OpenGL Notes [a]

Stu Pomerantz
smp@psc.edu
http://www.psc.edu/~smp
November 3, 2004

[a]Most material is adapted from: OpenGL ARB, et. al, "The OpenGL Programming Guide", Third Ed., Reading: Addison-Wesley, 1999

What is Texture Mapping?

Texture mapping specifies how an array of pixels is applied to geometric primitives.

- Points, lines, triangles and quads may all have texture pixels applied to them.
- Textures may be 1D, 2D, or 3D.
- OpenGL 1.5 and earlier requires texture dimensions to be a power of 2. OpenGL 2.0 eliminates this requirement.

Steps in Texture mapping

- Create a texture object and specify a texture for that object.
- Indicate how the texture is applied to each pixel.
- Enable texture mapping.
- Draw the scene supplying texture and geometric coordinates.

Creating a Texture Object

```
void glGenTextures( GLsizei n, GLuint *textures ) ;
```

- Ask OpenGL to generate a unique integer identifier through which the texture information will be referenced.
- One or more texture identifiers can be generated.
- Used once when textures are created.

Creating a Texture Object

```
void glBindTexture( GLenum target, GLuint texture ) ;
```

- Binds a texture to a texture target.
- *target* constants are: `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, and `GL_TEXTURE_3D`
- *texture* is the texture identifier generated by `glGenTextures()`
- texture identifiers (loosely just called textures) must be bound before they can be created or used.
- Used anytime a texture is created or used.
- `glBindTexture(GL_TEXTURE_2D,0)` binds no texture.

Creating a Texture Object

```
void glTexParameter*(GLenum target, GLenum pname,
                     GLfloat param ) ;
```

- Specifies how the texture wraps (in each dimension).
- Specifies how the texture is to be filtered if there isn't an exact match between texture pixels and screen pixels (in each dimension).
- Usually specified once per texture.
- See the manual and the following examples for a detailed description of all possible parameters.

Creating a Texture Object

```
void glTexImage2D(GLenum target,
                  GLint level,
                  GLint internalformat,
                  GLsizei width,
                  GLsizei height,
                  GLint border,
                  GLenum format,
                  GLenum type,
                  const GLvoid *pixels ) ;
```

- Specifies a 2D texture.

Creating a Texture Object

```
void glTexImage2D() ;
```

- *target* is usually `GL_TEXTURE_2D` (always for this class).
- *level* is the level of detail number. For a base level texture, this value is 0. If *level* < 0, it is interpreted as the *mipmap* level.
- *internalformat* specifies the number of color components in the texture array being passed to OpenGL. May be 1,2,3,4 or a one of many constants.
- *width* & *height* specifies the size of the texture array being passed to OpenGL.

Creating a Texture Object

`void glTexImage2D() ;`

- *border* is 0 if the texture has no border and 1 if the texture has a border. Texture size must be $2m + 2$ if they have a border.
- *format* specifies how the texture array is to be interpreted by OpenGL. For example: `GL_RGB` or `GL_GREEN`.
- *type* specifies the data type of the texture array being passed to OpenGL. For example: `GL_UNSIGNED_BYTE`.
- *pixels* is a pointer to the texture array.
- Transfers texture data across the bus to GPU memory.
- Note the analogy with `glDrawPixels()`

Creating a Texture Object

Putting it all together:

```
// store a new texture identifier in tex0
glGenTextures(1, &tex0) ;
// bind tex0 so the texture can be specified
glBindTexture(GL_TEXTURE_2D, tex0) ;
// specify linear filtering if the texture is
// magnified or minified during drawing
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_MAG_FILTER,GL_LINEAR) ;
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_MIN_FILTER,GL_LINEAR) ;
```

. . . continued

Creating a Texture Object

```
// specify repeating texture wrap
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S,GL_REPEAT) ;
glTexParameteri(GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T,GL_REPEAT) ;
// download my 512x256 array (bytes)
// to the graphics card
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,512,256,0,
     GL_RGB,GL_UNSIGNED_BYTE,bytes) ;
```

The texture tex0 is now ready to be used.

