Introduction to OpenGL

Wesley Griffin

University of Maryland, Baltimore County

September 30, 2010

Overview

- Graphics Hardware APIs
- OpenGL Overview
- Specifics
- Demo

Graphics Hardware APIs

- RenderMan and raytracing are software-based methods for image generation
- Most computers today have a piece of hardware designed to accelerate image generation
- Using the graphics processing unit (GPU) requires using a new programming interface
- DirectX is a Microsoft API available only on Windows
- OpenGL is a cross-platform API and is used for this class
- Knowledge of one API is highly portable to the other API

OpenGL Overview

- You can use C or C++ with OpenGL
- OpenGL is a low-level hardware API
- Does not provide support for windows, user interaction, etc.
- GLUT is a simple cross-platform API that provides windows and user interaction
- For advanced projects, GLUT might be too simple, but for this class its perfect
- Programs hand execution control over to GLUT which uses callbacks when the program must do some work
- Sample program to illustrate basics of OpenGL and GLUT runs on Mac, Linux, and Windows

Immediate Mode

- Immediate mode is deprecated functionality in latest OpenGL specifications
- Most tutorials and examples on the web still use immediate mode
- Commands such as glVertex3f, glColor3f, glNormal3f are immediate mode commands
- The sample program uses data arrays which add some complexity but are more suited to modern hardware
- Caveat: data arrays (glVertexPoint, glColorPointer, glNormalPointer) are *also* deprecated in the latest specification
- Latest specifications are more complex to program for because they closely match the latest hardware
- Data arrays are an intermediate step away from old functionality

Demo

Header Includes

main.cpp

```cpp
#include <cstdlib>

#if defined(__APPLE__) || defined(MACOSX)
#include <GLUT/glut.h>
#elif defined(_WIN32)
#include "glut.h"
#else
#include <GL/glut.h>
#endif

#include "userinput.h"
#include "drawcube.h"

static int g_WindowWidth = 1024, g_WindowHeight = 768;
static int g_WindowTopLeftX = 100, g_WindowTopLeftY = 100;
static char const* g_WindowTitle = "OpenGL Intro";

// Forward declarations for callback functions.
extern "C" void idle();
extern "C" void reshape(int width, int height);
extern "C" void display();

void initGLUT(int *argc, char *argv[]);
void initGL();
```

Main

main.cpp

```
void initGL();

int main(int argc, char *argv[]) {
    initGLUT(&argc, argv);
    initGL();

    // Start the GLUT processing loop.
    glutMainLoop();

    return EXIT_SUCCESS;
}
```

GLUT Initialization

main.cpp

```cpp
void initGLUT(int *argc, char *argv[]) {
    glutInit(argc, argv);

    // We want a full color, depth buffer, and double-buffered context.
    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);

    // Set the initial size and position for the window.
    glutInitWindowSize(g_WindowWidth, g_WindowHeight);
    glutInitWindowPosition(g_WindowTopLeftX, g_WindowTopLeftY);

    // Creating the window creates the OpenGL context.
    glutCreateWindow(g_WindowTitle);

    // Set the callback functions, must be set after creating the window.
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(motion);

    glutIdleFunc(idle);
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
}
```

OpenGL Initialization

main.cpp

```
void initGL() {
    // Set the color to clear the screen to.
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    // Enable depth testing.
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glClearDepth(1.0);

    // Enable face culling.
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);
}
```

Support Functions

main.cpp

```cpp
// Request a redraw anytime the loop is idle.
void idle() {
    glutPostRedisplay();
}

// Handle window reshape events.
void reshape(int width, int height) {
    // Reset the viewport.
    glViewport(0, 0, width, height);

    // Set up the projection matrix.
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluPerspective(45.0, width / static_cast<GLdouble>(height), 0.1, 100.0);
}
```

Rendering

main.cpp

```cpp
// Perform all rendering.
void display() {
    // Clear the color and depth buffers.
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


    // Set up the model view matrix.
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glTranslated(g_PositionX, g_PositionY, g_PositionZ);
    glRotated(g_AngleX, 1.0, 0.0, 0.0);
    glRotated(g_AngleY, 0.0, 1.0, 0.0);
    glRotated(g_AngleZ, 0.0, 0.0, 1.0);

    // Draw a cube.
    drawCube();

    // Swap the front and back buffers.
    glutSwapBuffers();
}
```

