Introduction to OpenGL 1

1.1 What is OpenGL?

OpenGL is a *graphics rendering* API whose version 1.0 was released by Silicon Graphics Inc. in July 1994, and currently it is maintained by the Khronos Group (`www.khronos.org`). It provides the support for generating and displaying high-quality color images composed of geometric and image primitives. It is windowing system and operating system independent.

Although the above mentioned independence it needs some basic support from them. Different platforms have specific implementations. These implementations are known as WGL, AGL and GLX, respectively for Windows, Mac OS X and unix-based systems. There are a number of libraries that provide extra functionalities:

- GLU (OpenGL Utility Library) which is part of OpenGL provides among other things quadric shapes.
- GLUT (OpenGL Utility Toolkit), is not an official part of OpenGL, provides a portable windowing API.
- SDL (Simple Direct Media Layer), provides support for 2D graphics, OpenGL, windowing, input devices, etc.

In the following you will explore three different ways of develop OpenGL programs: using SDL, GLUT libraries in C/C++ and using Python. To support them you must install the necessary libraries. For Ubuntu you may use the following commands.

```
sudo apt install python-pip freeglut3-dev libsdl2-dev
pip install PyOpenGL PyOpenGL_accelerate
```

1.2 OpenGL with SDL

The following code is a very basic skeleton of an OpenGL program using SDL to communicate with the windowing system.

```
#include <stdio.h>
#include <stdint.h>
#include <assert.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_opengl.h>

#ifdef __APPLE__
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#else
#include <GL/gl.h>
#include <GL/glu.h>
#endif

typedef int32_t i32;
typedef uint32_t u32;
typedef int32_t b32;

#define WinWidth 640
#define WinHeight 480
SDL_Window *Window;

void InitGL(int Width, int Height) {
    glClearColor( 0.0, 0.0, 0.0, 1.0 );
    glClearDepth( 1.0 );
    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

void ReSizeGLScene(int Width, int Height){
    glViewport(0, 0, Width, Height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

void DrawGLScene( void ) {
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();

/* YOUR CODE HERE */

  SDL_GL_SwapWindow(Window);
}

int main (int ArgCount, char **Args) {

    u32 WindowFlags = SDL_WINDOW_RESIZABLE|SDL_WINDOW_OPENGL;
    Window = SDL_CreateWindow("CG&RA SDL", 0, 0,
                                        WinWidth, WinHeight, WindowFlags);
```

```
    SDL_GLContext Context = SDL_GL_CreateContext(Window);

    b32 Running = 1;
    b32 FullScreen = 0;

    InitGL(WinWidth, WinHeight);

    while (Running) { // Main loop
        SDL_Event ev;
        while (SDL_PollEvent(&ev)) {
            switch(ev.type) {
            case SDL_QUIT:
                Running=false;
                break;
            case SDL_WINDOWEVENT:
                switch (ev.window.event) {
                case SDL_WINDOWEVENT_RESIZED:
                    SDL_Log("Window %d resized to %dx%d",
                            ev.window.windowID, ev.window.data1,
                            ev.window.data2);
                    ReSizeGLScene(ev.window.data1, ev.window.data2);
                    break;
                } break;

            case SDL_KEYDOWN:
                switch (ev.key.keysym.sym){
                case  SDLK_ESCAPE:
                case SDLK_q:
                    exit(0);
                case SDLK_f:
                    FullScreen = !FullScreen;
                    if (FullScreen) {
                        SDL_SetWindowFullscreen(Window, WindowFlags |
                                SDL_WINDOW_FULLSCREEN_DESKTOP);
                    }
                    else {
                        SDL_SetWindowFullscreen(Window, WindowFlags);
                    }
                    break;
                }
            }
        }

        DrawGLScene();
    } // Main loop
    return 0;
}
```