How a Texture is applied to a pixel

```
void glTexEnv*(GLenum target, GLenum pname,
               GLint param) ;
```

- *target* must be `GL_TEXTURE_2D`.
- *pname* must be `GL_TEXTURE_ENV_MODE`.
- *param* specifies how the texture is applied.
  - constants include `GL_MODULATE`, `GL_DECAL`, `GL_BLEND`, and `GL_REPLACE`.
- These are also called *texture functions*.

Enabling Texture Mapping

Two calls:

- `glEnable(GL_TEXTURE_2D)`.
- `glBindTexture(GL_TEXTURE_2D, texname)`.

Textured geometry may now be drawn.

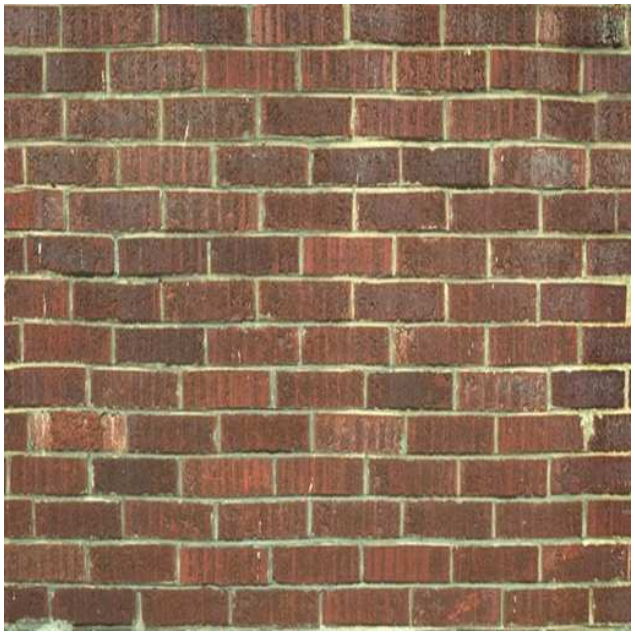
Texture Coordinates

`void glTexCoord*() ;`

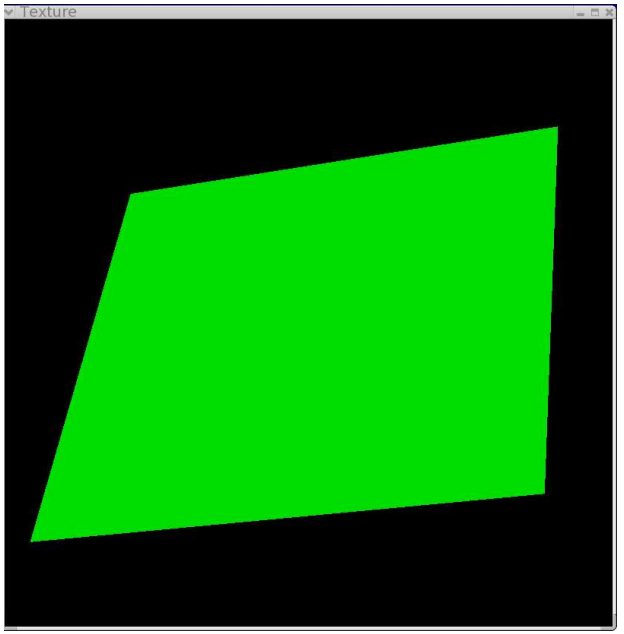
- Texture coordinates map a point in the texture array to a vertex.
- Each vertex is assigned a texture coordinate.
- Pixels in between vertices are automatically interpolated.
- A state change, just like `glColor*()` and `glNormal*()`
- A point in 2D space is conventially specified by a point (x,y).
- By convention, a 2D texture point is specified by (s,t) since (x,y) are already used.

Texturing Example 1

Put the brick wall pixels onto the green quad.



