OpenGL Notes [a]

Stu Pomerantz
smp@psc.edu
http://www.psc.edu/~smp
June 20, 2007

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

Introduction to Textures

- What is a texture ?
  - A texture starts as a 1, 2, or 3 dimensional array of pixels.
  - Like display lists, texture data is downloaded to the card and referred to by an integer identifier.
  - Unlike display lists, there are numerous parameters which can (and must) be set before a texture is used.
  - Texture data, or *texels*, are mapped to primitives, such as triangles, using *texture coordinates* specified at vertices.
  - Many types, such as GL_FLOAT, GL_SHORT, and GL_UNSIGNED_BYTE can be used for textures.
  - Until recently, textures **must** be power of 2 sized.

Introduction to Textures

- The 1, 2, or 3 dimensional array of pixels
  - A PPM (P6) file is a 2 dimensional array of pixels.
  - Each pixel is 3 unsigned char bytes, one red, one green, and one blue.
  - That is, the type is GL_UNSIGNED_BYTE and the format is GL_RGB.

Of course, OpenGL supports many other possible formats and types.

Texture Setup

- 3 step initialization:

```
// ask GL to generate a new identifier for a texture
glGenTextures(...)

// tell GL which texture object to use
glBindTexture(...)

// specify an array of pixels and send it to the card
glTexImage2D(...)
```

Texture Setup

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

Generate 1 texture identifier and store it in the GLuint texid:

```
glGenTextures(1, &texid);
```

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

Bind (tell GL we wish to use) the texture *texid* for 2D texturing:

```
glBindTexture(GL_TEXTURE_2D, texid);
```

Texture Setup

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

- *target* how to use the texture, GL_TEXTURE_2D for basic 2D texturing.
- *level* level of detail (*mipmap*), 0 for the base image.
- *internalFormat* number of color components 1,2,3,4 or a const.
- *width* & *height* size of the array.
- *border* width of the array border (0 or 1)
- *format* format of the array data, GL_RGB
- *type* data type of the array data, GL_UNSIGNED_BYTE
- *data* a pointer to the array of data

Texture Setup

Note that *internalFormat* and *format* tend to match.
internalFormat specifies how the array data is to be used and
format specifies the format of the array data.

A sample call for a PPM (P6) array that is 256 x 256 might be:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
             256, 256, 0,
             GL_RGB, GL_UNSIGNED_BYTE, bytes) ;
```

Texture Setup

Texture initialization for the same PPM (P6) array that is 256 x 256 might be:

```
glGenTextures(1, &texid);
glBindTexture(GL_TEXTURE_2D, texid);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
             256, 256, 0,
             GL_RGB, GL_UNSIGNED_BYTE, bytes) ;
```

Texture Coordinates

Given the triangle:

```
glBegin(GL_TRIANGLES) ;
 glVertex3f(0,0,0) ;
 glVertex3f(1,0,0) ;
 glVertex3f(1,1,0) ;
glEnd() ;
```

Texture Coordinates

Numerous properties can be specified for each vertex:

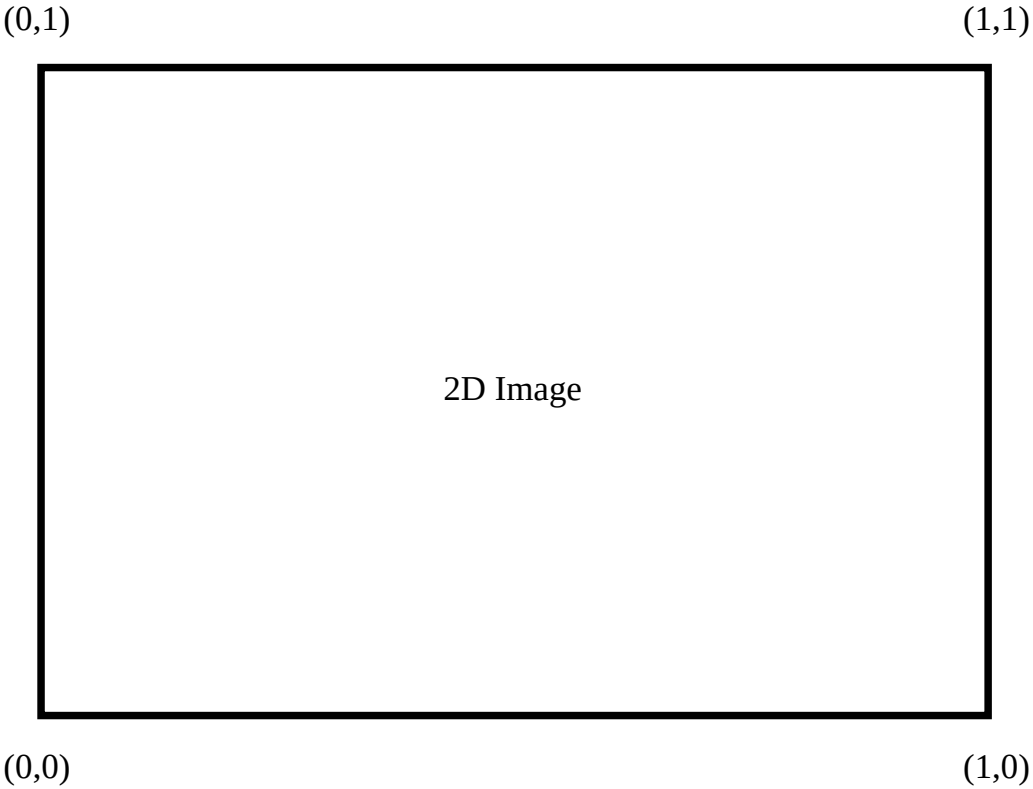
```
glBegin(GL_TRIANGLES) ;
 glNormalf(0,1,0) ;
 glColor3f(1,0,0) ;
 glVertex3f(0,0,0) ;