The event processing switch clauses can be extended, as there are many different events that are supported by SDL. Hereafter follows an example piece of code that could be used to extend the events supported by the application.

```
...
    case SDL_WINDOWEVENT:
        switch (ev.window.event) {
        case SDL_WINDOWEVENT_SHOWN:
          SDL_Log("Window %d shown", ev.window.windowID);
          break;
        case SDL_WINDOWEVENT_HIDDEN:
```

```
            SDL_Log("Window␣%d␣hidden", ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_EXPOSED:
            SDL_Log("Window␣%d␣exposed", ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_MOVED:
            SDL_Log("Window␣%d␣moved␣to␣%d,%d",
                    ev.window.windowID, ev.window.data1,
                    ev.window.data2);
            break;
          case SDL_WINDOWEVENT_RESIZED:
            SDL_Log("Window␣%d␣resized␣to␣%dx%d",
                    ev.window.windowID, ev.window.data1,
                    ev.window.data2);
            ReSizeGLScene(ev.window.data1, ev.window.data2);
            break;
          case SDL_WINDOWEVENT_MINIMIZED:
            SDL_Log("Window␣%d␣minimized", ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_MAXIMIZED:
            SDL_Log("Window␣%d␣maximized", ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_RESTORED:
            SDL_Log("Window␣%d␣restored", ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_ENTER:
            SDL_Log("Mouse␣entered␣window␣%d",
                    ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_LEAVE:
            SDL_Log("Mouse␣left␣window␣%d", ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_FOCUS_GAINED:
            SDL_Log("Window␣%d␣gained␣keyboard␣focus",
                    ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_FOCUS_LOST:
            SDL_Log("Window␣%d␣lost␣keyboard␣focus",
                    ev.window.windowID);
            break;
          case SDL_WINDOWEVENT_CLOSE:
            SDL_Log("Window␣%d␣closed", ev.window.windowID);
            break;
          default:
            SDL_Log("Window␣%d␣got␣unknown␣event␣%d",
                    ev.window.windowID, ev.window.event);
            break;
          } break;

        case SDL_MOUSEBUTTONDOWN:
          switch (ev.button.button){
          case SDL_BUTTON_LEFT:
            SDL_Log("Left␣mouse␣button␣down");
            break;
          case SDL_BUTTON_RIGHT:
            break;
          }
          break;

        case SDL_MOUSEBUTTONUP:
          switch (ev.button.button){
          case SDL_BUTTON_LEFT:
```

```
            SDL_Log("Left mouse button up");
            break;
          case SDL_BUTTON_RIGHT:
            break;
          }
          break;

        case SDL_MOUSEMOTION:
          break;
...
```

The above code can be compiled on Ubuntu with

```
g++ opengl_sdl.cpp -lSDL2 -lGLU -lGL
```

On Windows with msvc

```
cl sdl2_opengl.cpp /I C:\sdl2path\include /link C:\path\SDL2.lib
C:\path\SDL2main.lib /SUBSYSTEM:CONSOLE
/NODEFAULTLIB:libcmtd.lib opengl32.lib
```

and on Mac OS X with gcc and SDL2 installed via macports

```
gcc -I/opt/local/include sdl2_opengl.cpp -L/opt/local/lib -lSDL2 \
-framework OpenGL
```

1.3 GLUT

The same can be implemented using GLUT as follows

```
#ifdef __APPLE__
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#else
#include <GL/gl.h>
#include <GL/glu.h>
#endif

#include <GL/glut.h>
#include <stdio.h>

void InitGL(int Width, int Height) {
  glClearColor( 0.0, 0.0, 0.0, 1.0 );
  glClearDepth( 1.0 );
  glDepthFunc(GL_LESS);
  glEnable(GL_DEPTH_TEST);
  glShadeModel(GL_SMOOTH);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0);
  glMatrixMode(GL_MODELVIEW);
}

void ReSizeGLScene(int Width, int Height){
  glViewport(0, 0, Width, Height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
```

```
  gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0);
  glMatrixMode(GL_MODELVIEW);
}

void DrawGLScene( void ) {
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  /* YOUR CODE HERE */

  glutSwapBuffers();
}
void keyPressed( unsigned char key, int x, int y ) {
  switch( key ) {
  case 27:
  case 'q' :
  case 'Q' :
    exit( EXIT_SUCCESS );
    break;
  }
}

void idle( void ) {
  glutPostRedisplay();
}

int main( int argc, char** argv ) {
  glutInit(&argc, argv);

  glutInitDisplayMode( GLUT_RGBA|GLUT_DOUBLE|GLUT_DEPTH );
  glutInitWindowSize(640,480);
  glutInitWindowPosition(0,0);
  glutCreateWindow("CG&RA GLUT");
  glutDisplayFunc( DrawGLScene );  // register callback routines
  glutReshapeFunc( ReSizeGLScene );
  glutKeyboardFunc( keyPressed );
  glutIdleFunc( idle );
  InitGL(640,480); // initiate OpenGL states, program variables
  glutMainLoop(); // enter the event-driven loop
}
```