Texture



Geometry

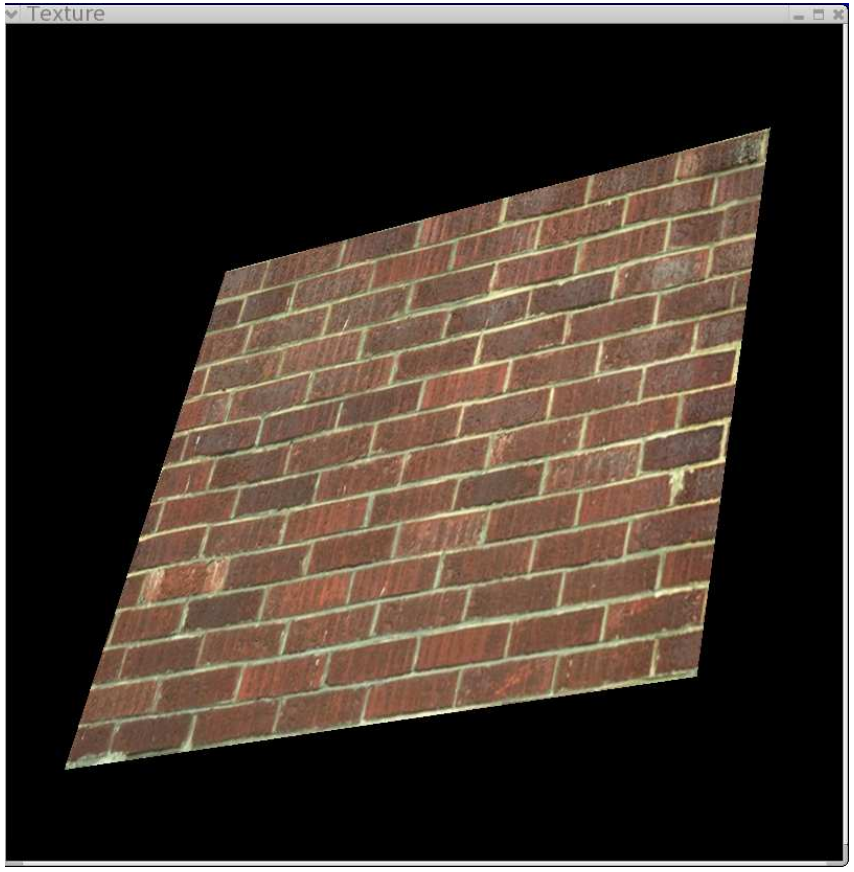
Texturing Example 1

Drawing with texture coordinates:

```
glBegin(GL_QUADS) ;
  glTexCoord2i(0,0) ;
  glVertex3f(-1,-1,0) ;
  glTexCoord2i(1,0) ;
  glVertex3f( 1,-1,0) ;
  glTexCoord2i(1,1) ;
  glVertex3f( 1, 1,0) ;
  glTexCoord2i(0,1) ;
  glVertex3f(-1, 1,0) ;
glEnd() ;
```