 glNormalf(0,1,1) ;
 glColor3f(1,1,0) ;
 glVertex3f(1,0,0) ;

 glNormalf(1,1,0) ;
 glColor3f(1,1,0) ;
 glVertex3f(1,1,0) ;
glEnd() ;
```

Texture coordinates are another property specified at vertices.

Texture Coordinates



To avoid confusion texture coordinates are specified at (s,t) rather than (x,y)

Texture Coordinates

Given this texture mapping:

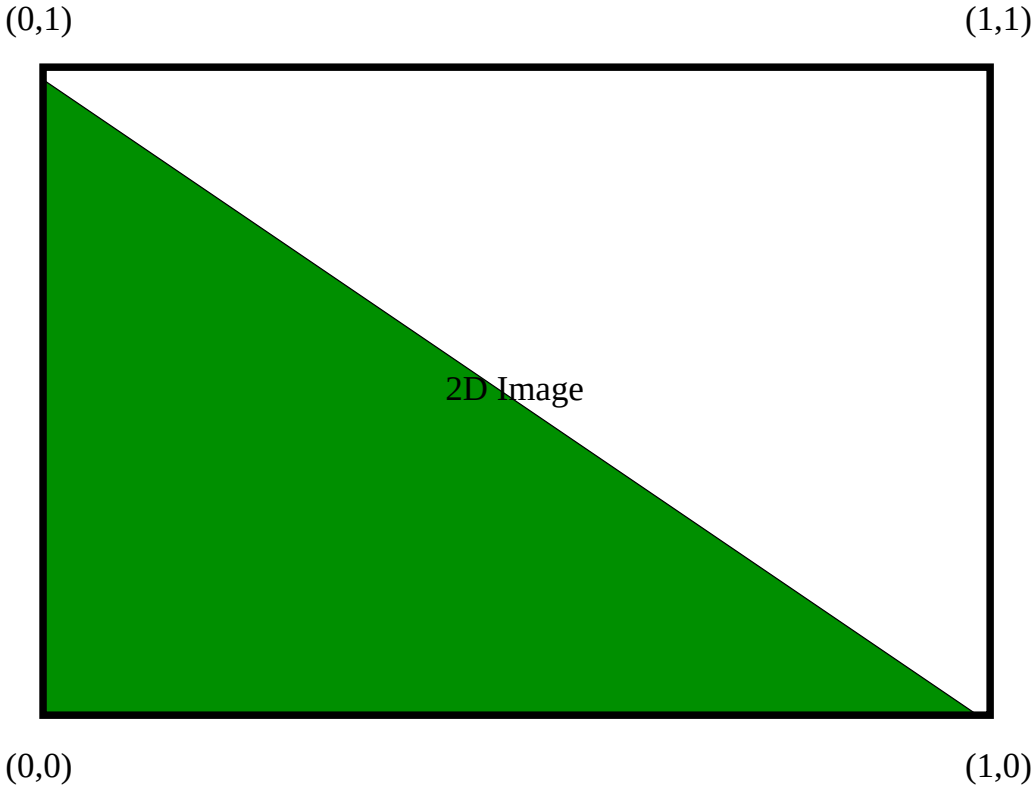
```
glBegin(GL_TRIANGLES) ;
 glTexCoord2f(0,0) ;
 glVertex3f(0,0,0) ;

 glTexCoord2f(1,0) ;
 glVertex3f(1,0,0) ;

 glTexCoord2f(0,1) ;
 glVertex3f(1,1,0) ;
glEnd() ;
```

Texture Coordinates

The green patch of the image is mapped to the triangle.



Texture Use

3 steps to use a texture:

