

OpenGL

What is OpenGL?

OpenGL is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering.

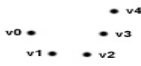
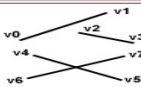
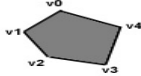

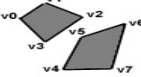
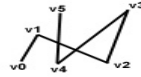
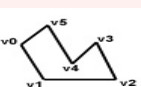
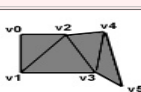
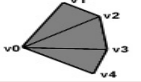
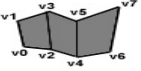
Introduction

- [Silicon Graphics, Inc.](#) (SGI) began developing OpenGL in 1991 and released it on June 30, 1992.
- OpenGL is a hardware-independent, operating system independent API.
- The well-specified OpenGL standard has language bindings for C, C++, Fortran, Ada, and Java.
- OpenGL does not provide direct support for complex geometrical shapes, such as cubes or spheres. These must be built up from supported primitives.

Vertices and Primitives

Most objects (with the exception of pixel rectangles and bitmaps), use Begin/End primitives. Each Begin/End primitive contains a series of vertex data, and may optionally contain normals, texture coordinates, colors, edge flags, and material properties.

There are ten primitive types, as follows:

Points	individual points	
Lines	pairs of vertices interpreted as individual line segments	
Polygon	boundary of a simple, convex polygon	
Triangles	triples of vertices interpreted as triangles	
Quads	quadruples of vertices interpreted as four-sided polygons	
Line Strip	series of connected line segments	
Line Loop	same as above, with a segment added between last and first vertices	
Triangle Strip	linked strip of triangles	
Triangle Fan	linked fan of triangles	
Quad Strip	linked strip of quadrilaterals	

Some features of OpenGL

- Geometric and raster primitives
- RGBA or color index mode
- Display list or immediate mode
- Viewing and modeling transformations
- Lighting and Shading
- Hidden surface removal (Depth Buffer)
- Alpha Blending
- Anti-aliasing
- Texture Mapping
- Atmospheric Effects (Fog, Smoke, Haze)
- Feedback and Selection
- Stencil Planes
- Accumulation Buffer
- Depth Cueing
- Wireframes
- Motion blur

Programming OpenGL in C/C++

We need the following sets of libraries in programming OpenGL:

1. **Core OpenGL (GL):** consists of hundreds of functions, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives, such as point, line, and polygon.
2. **OpenGL Utility Library (GLU):** built on-top of the core OpenGL to provide important utilities and more building models (such as quadric surfaces). GLU functions start with a prefix "glu" (e.g., gluLookAt, gluPerspective)
3. **OpenGL Utilities Toolkit (GLUT):** provides support to interact with the Operating System (such as creating a window, handling key and mouse inputs); and more building models (such as sphere and torus). GLUT functions start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc).
Quoting from the [opengl.org](https://www.opengl.org): "GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. GLUT is simple, easy, and small."
4. **OpenGL Extension Wrangler Library (GLEW):** "GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform."

OpenGL Overview

- ❖ Most Widely Adopted Graphics Standard
- ❖ High Visual Quality and Performance
- ❖ Developer-Driven Advantages:
 - Industry standard
 - Stable
 - Reliable and portable
 - Evolving
 - Scalable
 - Easy to use
 - Well-documented
- ❖ Simplifies Software Development, Speeds Time-to-Market
- ❖ Available Everywhere

Where is it used?

- Video games
- CAD applications such AutoCAD, Blender
- Virtual reality
- Scientific visualization
- Information visualization
- Flight simulation

How to install OpenGL in Linux?

Open a terminal and execute the following commands

`sudo apt-get update` *// To get information on the newest version of package and their dependencies*

`sudo apt-get install freeglut3` *// Provides simple windowing API and I/O operations*

`sudo apt-get install freeglut3-dev`
// header files for freeglut3

`sudo apt-get install binutils-gold` *// A linker for ELF files. Faster than GNU Linker.*

`sudo apt-get install g++ cmake` *// Software tool for managing the build process of software*

`sudo apt-get install libglew-dev` *// For determining which OpenGL extensions are supported on the platform*

`sudo apt-get install g++` *// GNU C++ compiler*

`sudo apt-get install mesa-common-dev` *// mesa is an OpenGL compatible 3D graphics library*

`sudo apt-get install build-essential` *// All the packages needed to compile a debian package*

`sudo apt-get install libglew1.5-dev libglm-dev` *// (glm) C++ mathematical library for graphics program.*