Drawing the Cube

drawcube.cpp

```cpp
void drawCube() {
    // Specify the vertex, color, and normal data.
    glVertexPointer(3, GL_FLOAT, 0, g_Vertices);
    glEnableClientState(GL_VERTEX_ARRAY);

    glColorPointer(3, GL_FLOAT, 0, g_Colors);
    glEnableClientState(GL_COLOR_ARRAY);

    glNormalPointer(GL_FLOAT, 0, g_Normals);
    glEnableClientState(GL_NORMAL_ARRAY);


    // Draw the specified triangles.
    int triangles = sizeof(g_Indices) / sizeof(g_Indices[0]);
    glDrawElements(GL_TRIANGLES, triangles, GL_UNSIGNED_INT, g_Indices);


    // Reset what we turned on.
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisableClientState(GL_COLOR_ARRAY);
    glDisableClientState(GL_VERTEX_ARRAY);
}
```

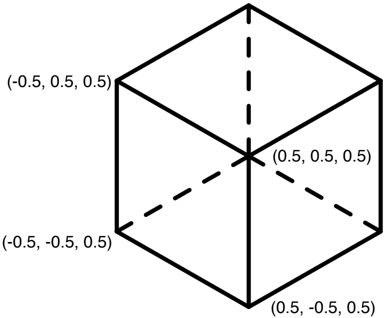
Specifying Geometry

- OpenGL supports a small set of primitive types
- The most common primitive is the triangle: GL_TRIANGLES
- Primitives are described using an array of vertices
- Indexed vertices are the most efficient way to draw primitives
- The array of vertices holds a single position for each vertex (i.e. there are no overlapping vertices)
- An array of *indices* indicates which vertices from the vertex array to use to create a primitive

Defining the Vertices

drawcube.cpp

```
// A set of vertices for a triangle.
static GLfloat g_Vertices[] = {
     0.500000f,  0.500000f,  0.500000f,
    -0.500000f,  0.500000f,  0.500000f,
     0.500000f, -0.500000f,  0.500000f,
    -0.500000f, -0.500000f,  0.500000f,
     0.500000f,  0.500000f, -0.500000f,
    -0.500000f,  0.500000f, -0.500000f,
     0.500000f, -0.500000f, -0.500000f,
    -0.500000f, -0.500000f, -0.500000f
};
```



Normals and Colors for the Vertices

drawcube.cpp

```
// A set of normals for the vertices.
static GLfloat g_Normals[] = {
     0.577350f,  0.577350f,  0.577350f,
    -0.333333f,  0.666667f,  0.666667f,
     0.666667f, -0.333333f,  0.666667f,
    -0.666667f, -0.666667f,  0.333333f,
     0.666667f,  0.666667f, -0.333333f,
    -0.666667f,  0.333333f, -0.666667f,
     0.333333f, -0.666667f, -0.666667f,
    -0.577350f, -0.577350f, -0.577350f
};

// A set of vertex colors for the vertices.
static GLfloat g_Colors[] = {
    0.0f, 0.0f, 0.0f,
    0.0f, 0.0f, 1.0f,
    0.0f, 1.0f, 0.0f,
    0.0f, 1.0f, 1.0f,
    1.0f, 0.0f, 0.0f,
    1.0f, 0.0f, 1.0f,
    1.0f, 1.0f, 0.0f,
    1.0f, 1.0f, 1.0f
};
```

Triangle Indices

Inside void drawcube():

```
    // Draw the specified triangles.
    int triangles = sizeof(g_Indices) / sizeof(g_Indices[0]);
    glDrawElements(GL_TRIANGLES, triangles, GL_UNSIGNED_INT, g_Indices);
```

drawcube.cpp

```
// A set of indices for 4 triangles to form a cube.
static GLuint g_Indices[] = {
    0, 1, 2,
    3, 2, 1,
    0, 2, 4,
    6, 4, 2,
    0, 4, 1,
    5, 1, 4,
    7, 5, 6,
    4, 6, 5,
    7, 6, 3,
    2, 3, 6,
    7, 3, 5,
    1, 5, 3
};
```