integer
  - GL_BYTE         signed 8-bit integer
  - GL_UNSIGNED_SHORT unsigned 16-bit integer
  - GL_SHORT        signed 16-bit integer
  - GL_INT          signed 32-bit integer
  - GL_FLOAT        single-precision floating point
Scaling and Copying Images

- Scales the image, using appropriate pixel storage modes to unpack the data from datain. Image is scaled using linear interpolation

```c
int gluScaleImage(GLenum format, GLint widthin, GLint heightin, GLenum typein, const void *datain, GLint widthout, GLint heightout, GLenum typeout, void *dataout );
```

- Creates a 2D texture, using framebuffer data to define the texels. The pixels are read from the current GL_READ_BUFFER

```c
void glCopyTexImage(GLenum target, GLint level, GLint internalFormat, GLint x, GLint y, GLsizei width, GLsizei height, GLboolean border);
```
Enable Texture Mapping

To enable or disable texture mapping:

```c
glEnable(mode)
glDisable(mode)
```

where `mode` is `GL_TEXTURE_1D`  
`GL_TEXTURE_2D` or  
`GL_TEXTURE_3D`
Texture Coordinates

• You need to specify BOTH texture & geometric coordinates as you specify the object in your scene

• Example:
  
glBegin(GL_QUADS);
  glTexCoord2f(0.0, 0.0);
  glVertex3f(-2.0, -1.0, 0.0);
  glTexCoord2f(0.0, 1.0);
  glVertex3f(-2.0, 1.0, 0.0);
  glTexCoord2f(1.0, 1.0);
  glVertex3f(0.0, 1.0, 0.0);
  glTexCoord2f(1.0, 0.0);
  glVertex3f(0.0, -1.0, 0.0);
  glEnd();
Texture Coordinates

• **Example:** texture mapped polygons

Original texture
Texturing Functions

• Indicate how the texture is applied to each pixel:

• **REPLACE** or **DECAL**: Texture is painted on top of the fragment

• **MODULATE**: Combine texture with fragment color. This technique is useful to combine the effects of lighting with texturing

• **BLEND**: A constant color is blended with that of the fragment, based on the texture value
Texturing Functions

Example: Quake
Texturing Functions

```c
glTexEnvf{if}(GLenum target, GLenum pname, GLfloat param);
```

**target**: GL_TEXTURE_ENV

**pname**: GL_TEXTURE_ENV_MODE, GL_TEXTURE_ENV_COLOR

**param**: GL_DECAL, GL_REPLACE, GL_MODULATE, GL_BLEND

<table>
<thead>
<tr>
<th>Internal Format</th>
<th>Decal</th>
<th>Replace</th>
<th>Modulate</th>
<th>Blend</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_ALPHA</td>
<td>ND</td>
<td>C = C_f</td>
<td>C = C_f</td>
<td>C = C_f</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = A_t</td>
<td>A = A_tA_t</td>
<td>A = A_tA_t</td>
</tr>
<tr>
<td>GL_LUMINANCE</td>
<td>ND</td>
<td>C = L_t</td>
<td>C = C_fL_t</td>
<td>C = C_f(1-L_t)+C_cL_t</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = A_f</td>
<td>A = A_f</td>
<td>A = A_f</td>
</tr>
<tr>
<td>GL_LUMINANCE_ALPHA</td>
<td>ND</td>
<td>C = L_t</td>
<td>C = C_fL_t</td>
<td>C = C_f(1-L_t)+C_cL_t</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = A_f</td>
<td>A = A_fA_t</td>
<td>A = A_fA_t</td>
</tr>
<tr>
<td>GL_INTENSITY</td>
<td>ND</td>
<td>C = I_t</td>
<td>C = C_fI_t</td>
<td>C = C_f(1-I_t)+C_cI_t</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = I_t</td>
<td>A = I_tA_t</td>
<td>A = I_tA_t</td>
</tr>
<tr>
<td>GL_RGB</td>
<td>C = C_t</td>
<td>C = C_t</td>
<td>C = C_fC_t</td>
<td>C = C_f(1-C_t)+C_cC_t</td>
</tr>
<tr>
<td></td>
<td>A = A_f</td>
<td>A = A_f</td>
<td>A = A_f</td>
<td>A = A_f</td>
</tr>
<tr>
<td>GL_RGBA</td>
<td>C = C_f(1-A_t)+C_tA_t</td>
<td>C = C_t</td>
<td>C = C_fC_t</td>
<td>C = C_f(1-C_t)+C_cC_t</td>
</tr>
<tr>
<td></td>
<td>A = A_fA_t</td>
<td>A = A_t</td>
<td>A = A_fA_t</td>
<td>A = A_fA_t</td>
</tr>
</tbody>
</table>
# Repeating and Clamping

```c
glTexParameter{if}(GLenum target, GLenum pname, GLfloat param);
```

**target**: GL_TEXTURE_2D

**pname**: Parameter symbolic-constant

**param**: The value of “pname”

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_TEXTURE_WRAP_S</td>
<td>GL_CLAMP, GL_REPEAT</td>
</tr>
<tr>
<td>GL_TEXTURE_WRAP_T</td>
<td>GL_CLAMP, GL_REPEAT</td>
</tr>
<tr>
<td>GL_TEXTURE_MAG_FILTER</td>
<td>GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_LINEAR</td>
</tr>
<tr>
<td>GL_TEXTURE_MIN_FILTER</td>
<td>GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_LINEAR</td>
</tr>
<tr>
<td>GL_TEXTURE_BORDER_COLOR</td>
<td>Any 4 values in [0.0, 1.0]</td>
</tr>
<tr>
<td>GL_TEXTURE_PRIORITY</td>
<td>[0.0, 1.0] for the current texture object</td>
</tr>
</tbody>
</table>
Filtering Methods

• Usually the size of a texel does not match the size of the corresponding pixel

• In order to map the texels to the pixels, we need to do some kind of filtering
Filtering Methods

• **GL_NEAREST**: The texel with coordinates nearest to the center of the pixel is used. This might result in aliasing artifacts.

• **GL_LINEAR**: A weighted linear average of the 2x2 array of texels that lie nearest to the center of the pixel is used.

• **GL_NEAREST_MIPMAP_NEAREST**: In the nearest mipmap, choose the nearest texel value.

• **GL_LINEAR_MIPMAP_NEAREST**: In the nearest mipmap, interpolate linearly between the 2 nearest texels.

• **GL_LINEAR_MIPMAP_LINEAR**: Interpolate between the nearest values in the 2 best choices from the mipmaps.
Mipmaps

• To use mipmapping, all sizes (in powers of 2) of the texture must be provided, either by calls to glTexImage2D() once for each resolution, or by using:

```c
int gluBuild2DMipMaps(
    GLenum target, GLint comps,
    GLint width, GLint height,
    GLenum format,
    GLenum type, void *data);
```

This function constructs a series of mipmaps and calls glTexImage to load the images.