Check the installation

```
roshin@pop-os:~$ glxinfo | grep "OpenGL version"  
OpenGL version string: 4.6 (Compatibility Profile) Mesa 20.0.8
```

Sample code

```
1  #include <GL/freeglut.h>
2  #include <GL/gl.h>
3
4  void renderFunction()
5  {
6      glClearColor(0.0, 0.0, 0.0, 0.0);
7      glClear(GL_COLOR_BUFFER_BIT);
8      glColor3f(1.0, 1.0, 1.0);
9      glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
10
11     glBegin(GL_POLYGON);
12         glVertex2f(-0.5, -0.5);
13         glVertex2f(-0.5, 0.5);
14         glVertex2f(0.5, 0.5);
15         glVertex2f(0.5, -0.5);
16     glEnd();
17
18     glFlush();
19 }
20 int main(int argc, char** argv)
21 {
22     glutInit(&argc, argv);
23     glutInitDisplayMode(GLUT_SINGLE);
24     glutInitWindowSize(500,500);
25     glutInitWindowPosition(100,100);
26     glutCreateWindow("OpenGL - First window demo");
27     glutDisplayFunc(renderFunction);
28     glutMainLoop();
29     return 0;
30 }
```

Let's see what these commands are used for

Initializations

glutInit

```
void glutInit(int *argcp, char **argv);
```

glutInit will initialize the GLUT library and negotiate a session with the window system.

glutInit also processes command line options.

Ex:-

-display *DISPLAY*

-geometry *W x H + X + Y*

glutInitDisplayMode

```
void glutInitDisplayMode(unsigned int mode);
```

This can be used to select the features we would want a window to have. It can be the color system we are using, the frame buffers needed etc.

Ex:- `glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE)`

glutInitWindowSize

```
void glutInitWindowSize(int width, int height)
```

The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size. The window system is not obligated to use this information.

glutInitWindowPosition

```
void glutInitWindowPosition(int x, int y);
```

glutInitWindowSize set the initial window position

glutCreateWindow

```
void glutCreateWindow(char *name);
```

The parameter will be used to set the window name.

glutCreateWindow creates a top-level window.

glutDisplayFunc

```
void glutDisplayFunc(void (*func)(void))
```

glutDisplayFunc sets the display callback for the *current window*.

This display callback function (the painter) gets called whenever your window must be redrawn

- When its size changes
- When it becomes visible
- When some parts of it become visible
- When it is moved

glutMainLoop

```
void glutMainLoop(void);
```

glutMainLoop enters the GLUT event processing loop.

Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

Drawing

glClearColor

glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range 0 - 1 .

Syntax

```
void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
```

Usage

```
glClearColor(0.0, 0.0, 0.0, 0.0);
```

glClear

glClear sets the bitplane area of the window to values previously selected by glClearColor, glClearDepthf, and glClearStencil.

Syntax

```
void glClear(GLbitfield mask);
```

mask

Bitwise OR of masks that indicate the buffers to be cleared. The three masks are GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, and GL_STENCIL_BUFFER_BIT.

Usage

```
glClear(GL_COLOR_BUFFER_BIT);
```

glColor3f

Sets the current color of the pen

Syntax

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
```

Usage

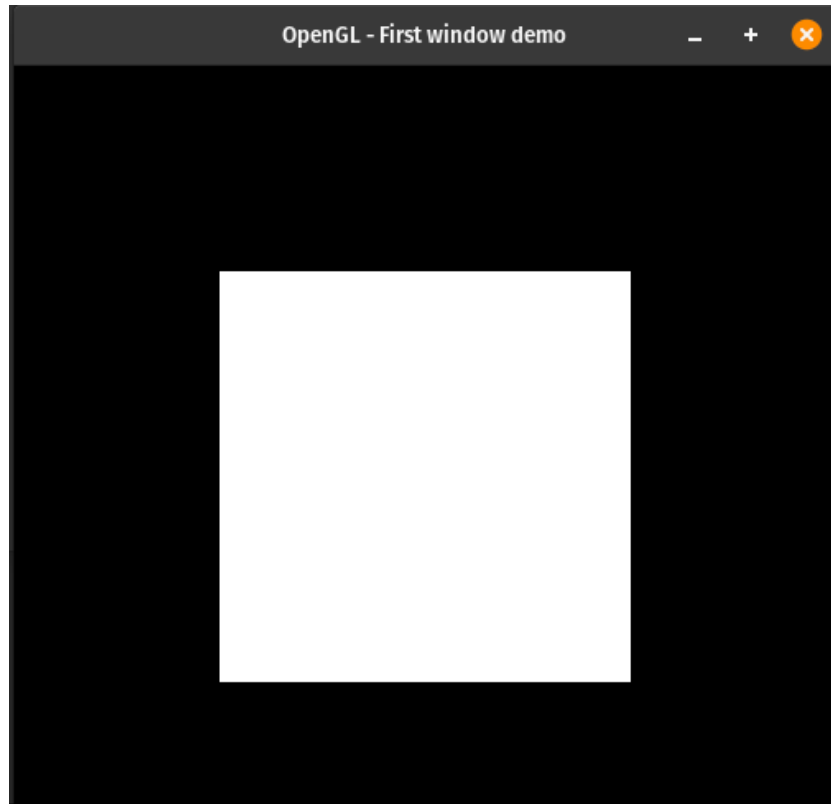
```
glColor3f(1.0, 1.0, 1.0);
```

glBegin, glEnd

glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted.

Example