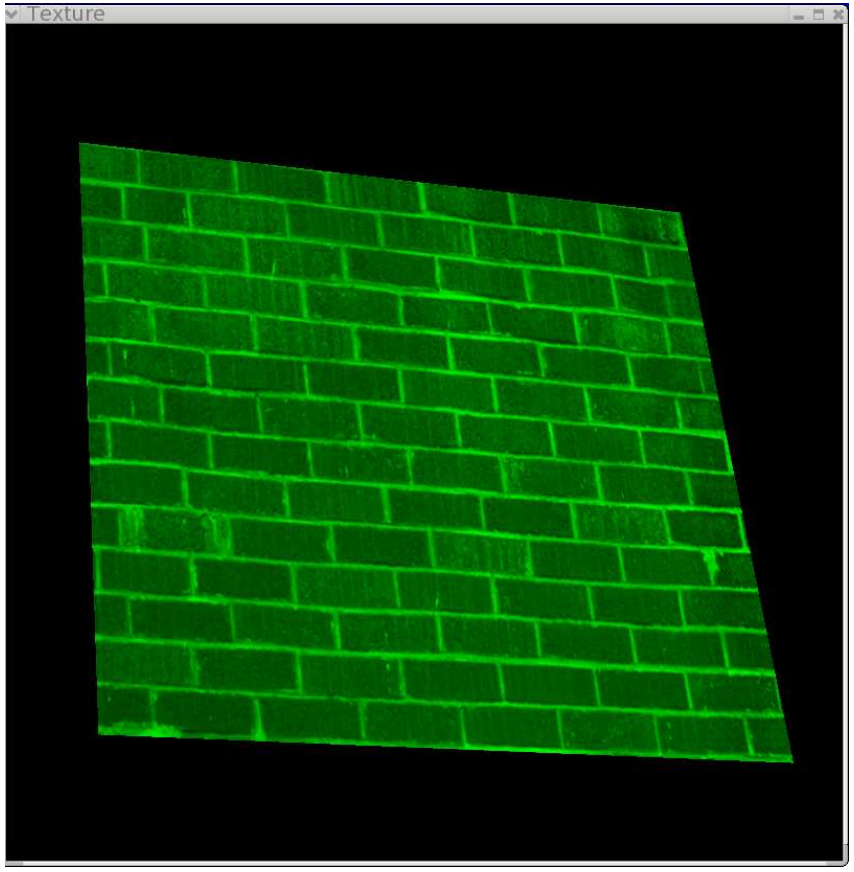
Texturing Example 1

Gives this result using `GL_REPLACE` in `glTexEnv()` :



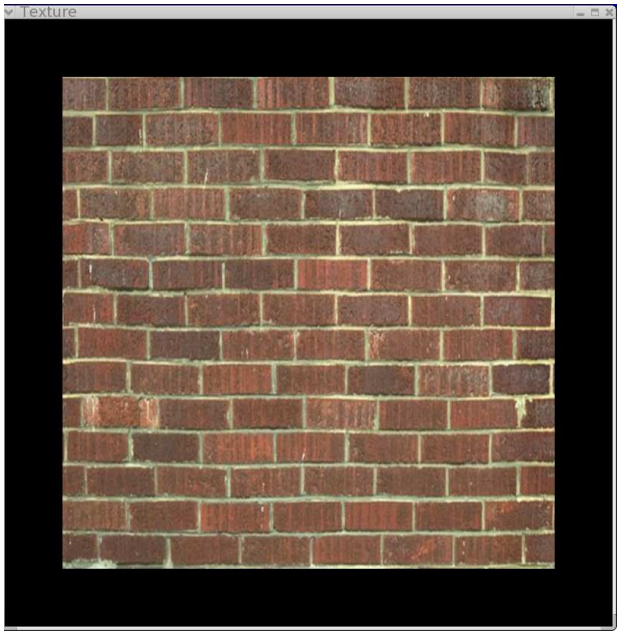
Texturing Example 1

Gives this result using `GL_MODULATE` in `glTexEnv()` :



Texturing Example 2

Rotating the pixels



The same mapping as example 1

Rotated texture coordinates

Texturing Example 2

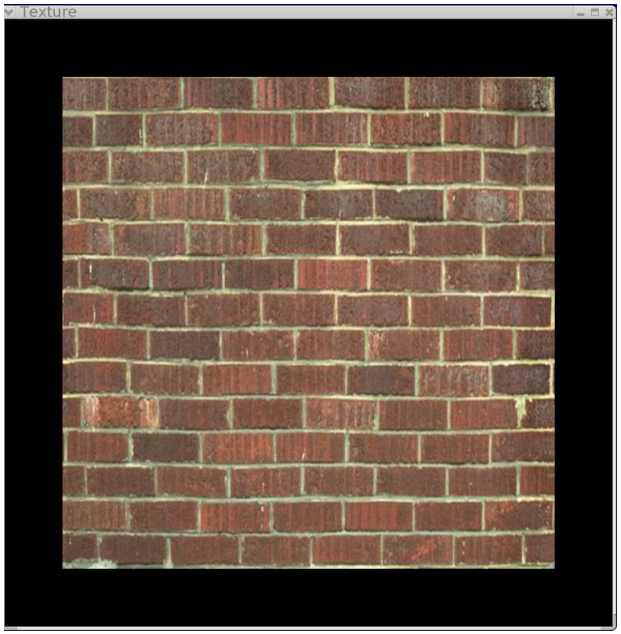
Rotate the texture coordinates using the texture maxtrix.

```
glMatrixMode(GL_TEXTURE) ;
glPushMatrix() ;
  glRotatef(45,0,0,1) ;
    draw_example1_quad() ;
  glPopMatrix() ;
glMatrixMode(GL_MODELVIEW) ;
```

