Appendix H

Programming Tips

This appendix lists some tips and guidelines that you might find useful. Keep in mind that these tips are based on the intentions of the designers of the OpenGL, not on any experience with actual applications and implementations! This appendix has the following major sections:

- **"OpenGL Correctness Tips"**
- **"OpenGL Performance Tips"**
- **"GLX Tips"**

OpenGL Correctness Tips

- Do not count on the error behavior of an OpenGL implementation—it might change in a future release of OpenGL. For example, OpenGL 1.0 ignores matrix operations invoked between *glBegin()* and *glEnd()* commands, but OpenGL 1.1 might not. Put another way, OpenGL error semantics may change between upward−compatible revisions.
- Use the projection matrix to collapse all geometry to a single plane. If the modelview matrix is used, OpenGL features that operate in eye coordinates (such as lighting and application−defined clipping planes) might fail.
- Do not make extensive changes to a single matrix. For example, do not animate a rotation by continually calling *glRotate()* with an incremental angle. Rather, use *glLoadIdentity()* to initialize the given matrix for each frame, then call *glRotate()* with the desired complete angle for that frame.
- Count on multiple passes through a rendering database to generate the same pixel fragments only if this behavior is guaranteed by the invariance rules established for a compliant OpenGL implementation. (See **Appendix I, "OpenGL Invariance"** for details on the invariance rules.) Otherwise, a different set of fragments might be generated.
- Do not expect errors to be reported while a display list is being defined. The commands within a display list generate errors only when the list is executed.
- Place the near frustum plane as far from the viewpoint as possible to optimize the operation of the depth buffer.
- Call *glFlush()* to force all previous OpenGL commands to be executed. Do not count on *glGet*()* or *glIs*()* to flush the rendering stream. Query commands flush as much of the stream as is required to return valid data but don't guarantee to complete all pending rendering commands.
- Turn dithering off when rendering predithered images (for example, when *glCopyPixels()* is called).
- Make use of the full range of the accumulation buffer. For example, if accumulating four images, scale each by one−quarter as it's accumulated.
- If exact two−dimensional rasterization is desired, you must carefully specify both the orthographic projection and the vertices of primitives that are to be rasterized. The orthographic projection should be specified with integer coordinates, as shown in the following example:

  ```
  gluOrtho2D(0, width, 0, height);
  ```

  where *width* and *height* are the dimensions of the viewport. Given this projection matrix, polygon vertices and pixel image positions should be placed at integer coordinates to rasterize predictably. For example, *glRecti(* 0, 0, 1, 1 *)* reliably fills the lower left pixel of the viewport, and *glRasterPos2i(* 0, 0*)* reliably positions an unzoomed image at the lower left of the viewport. Point vertices, line vertices, and bitmap positions should be placed at half−integer locations, however. For example, a line drawn from (*x1*, 0.5) to (*x2*, 0.5) will be reliably rendered along the bottom row of pixels int the viewport, and a point drawn at (0.5, 0.5) will reliably fill the same pixel as *glRecti(* 0, 0, 1, 1 *)*.

  An optimum compromise that allows all primitives to be specified at integer positions, while still ensuring predictable rasterization, is to translate *x* and *y* by 0.375, as shown in the following code fragment. Such a translation keeps polygon and pixel image edges safely away from the centers of pixels, while moving line vertices close enough to the pixel centers.

```
glViewport(0, 0, width, height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, width, 0, height);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.375, 0.375, 0.0);
/* render all primitives at integer positions */
```

- Avoid using negative *w* vertex coordinates and negative *q* texture coordinates. OpenGL might not clip such coordinates correctly and might make interpolation errors when shading primitives defined by such coordinates.

OpenGL Performance Tips

- Use *glColorMaterial()* when only a single material property is being varied rapidly (at each vertex, for example). Use *glMaterial()* for infrequent changes, or when more than a single material property is being varied rapidly.
- Use *glLoadIdentity()* to initialize a matrix, rather than loading your own copy of the identity matrix.
- Use specific matrix calls such as *glRotate*()* , *glTranslate*()* , and *glScale*()* , rather than composing your own rotation, translation, and scale matrices and calling *glMultMatrix()* .
- Use *glPushAttrib()* and *glPopAttrib()* to save and restore state values. Use query functions only when your application requires the state values for its own computations.
- Use display lists to encapsulate potentially expensive state changes. For example, place all the *glTexImage*()* calls required to completely specify a texture, and perhaps the associated *glTexParameter*()* , *glPixelStore*()* , and *glPixelTransfer*()* calls as well, into a single display list. Call this display list to select the texture.
- Use display lists to encapsulate the rendering calls of rigid objects that will be drawn repeatedly.
- Use evaluators even for simple surface tessellations to minimize network bandwidth in client−server environments.
- Provide unit−length normals if it’s possible to do so, and avoid the overhead of GL_NORMALIZE. Avoid using *glScale*()* when doing lighting because it almost always requies that GL_NORMALIZE be enabled.
- Set *glShadeModel()* to GL_FLAT if smooth shading isn’t required.
- Use a single *glClear()* call per frame if possible. Do not use *glClear()* to clear small subregions of the buffers; use it only for complete or near−complete clears.
- Use a single call to *glBegin(* GL_TRIANGLES *)* to draw multiple independent triangles, rather than calling *glBegin(* GL_TRIANGLES *)* multiple times, or calling *glBegin(* GL_POLYGON *)*. Even if only a single triangle is to be drawn, use GL_TRIANGLES rather than GL_POLYGON. Use a single call to *glBegin(* GL_QUADS *)* in the same manner, rather than calling *glBegin(* GL_POLYGON *)* repeatedly. Likewise, use a single call to *glBegin(* GL_LINES *)* to draw multiple independent line segments, rather than calling *glBegin(* GL_LINES *)* multiple times.
- In general, use the vector forms of commands to pass precomputed data, and use the scalar forms of commands to pass values that are computed near call time.
- Avoid making redundant mode changes, such as setting the color to the same value between each vertex of a flat−shaded polygon.
- Be sure to disable expensive rasterization and per−fragment operations when drawing or copying images. OpenGL will apply textures to pixel images if asked to!

GLX Tips

- Use *glXWaitGL()* rather than *glFinish()* to force X rendering commands to follow GL rendering commands.
- Likewise, use *glXWaitX()* rather than *glXSync()* to force GL rendering commands to follow X rendering commands.
- Be careful when using *glXChooseVisual()* because boolean selections are matched exactly. Since some implementations won't export visuals with all combinations of boolean capabilities, you should call *glXChooseVisual()* several times with different boolean values before you give up. For example, if no single−buffered visual with the required characteristics is available, check for a double−buffered visual with the same capabilities. It might be available, and it's easy to use.