```
glTexParameteri(...)
glTexEnv(...)
glBindTexture(...)
  glEnable(GL_TEXTURE_2D) ;
    draw_object()
  glDisable(GL_TEXTURE_2D) ;
```

glTexParameteri() and glTexEnv() specify how a texture will be applied to geometry.

Texture Use

glTexParameteri() specifies texture filtering and wrapping for both s and t coordinates.

```
void glTexParameteri(GLenum target, GLenum pname, GLint param);
```

There are separate magnifications and minification filters.

When doing magnification of 2d textures, linearly interpolate the pixels:

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

When doing minification of 2d textures, use the nearest neighboring pixel:

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

Texture Use

glTexParameteri() specifies texture filtering and wrapping for both s and t coordinates.

```
void glTexParameteri(GLenum target, GLenum pname, GLint param);
```

There are separate wrapping specifiers for s and t.

When doing wrapping of 2d textures, if s exceeds 1, repeat the texture starting back at 0.

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

When doing wrapping of 2d textures, if t exceeds 1, then t = 1.

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

Texture Use

glTexEnvi() specifies specifies how texture values are interpreted when a fragment is textured.

```
void glTexEnvi(GLenum target, GLenum pname, GLint param);
```

The final color = (texture color)*(primitive color)

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE) ;
```

The final color = (texture color)

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL) ;
```

Of course, many other combinations are available!

Texture Use

In summary:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT) ;
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT) ;
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR) ;
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR) ;
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE) ;

glBindTexture(GL_TEXTURE_2D, texid) ;
glEnable(GL_TEXTURE_2D) ;
    draw_object()
glDisable(GL_TEXTURE_2D) ;
```

Texture MipMaps

MipMap stands for "Many Image Map". The basic idea is that smaller and smaller versions of a base texture image are precomputed and filter.

Then when a far primitive needs to be textured, a mipmap of the appropriate size is selected and that is is applied. This process is can be both faster and have results that look good.

Why better looking ? A far primitive textured with GL_NEAREST will have texels that tend to shimmer. The interpolation of the texture map across the face of the primitive will be sensitive to small changes in the primitive.

Why Faster ? GL_LINEAR can get slow, so use GL_NEAREST_MIPMAP_NEAREST

gluUnProject()

gluUnProject() maps window coordinates to object coordinates. That is, it **reverses** OpenGL transformations.

```
GLint gluUnProject(
  GLdouble winX, GLdouble winY, GLdouble winZ,
  const GLdouble *model,
  const GLdouble *proj,
  const GLint *view,
  GLdouble *objX, GLdouble *objY, GLdouble *objZ);
```

winX & winY are easy to understand, they are points under the mouse. What is winZ ? It can be any depth value $\subseteq [0, 1]$. A convenient value may be the value in the depth buffer under the mouse, from glReadPixels().

gluUnProject()

gluUnProject() maps window coordinates to object coordinates. That is, it **reverses** OpenGL transformations.

```
GLint gluUnProject(
  GLdouble winX, GLdouble winY, GLdouble winZ,
  const GLdouble *model,
  const GLdouble *proj,
  const GLint *view,
  GLdouble *objX, GLdouble *objY, GLdouble *objZ);
```

model and proj are both 16 element matrices which represent the modelview and projection matrices, respectively. view is a 4 element matrix holding the min/max values of the viewport.

gluUnProject()

gluUnProject() maps window coordinates to object coordinates. That is, it **reverses** OpenGL transformations.

```
GLint gluUnProject(
  GLdouble winX, GLdouble winY, GLdouble winZ,
  const GLdouble *model,
  const GLdouble *proj,
  const GLint *view,
  GLdouble *objX, GLdouble *objY, GLdouble *objZ);
```

objX, objY, objZ are the components of the 3D point returned by gluUnProject().