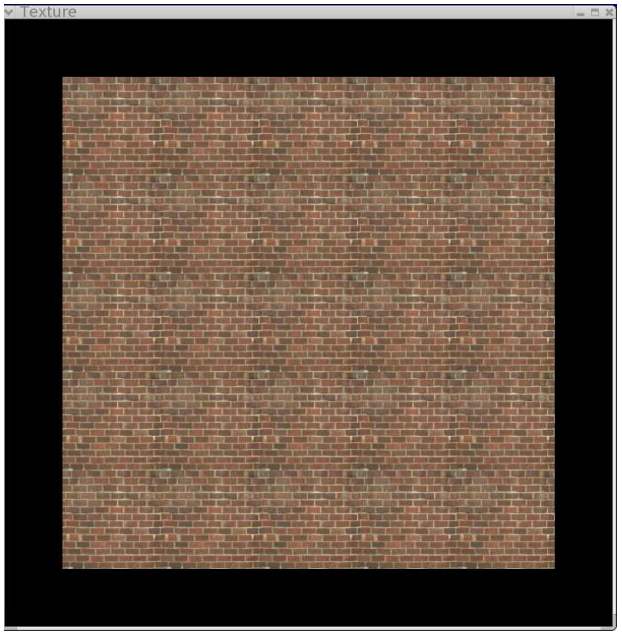
All the usual OpenGL transforms can be applied to texture coordinates using the texture matrix mode.

Texturing Example 3

Repeating the texture



The same mapping as example 1



Texture repeated 5 times in each direction

Texturing Example 3

Drawing with texture coordinates:

```
glBegin(GL_QUADS) ;
  glTexCoord2i(0,0) ;
  glVertex3f(-1,-1,0) ;
  glTexCoord2i(5,0) ;
  glVertex3f( 1,-1,0) ;
  glTexCoord2i(5,5) ;
  glVertex3f( 1, 1,0) ;
  glTexCoord2i(0,5) ;
  glVertex3f(-1, 1,0) ;
glEnd() ;
```

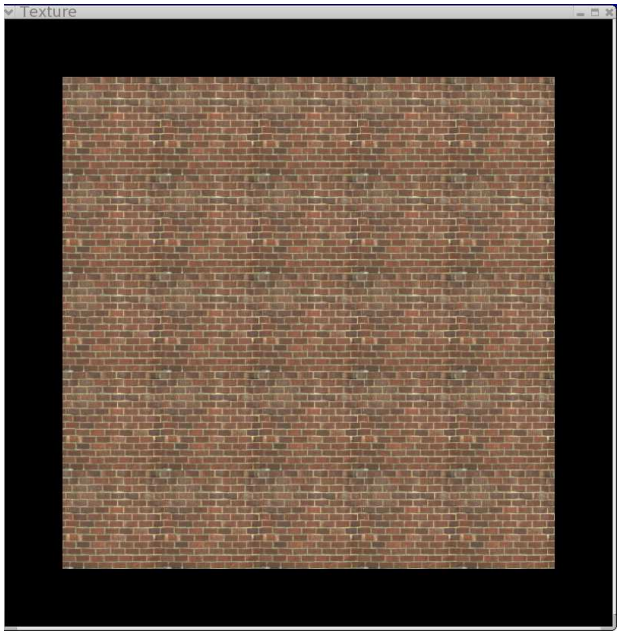
Texturing Example 3

And texture parameters set to:

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,
                GL_REPEAT) ;

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,
                GL_REPEAT) ;
```

Texturing Example 4

Clamping the texture



The same mapping as example 3

Clamped Texture

Texturing Example 4

The quad was drawn using the same texture coordinates as example 3, but with texture parameters set to:

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,
                GL_CLAMP) ;

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,
                GL_REPEAT) ;
```

Texture Level of Detail – Mipmaps

Mip stands for *multum in parvo* or "many things in one place."