```
glBegin(GL_POLYGON);  
    glVertex2f(-0.5, -0.5);  
  
    glVertex2f(-0.5, 0.5);  
  
    glVertex2f(0.5, 0.5);  
  
    glVertex2f(0.5, -0.5);  
  
glEnd();
```



Some of the other interpretations possible

GL_POINTS

GL_LINES

GL_LINE_STRIP

GL_LINE_LOOP

GL_QUAD_STRIP

GL_TRIANGLES

GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN

GL_QUADS

GL_POLYGON

glOrtho

It describes the field of vision of the camera specified by a cuboid of certain dimensions and the type of projection that we use to project points onto the screen plane(here it is orthogonal projection).

glOrtho

Syntax

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);
```

Usage

```
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

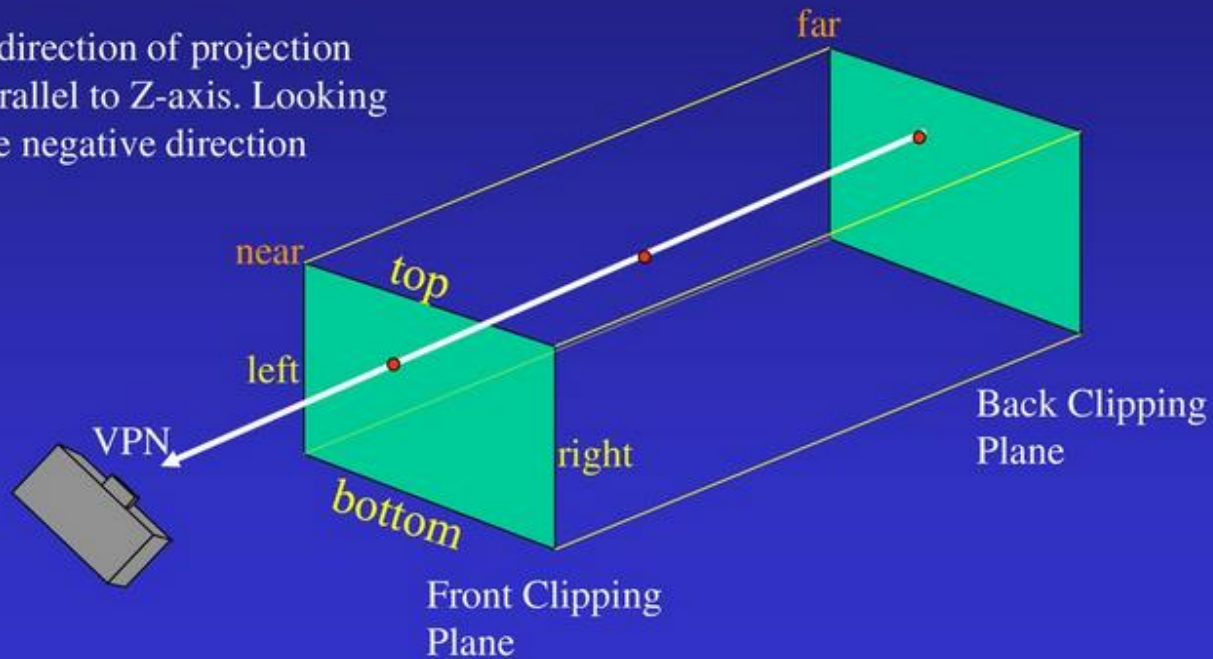
Please check the link to understand more

<https://www.youtube.com/watch?v=V87lLvKscIY>

In OpenGL

`glOrtho`(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

The direction of projection is parallel to Z-axis. Looking at the negative direction



glFlush

