COMP3811: Computer Graphics
Tutorial 1: Practical
Block 1: First Experiments with OpenGL

Marc de Kamps

September 25, 2020

Objective

In this tutorial, we'll do a *Hello World* style version of a program. The main objective is to check whether your set up works, whether this is on `feng-linux`, or on your own setup.

Access to School Machine OpenGL Setup

- Open the `Chrome` browser.
- Paste the following url: `https://feng-linux.leeds.ac.uk/gpu/` into your browser.
- A login screen should appear: give name, then password.
- In your browser window a Linux Desktop should appear. It will look as if you had logged on to one of the School Desktops.
- You can run a browser in this Desktop. Go on `Minerva` and download the source code for tutorial 1: `tutorial1.tar.gz`.
- Open a terminal. Any commands below need to be entered in the terminal window.
- Type: `module add qt`. Without this you will not be able to use Qt. As this gets annoying after a while, add this command to your `.bashrc` file, or in its equivalent for whatever shell you're using.
- Untar `tutorial1.tar.gz` in a directory of your choice.
- Enter the `cube_construct` directory.
- Type: `qmake HelloCube.pro`.
- Type: `make`
- There should be an executable textttHelloCube. Run it: `./HelloCube`.

Setting up a Shape

You are hopefully familiar with the basic structure of a `Qt` program: a main window containing one or more widgets. In this case a `SolidCubeWidget`, which is in the inheritance chain of both `QWidget` and `QGLWidget`. The following construction makes the `OpenGL` API available in the source files that include the `QGLWidget` header:

```
#include <QGLWidget>

class SolidCubeWidget: public QGLWidget
```

For the purpose of rendering, the following methods are important:

```
// called when OpenGL context is set up

  void SolidCubeWidget::initializeGL()
  {   // initializeGL()
      // set the widget background colour
      glClearColor(0.3, 0.3, 0.3, 0.0);

      // This create a view volume around the origin that extends to coordinate 4 in every direction.
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-4.0, 4.0, -4.0, 4.0, -4.0, 4.0);


      // You must set the matrix mode to model view directly before enabling the depth test
      glMatrixMode(GL_MODELVIEW);
      //       glEnable(GL_DEPTH_TEST); // comment out depth test to observe the result

  } // initializeGL()
```

`OpenGL` needs some initialization code, which is best placed in the `initializeGL()` method. A camera volume is set up as an orthographic projection. There is also a statement to enable a Z buffer test, or depth buffer test, which for now has been commented out.

```
// called every time the widget needs painting

void SolidCubeWidget::paintGL()
{ // paintGL()
  // clear the widget
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      this->cube();

      glLoadIdentity();
      gluLookAt(0.,0.,2., 0.0,0.0,0.0, 0.0,1.0,0.0);

      // flush to screen
      glFlush();

} // paintGL()
```

OpenGL is (mainly) a state machine, meaning that at the moment of rendering all graphical primitives that need to rendered need to be defined. Rendering requires the use of buffers, that will be discussed in greater detail in the lectures. When OpenGL renders, colour information is written into a colour buffer. Depth buffering will change what you see on the screen. Try to guess what this is doing.

All buffers in use need to be cleared before rendering which is achieved `glClear`. The rendering of the cube will be discussed below. A camera is positioned at $(0, 0, 2)$, directed to the point $(0, 0, 0)$, with an up direction in the positive y-direction. The reason that this statement is in `paintGL` rather than in the initialization is that camera positions can be animated, i.e. shift from frame to frame.

```
void SolidCubeWidget::cube(){

  glColor3f(1.0,0.0,0.0);
  glBegin(GL_POLYGON);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f( 1.0, -1.0, -1.0);
    glVertex3f( 1.0,  1.0, -1.0);
    glVertex3f(-1.0,  1.0, -1.0);
  glEnd();

  glColor3f(0.0,0.0,1.0);
  glBegin(GL_POLYGON);
    glVertex3f( 1.0, -1.0,  1.0);
    glVertex3f( 1.0, -1.0, -1.0);
```

```
    glVertex3f( 1.0,  1.0, -1.0);
    glVertex3f( 1.0,  1.0,  1.0);
    glEnd();


  glColor3f(0.0,1.0,0.0);
  glBegin(GL_POLYGON);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f( 1.0, -1.0, 1.0);
    glVertex3f( 1.0,  1.0, 1.0);
    glVertex3f(-1.0,  1.0, 1.0);
  glEnd();
}
```

- Verify that these definitions create a world model like that of Figure 1.
- Compile and run the program. Observe the output
- Exchange the order in which the different polygons are created. Compile, run. Does the result matter?
- Now uncomment `glEnable(GL_DEPTH_TEST);` and repeat the experiment. If necessary discuss the result with the demonstrator.
- Find the online documentation of `gluLookAt`
- Move the camera to position (1.,1.,1.). Again experiment with `glEnable(GL_DEPTH_TEST)`.
- The use of the word 'camera' is slightly misleading. Enable the depth test put the camera at $(0, 0, 0)$ and let the view direction be $(0, 0, 1)$. Do you see the red or the green plane? Explain the result.
- Complete the cube.

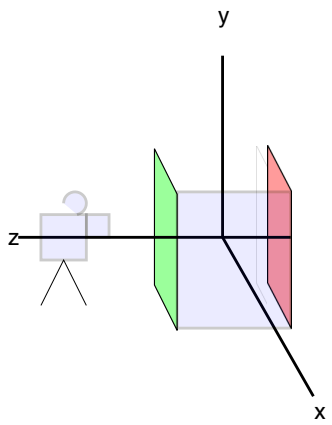

Figure 1: A partially built cube.