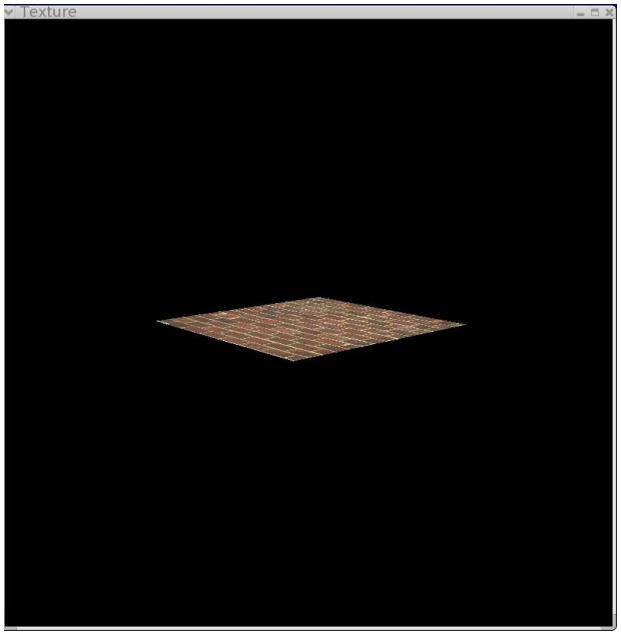
- As a textured object moves farther from the viewpoint, the texture map must decrease in size with the projected object.
- OpenGL will filter the texture down to the appropriate size for mapping on to the object.
- The filtering must be done so that it avoid artifacts such as shimmering flashing, scintillation and abrupt changes.
- To accomplish this, a series of prefiltered texture maps of decreasing resolutions, called *Mipmaps* are used.

Texture Level of Detail – Mipmaps

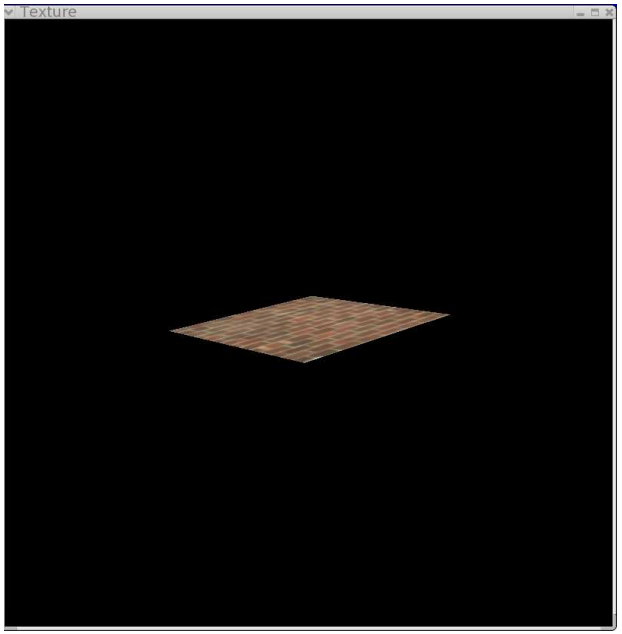
- The *level* parameter of `glTexImage2D()` specifies which mipmap level of detail the bytes represent.
- Mipmaps may calculate explicitly and specified with `glTexImage2D()` .
- The OpenGL GLU library can calculate mipmaps automatically.

Texture Level of Detail – Mipmaps

The effect of mipmaping:



No Mipmapping



With Mipmapping

Texture Level of Detail – Mipmaps

To tell OpenGL to use mipmaps when it minifies a texture, change:

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                GL_LINEAR) ;
```

to

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR) ;
```

- The highest level of detail is specfied using `glTexImage2D()`
- The artifacts fixed by mipmaping don't show up during magnification so don't need to change the `GL_TEXTURE_MAX_FILTER`.

Texture Level of Detail – Mipmaps

To tell OpenGL to generate mipmaps add the call:

```
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, 512, 256,
                  GL_RGB, GL_UNSIGNED_BYTE, bytes) ;
```

- Make this call after the call to `glTexImage2D()`.
- The meanings of the parameters are the same as for `glTexImage2D()`.