```
void glFlush(void);
```

Forces execution of GL commands in finite time.

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. `glFlush` empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine.

Compilation

```
g++ main.c -lglut -lGL -lGLEW -lGLU -o OpenGLExample
```

```
/* -lglut: Link with glut
```

```
-lGL: Link with GL
```

```
-lGLEW: Link with GLEW
```

```
-lGLU: Link with GLU */
```

Run the executable file

```
./OpenGLExample
```

Demo 1 - Plotting some points read from a file

```
#include <GL/freeglut.h>
#include <GL/gl.h>
#include <stdio.h>

struct Point
{
    GLfloat x,y;
};

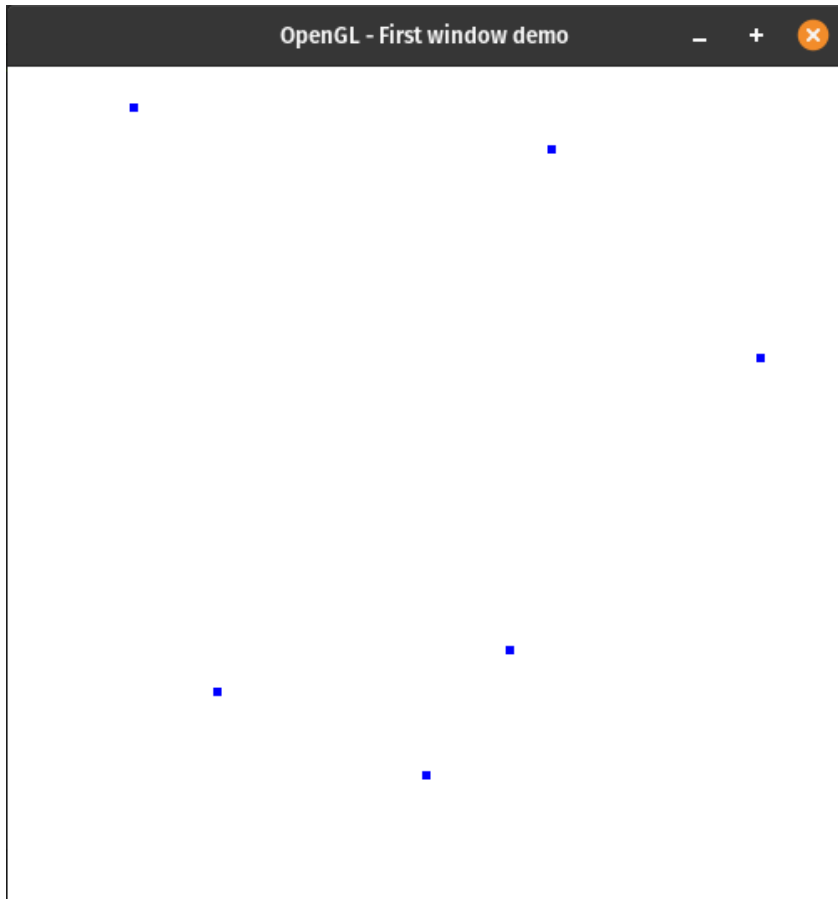
struct Point p[100000];
int count = 0;

void readInput()
{
    FILE *fptr = fopen("input.txt", "r");
    if(fptr)
    {
        while(fscanf(fptr, "%f %f", &(p[count].x), &(p[count].y))!=EOF)
        {
            count++;
        }
        fclose(fptr);
    }
}
```

```
void renderFunction()
{
    int i=0;
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 1.0);
    glPointSize(5);
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

    glBegin(GL_POINTS);
    while(i < count)
    {
        glVertex2f(p[i].x, p[i].y);
        i++;
    }
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    readInput();
    glutCreateWindow("OpenGL - First window demo");
    glutDisplayFunc(renderFunction);
    glutMainLoop();
    return 0;
}
```



Event handling demo in OpenGL

In the following program we try to clear the screen using a color depending on the key we pressed on the keyboard.

To do this we need to register a callback corresponding to the keypress event in the initialization part

```
//Registering a keyboard event callback.  
//MyKeyboardFunc will now get called on any key board button press.  
glutKeyboardFunc(MyKeyboardFunc);
```

We also need to define what MyKeyboardFunc does

Check out the following slides for the entire program