The above code can be compiled on Ubuntu with

```
g++ opengl_glut.cpp -lGLUT -lGLU -lGL
```

For better dealing with the build-edit process it is advised to write a Makefile as follows, and then use the make command on a terminal.

```
all: opengl_sdl opengl_glut

opengl_sdl: opengl_sdl.cpp
	g++ opengl_sdl.cpp -lSDL2 -lGLU -lGL -o opengl_sdl

opengl_glut: opengl_glut.cpp
	g++ -o opengl_glut opengl_glut.cpp -lGLU -lglut -lGL

clean:
	rm -r opengl_sdl opengl_glut *~
```

1.4 Python with PyOpenGL

PyOpenGL is available as a module installable via pip command and requires GLUT or other supported windowing library to be installed.

For other operating systems check http://pyopengl.sourceforge.net/ website.

The following code is similar to the previous versions in C, but this time in python using GLUT.

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import sys

ESCAPE = '\033'
window = 0

def InitGL(Width, Height):
    glClearColor(0.0, 0.0, 0.0, 0.0)
    glClearDepth(1.0)
    glDepthFunc(GL_LESS)
    glEnable(GL_DEPTH_TEST)
    glShadeModel(GL_SMOOTH)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0)
    glMatrixMode(GL_MODELVIEW)

def ReSizeGLScene(Width, Height):
    glViewport(0, 0, Width, Height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45.0, float(Width)/float(Height), 0.1, 100.0)
    glMatrixMode(GL_MODELVIEW)

def DrawGLScene():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    glTranslatef(0.0, 0.0, -6.0)
    # YOUR CODE HERE

    glutSwapBuffers()

def keyPressed(*args):
    global window
    if args[0] == ESCAPE:
        sys.exit()

def idle():
     glutPostRedisplay()

def main():
    global window
    glutInit(sys.argv)

    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
    glutInitWindowSize(640, 480)
    glutInitWindowPosition(0, 0)
    window = glutCreateWindow("CG&RA Python")
```

```
    glutDisplayFunc(DrawGLScene)

    glutIdleFunc(DrawGLScene)
    glutReshapeFunc(ReSizeGLScene)
    glutKeyboardFunc(keyPressed)
    InitGL(640, 480)
    glutMainLoop()

# Print message to console, and kick off the main to get it rolling.
print "Hit ESC key to quit."
main()
```

It can be executed by typing on the terminal

```
python opengl_py.py
```

1.5 Exercises

Exercise 1.1
Write the 3 versions of code in 3 separate files. Write a Makefile to compile the C versions. Execute make command and execute the 3 versions... Analyse the 3 versions of code their execution results and add comments that explain each piece of code.

Exercise 1.2
Add to the drawing function of the above files the following code.

```
        glBegin(GL_POLYGON)
        glVertex3f(0.0, 1.0, 0.0)
        glVertex3f(1.0, -1.0, 0.0)
        glVertex3f(-1.0, -1.0, 0.0)
        glEnd()


        glBegin(GL_QUADS)
        glVertex3f(-1.0, 1.0, 0.0)
        glVertex3f(1.0, 1.0, 0.0)
        glVertex3f(1.0, -1.0, 0.0)
        glVertex3f(-1.0, -1.0, 0.0)
        glEnd()
```

Compile and run the 3 versions. Add comments to these lines.

Exercise 1.3
Add the following lines to the previous code... one before each vertex of the polygon

```
glColor3f(1.0, 0.0, 0.0)
...
```

```
glColor3f(0.0, 1.0, 0.0)
...
 glColor3f(0.0, 0.0, 1.0)
 ...
```

Add the following line before the fist vertex of the QUAD

```
        glTranslatef(-1.5, 0.0, -6.0)
```

Compile, run and comment the code.

Exercise 1.4

Use the glRotate(angle,ax,ay,az) to animate the previous figures. Note that angle is the value of the angle in degrees and the remaining parameters are the components of the vector that serves as rotation axis.