

Basic OpenGL Texture Functions

OpenGL texture mapping functions break up into a number of categories. If you understand that each function fits into one of these categories, the multitude of functions will not confuse you as much. Many of these functions are only useful in very particular circumstances, so it is not important to understand all of them. The **bold functions** are all that is necessary for very basic texturing, other functions can be examined later for more advanced functionality or speed optimizations.

Managing Textures:

- **void glGenTextures(n, textureNames)** (*see p. 404*)
 - Returns n unused texture IDs in the array textureNames.
 - IDs are marked as used, but NOTE they have no state until bound (using glBindTexture()) and set (using a function from the “Creating and Formatting a Texture” section below).
- GLboolean glIsTexture(textureName) (*see p. 404*)
 - Returns GL_TRUE if textureName is a valid name of a texture that has been bound.
 - Returns GL_FALSE if you did not use glGenTextures() to select your ID
 - Returns GL_FALSE if you used glGenTextures() but have not yet bound data to the ID
 - Returns GL_FALSE if the texture has been deleted or never existed.
- **void glBindTexture(target, textureName)** (*see p. 405*)
 - OpenGL texture functions generally operate on the *currently bound function*.
 - In order to set the data in your texture, it needs to be bound.
 - In order to render using the texture, it must be bound.
 - *target* will likely be GL_TEXTURE_2D in your applications
 - *textureName* is the ID of the texture (which you acquire via glGenTextures()).
- void glDeleteTextures(n, textureNames) (*see p. 407*)
 - Marks the n textures in the array textureNames as unused, and frees the memory associated with them.
 - Deleting nonexistent texture is OK (such calls are ignored).
 - Deleting a currently bound texture *should be avoided*, but should revert the bound texture to the “default texture.”

Creating and Formatting a Texture:

- **void glTexImage2D(target, level, internalFormat, width, height, border, format, type, texels)** (*see p. 370*)
 - *target* will likely be GL_TEXTURE_2D for many applications
 - *level* will be 0, unless (perhaps later in the course) you need to specify levels of a MIP-map.
 - *internalFormat* will probably be GL_RGBA (or perhaps GL_RGB)
 - *width, height* of your texture, both must be powers of two (minimum size: 64 × 64)
 - *border* will be 0, unless your texture has a border (read p. 370 for changes that requires)
 - *format* is the format you specify your data, probably GL_RGBA or GL_RGB.

- *type* is the data type used to specify your data, probably GL_BYTE, GL_INT, or GL_FLOAT.
- *texels* is a pointer to your data
- void glTexImage2D(target, level, xoffset, yoffset, width, height, format, type, texels) (see p. 376)
 - Replaces part of the current texture with new data. Note that the internalFormat is not specified (as above), because it has already been defined when the texture was initially defined. The width, height, format, types, and texels all specify information about the new subregion (specified by the data in texels).
- int gluScaleImage(format, widthin, heightin, typein, datain, widthout, heightout, typeout, dataout) (see p. 372)
 - Use to *scale* an image so that it is a power of two in both dimensions.
- void glCopyTexImage2D(target, level, internalFormat, x, y, width, height, border) (see p. 373)
 - Reads data from the framebuffer (screen) to use as a texture.
- void glCopyTexSubImage2D(target, level, xoffset, yoffset, x, y, width, height) (see p. 378)
 - Similar to glCopyTexImage2D, but it uses the framebuffer data to replace information in a previously defined texture.
- void glTexImage1D(...), void glTexSubImage1D(...) (see p. 379-380)
- void glCopyTexImage1D(...), void glCopyTexSubImage1D(...) (see p. 380-381)
- void glTexImage3D(...), void glTexSubImage3D(...) (see p. 382-383)
- void glCopyTexSubImage3D(...) (see p. 384) **Think about: Why is there no glCopyTexImage3D()??**
- void glCompressedTexImage1D(...), void glCompressedTexImage2D(...) (see p. 387)
- void glCompressedTexImage3D(...) (see p. 387)

Examining and Setting OpenGL Texture State:

- void glGetTexParameter*(target, pname, param) (see p. 420-421)
 - *target* will likely be GL_TEXTURE_2D.
 - *pname* has a number of values, of which the most important to set are:
 - * GL_TEXTURE_WRAP_S
 - * GL_TEXTURE_WRAP_T
 - * GL_TEXTURE_MAX_FILTER
 - * GL_TEXTURE_MIN_FILTER
 - *param* depends on *pname*, see Table 9-7 (p. 421) for various options.
- void glGetTexLevelParameter*(target, level, pname, params) (see p. 375)
 - *target* is as specified in glTexImage2D or similar (probably GL_TEXTURE_2D)
 - *level* is the MIP-map level in question (probably 0 for now, or if not using MIP-maps)
 - *pname* is the information you'd like to query for (see p. 375) for possibilities
 - *params* is a pointer to a location to store the data you queried for.

Filtering and Indexing into Texture Data:

- **void glTexEnv*(target, pname, param)** (*see p. 410, 438*)
 - Allows textures to be used to supplement (instead of replacing) colors specified by glColor* or OpenGL lighting.
 - *target* will likely be GL_TEXTURE_ENV
 - *pname* will likely be GL_TEXTURE_ENV_MODE
 - *param* will then be GL_DECAL, GL_REPLACE, GL_MODULATE, GL_BLEND, or GL_ADD.
 - Be aware correct functioning will probably require a call to “glEnable(GL_BLEND);”

Assigning Texture Coordinates:

- **void glTexCoord*(coords)** (*see p. 415*)
 - This function has 1, 2, 3, and 4 coordinate versions.
 - Likely you will be calling **glTexCoord2f(uCoord, vCoord)** or **glTexCoord2fv(coords)**
 - If you are thinking about calling glTexCoord2i, think again, as this will probably not give you the behavior you desire.
 - Remember, texture coordinates by default range [0..1], so specifying integer texture coordinates is generally not useful.
- **void glMultiTexCoord*(texUnit, coords)** (*see p. 435*)
 - Identical to glTexCoord*(), except it allows you to select which texture unit the coordinates will be applied to (glTexCoord*() specifies coordinates for texture unit 0).

Advanced Functions:

- GLboolean glAreTexturesResident(...) (*see p. 408*) allows you to determine if numerous textures are resident in texture memory (i.e., on the graphics card)
- void glPrioritizeTextures(...) (*see p. 409*) allows you to give OpenGL hints as to which textures are more likely to be used (and should thus be more likely to be resident in texture memory)
- void glTexGen*(...) (*see p. 422*) allows automatic generation of texture coordinates according to certain (limited) built-in rule sets.
- void glActiveTexture(...) (*see p. 434*) allows you to set the texture unit for which texturing commands are applied. This function is vital to multitexturing.
- void glClientActiveTexture(...) (*see p. 437*) is also useful for multitexturing. If you need it, you’ll know you need it, otherwise do not worry about it.