Multitexturing

Multitexturing allows several textures to be applied, one by one in a pipeline of texture operations, to the same polygon.

- The graphics card has a series of *texture units*.
- Each texture unit performs a single texturing operation and passes its result on to the next unit.

Multitexturing

Steps in multitexturing:

- Create two (or more) ordinary texture objects.
- At drawing time:
  - For each texture unit, establish the texturing state.
  - use `glMultiTexCoord*()` to specify more than one texture coordinate per vertex. A different texture coordinate may be used for each texture unit.

Multitexturing

Establishing the texturing state for two textures:

```
glActiveTexture(GL_TEXTURE0) ;
glEnable(GL_TEXTURE_2D) ;
glBindTexture(GL_TEXTURE_2D, tex0) ;
glTexEnvi(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,
          GL_REPLACE) ;

glActiveTexture(GL_TEXTURE1) ;
glEnable(GL_TEXTURE_2D) ;
glBindTexture(GL_TEXTURE_2D, tex1) ;
glTexEnvi(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,
          GL_MODULATE) ;
```

Multitexturing

- `glActiveTexture()` set the active texture unit.
- `glEnable(GL_TEXTURE_2D)` enables texturing *for the active texture unit*
- Note the ordering.
- Note that these texture units don't necessarily correspond exactly to hardware.

Multitexturing

Drawing with texture coordinates:

```
glBegin(GL_QUADS) ;
  glMultiTexCoord2i(GL_TEXTURE0,0,0) ;
  glMultiTexCoord2i(GL_TEXTURE1,0,0) ;
  glVertex3f(-1,-1,0) ;

  glMultiTexCoord2i(GL_TEXTURE0,1,0) ;
  glMultiTexCoord2i(GL_TEXTURE1,3,0) ;
  glVertex3f( 1,-1,0) ;
```

continued ...

Multitexturing

. . . continuing.

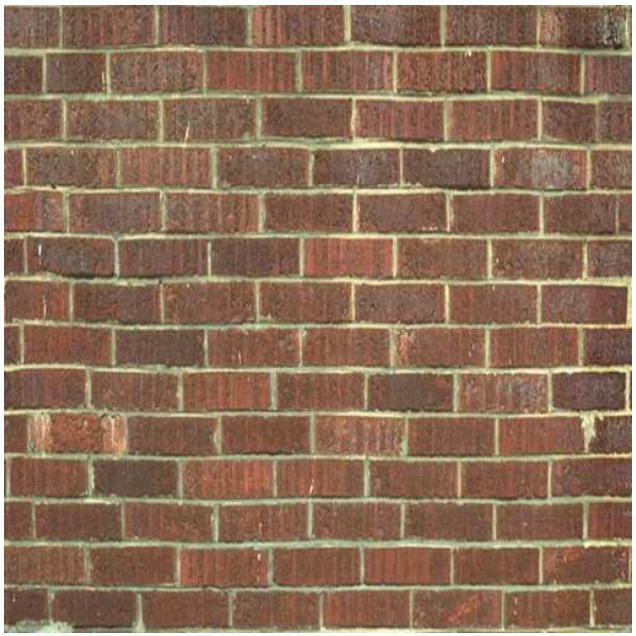
```
  glMultiTexCoord2i(GL_TEXTURE0,1,1) ;
  glMultiTexCoord2i(GL_TEXTURE1,3,3) ;
  glVertex3f( 1, 1,0) ;

  glMultiTexCoord2i(GL_TEXTURE0,0,1) ;
  glMultiTexCoord2i(GL_TEXTURE1,0,3) ;
  glVertex3f(-1, 1,0) ;
glEnd() ;
```

Multitexturing

- The texture in unit 0 will be applied exactly once to the quad, replacing the color of the quad.
- The texture in unit 1 will be applied 3 times (assuming it is repeated) and will modulate the color already on the quad.
- The texture in unit 0 is applied first, then the texture in unit 1.
- So, the texture in unit 1 will modulate the texture in unit 0.

Multitexturing

Given these two textures:



In texture unit 0

In texture unit 1

Multitexturing

Multitexturing as previously described gives:

Multitexturing

Just swapping texture units gives: