Computer Graphics 200: OpenGL assignment

Delan Azabani

November 1, 2014

1 Design choices

Only features from OpenGL 1.4 and its fixed function pipeline were used to write this assignment. Although this API has long been deprecated, I chose it for a handful of reasons:

- OpenGL 1.4 is the only version supported by the lab machines, Xubuntu *and* Cygwin
- I found the syntax and semantics easier to learn than using shaders and buffer objects
- There is a large set of established tutorials and documentation for OpenGL 1.4

While the assignment lacks models relevant to the 'Australian outback' theme due to time constraints, every technical and interactive feature required by the specification has been met, and even exceeded in some areas.

2 System overview

The common namespace prefix `cg200_` will hereby be referred to as `@` for brevity.

The assignment's code ultimately starts at `@main`, which calls `@init` to configure a new window using GLUT. This window shall be double buffered for smooth rendering, support an alpha channel for transparency and fog, and contain a depth buffer to facilitate drawing of objects in a temporal order which is nearly unconstrained. The function also binds callbacks for events such as main loop iterations, key presses and window resizes.

`@start` turns on a large swath of features which are disabled by default for performance:

- `GL_COLOR_MATERIAL` allows objects to retain their coloured surfaces after illumination
- `GL_LIGHTING` allows this illumination instead of a simple, global, invariant light
- `GL_FOG` provides a simple global fog effect, which we configure to be exponential
- `GL_NORMALIZE` contributes to prevent lights from dimming when zooming in
- `GL_TEXTURE_2D` enables texturing which is used for the floor quadrilateral
- `GL_DEPTH_TEST` enables the use of the depth buffer to correctly draw overlapping objects
- `GL_BLEND` allows overlapping translucent fragments to blend with one another

Finally the GLUT main loop is started, and after an initial `@reshape` to configure the viewport, the process of rendering each frame occurs with a repeated sequence of calls from `@idle` to `glutPostRedisplay` to `@render`, which calls `@state_init` *once* to establish global state.

Every frame starts with `@state_tick`, which maintains accounting for frame counts, time in the animation and process 'worlds' and global scene rotation along three axes. The frame is cleared to a gaudy bright magenta to catch any accidental errors, then the camera is configured with a perspective projection, location and direction using `@gluLookAt` and zoom is applied.

`@render_body` is then called after the scene rotations, before swapping the frame buffers.

3 Objects and rendering

A simple night sky is cleared by `@render_sky`, then `@render_lights` creates the four lights positioned at the corners of the scene, while also establishing global fog. This fog intensifies as the scene becomes more distant by being inversely proportional to the zoom factor.

The floor of the scene is drawn with `@render_floor` and consists of a simple chequerboard texture which is dynamically generated and loaded once while rendering the first frame, then subsequently used repeatedly. A sequence of calls to `glMaterialfv` define the surface finishing for the floor, which reflects no ambient light, all diffuse light, and all specular light, while being slightly shiny with a small specular exponent.

The floor itself is not one quadrilateral, but 4096 smaller squares arranged such that lighting calculations shall be more accurate. This is because OpenGL calculates lighting for each vertex, and having more squares implies the creation of more vertices.

The oscillating prism is simply a collection of four triangles drawn in a single `GL_TRIANGLES` sequence. It takes advantage of smooth shading to yield faces that vaguely resemble colour gamut diagrams. The vertices that make up each triangle have been carefully ordered, not only to maintain a consistent anti-clockwise ordering for correct normals, but also to gracefully degrade with four distinct coloured faces when flat shading is enabled. This is achieved by having distinct colours for the *final* vertex in each triangle.

Motion for the prism is a combination of sinusoidal oscillation along the positive half of the z-axis, along with monotonic varying rotations around all three axial directions. Each axis has a rotation taking the form

$$120^\circ \times \left(t + \sin\left(t + \frac{2k\pi}{3}\right)\right)$$

where t is `STATE.time_animation` and k is an integer between zero and two inclusive, arbitrarily assigned to the axis of rotation, and intended to misalign the times when the rotations about each axis slow down. Because the derivative of $t + \sin(t)$ is $1 + \cos(t)$ and varies between zero and two inclusive, the function is monotonic and the rotation of the prism shall never reverse.

Transparency is satisfied by `@render_square`, which merely draws an elevated quadrilateral with translucent black. The aforementioned prism repeatedly cuts through this quadrilateral, demonstrating the ability of translucent objects to affect the surrounding colours.

`@render_balls` is the most interesting function in the rendering sequence. Using a sinusoidal, radial oscillation on the x-y-plane together with a linear rotation around the x-axis allows the CMYK coloured balls to meet one another in pairs. Level of detail in the sphere models is achieved with the supporting `@calculate_true_sphere_quality` function, which quadratically increases the number of faces as the user zooms closer to the scene.

To obtain full marks, key bindings and statistics are printed inside the OpenGL window instead of `stdout`. A primitive form of in-band signalling is used to allow rasterised text to consist of multiple fonts. Whenever `@draw_text` encounters a sequence matching `\a\d+;` where `\a` is the ASCII BEL, the font is changed, with the integer before the semicolon indexing `@fonts`.

4 Exceeding the specification

The following interactive features surpass what is required by the assignment specification:

- `[uU]` toggles clockwise scene rotation around the z-axis,
- `[lL]` increases/decreases the base level of detail for the spheres,
- `[+-]` increases/decreases the field of view from the default 90° angle,
- `[rR]` resets all global state, including the entire scene, and
- `[\n]` switches the assignment to fullscreen mode.

5 Interesting facts

- While depth buffering *mostly* frees developers from having to sort their objects prior to rendering, some restrictions still apply. Lights must be established before other objects that shall be affected by them, and opaque objects must be drawn before translucent objects that they may intersect when projected.
- Two techniques must be combined to ensure that lights do not erroneously change intensity while zooming with `glScalef` — `GL_NORMALIZE` must be enabled, as well as a `GL_LINEAR_ATTENUATION` which is inversely proportional to the zoom factor.
- There doesn't appear to be a stack for material parameters, so care must be taken to restore the surface finish values to their defaults to avoid clobbering the rendering of other objects.
- To attain correct colours for rasterised text, lighting and fog must be disabled prior to calling `glutBitmapCharacter` and subsequently enabled afterwards.
- Depth testing must also be disabled temporarily while drawing text, or some objects may obscure parts of the text in extreme cases of model locations and zoom.
- Logical XOR can easily be achieved with `!=` even though C doesn't have a `^^` operator.

6 References

- Clark, R. (2012) *OpenGL:Tutorial Framework:Light and Fog.* Retrieved from http://content.gpwiki.org/index.php/OpenGL:Tutorials:Tutorial_Framework
- Eusebeîa. (2014) *The Tetrahedron.* Retrieved from http://eusebeia.dyndns.org/4d/tetrahedron
- Gamedev.net. (2012) *NeHe Productions - Everything OpenGL.* Retrieved from http://nehe.gamedev.net/
- Hennig, M. (2011) *opengl — Setting glutBitmapCharacter color? — Stack Overflow.* Retrieved from http://stackoverflow.com/a/8239654
- Khronos Group. (2002) *The relationship between glScalef() and light intensity.* Retrieved from https://www.opengl.org/discussion_boards/showthread.php/154816
- Khronos Group. (2001) *Transparency, Translucency, and Blending.* Retrieved from https://www.opengl.org/archives/resources/faq/technical/transparency.htm
- Kilgard, M. J. (2000) *Avoiding 16 Common OpenGL Pitfalls.* Retrieved from http://www.opengl.org/archives/resources/features/KilgardTechniques/oglpitfall/
- Kilgard, M. J. (1996) *glutSolidCube, glutWireCube.* Retrieved from https://www.opengl.org/documentation/specs/glut/spec3/node82.html
- Kilgard, M. J. (1996) *glutSolidSphere, glutWireSphere.* Retrieved from https://www.opengl.org/resources/libraries/glut/spec3/node81.html
- Microsoft Corporation. (2012) *glMaterialfv function.* Retrieved from http://msdn.microsoft.com/en-us/library/windows/desktop/dd373945.aspx
- Silicon Graphics, Inc. (2006) *glColorMaterial.* Retrieved from https://www.opengl.org/sdk/docs/man2/xhtml/glColorMaterial.xml
- Silicon Graphics, Inc. (2006) *glLight.* Retrieved from https://www.opengl.org/sdk/docs/man2/xhtml/glLight.xml
- Silicon Graphics, Inc. (2006) *glRotate.* Retrieved from https://www.opengl.org/sdk/docs/man2/xhtml/glRotate.xml
- Silicon Graphics, Inc. (2006) *glShadeModel.* Retrieved from https://www.opengl.org/sdk/docs/man2/xhtml/glShadeModel.xml
- Urquhart, D. (2010) *OpenGL 2 Tutorials — Swiftless Tutorials.* Retrieved from http://www.swiftless.com/opengltuts.html