```
#include <GL/freeglut.h>
#include <GL/gl.h>
#include <stdio.h>

GLfloat r=0.0,g = 0.0,b = 0.0,a =0.0;

void renderFunction()
{
    int i=0;
    glClearColor(r, g, b, a);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 1.0);
    glPointSize(5);
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

    glFlush();
}
```

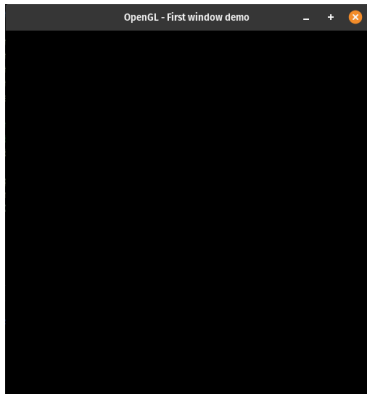
```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("OpenGL - First window demo");

    //Registering a keyboard event callback.
    //MyKeyboardFunc will now get called on any key board button press.
    glutKeyboardFunc(MyKeyboardFunc);

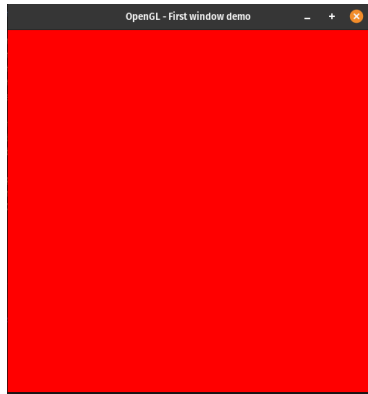
    glutDisplayFunc(renderFunction);
    glutMainLoop();
    return 0;
}
```

```
void MyKeyboardFunc(unsigned char Key, int x, int y)
{
    //Key contains the character we pressed
    //x and y contains the coordinates of the button cursor when
    //the keyboard key was pressed
    switch(Key)
    {
        case 'r':
            r = 1.0;g = 0.0;b = 0.0;a = 1.0;
            glClearColor(r, g, b, a);
            glClear(GL_COLOR_BUFFER_BIT);
            glFlush();
            break;
        case 'g':
            r = 0.0;g = 1.0;b = 0.0;a = 1.0;
            glClearColor(r, g, b, a);
            glClear(GL_COLOR_BUFFER_BIT);
            glFlush();
            break;
        case 'y':
            r = 1.0;g = 1.0;b = 0.0;a = 1.0;
            glClearColor(r, g, b, a);
            glClear(GL_COLOR_BUFFER_BIT);
            glFlush();
            break;
        default: break;
    };
}
```

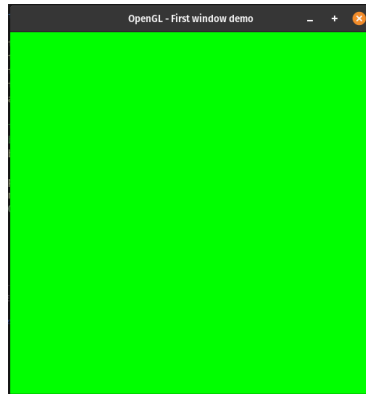
Output



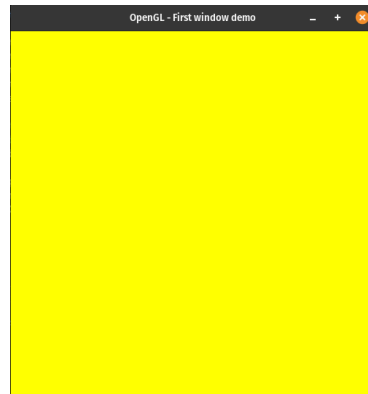
Initially



Pressed r



Pressed g



Pressed y

References

- [1]. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwi4ydrC7LHsAhU94HMBHViVDxcQFjADegQIBRAC&url=https%3A%2F%2Fwww.cs.utexas.edu%2Fusers%2Ffussell%2Fcourses%2Fcs354%2Fhandouts%2FAddison.Wesley.OpenGL.Programming.Guide.8th.Edition.Mar.2013.ISBN.0321773039.pdf&usg=AOvVaw0eSL-A754ij_wV_sq03JvS
- [2]. *OpenGL Architecture Review Board, OpenGL Reference Manual: The Official Reference Document for OpenGL, Release 1, Addison-Wesley, Reading, Massachusetts, 1992 (ISBN 0-201-63276-4).*
- [3]. <http://www.cs.toronto.edu/~kyros/courses/418/Notes/Visibility.pdf>
- [4]. <https://www.youtube.com/watch?v=pQcC2CqReSA> <https://flylib.com/books/en/2.789.1.32/1/>
- [5]. <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-1-opening-a-window/>

Thank you