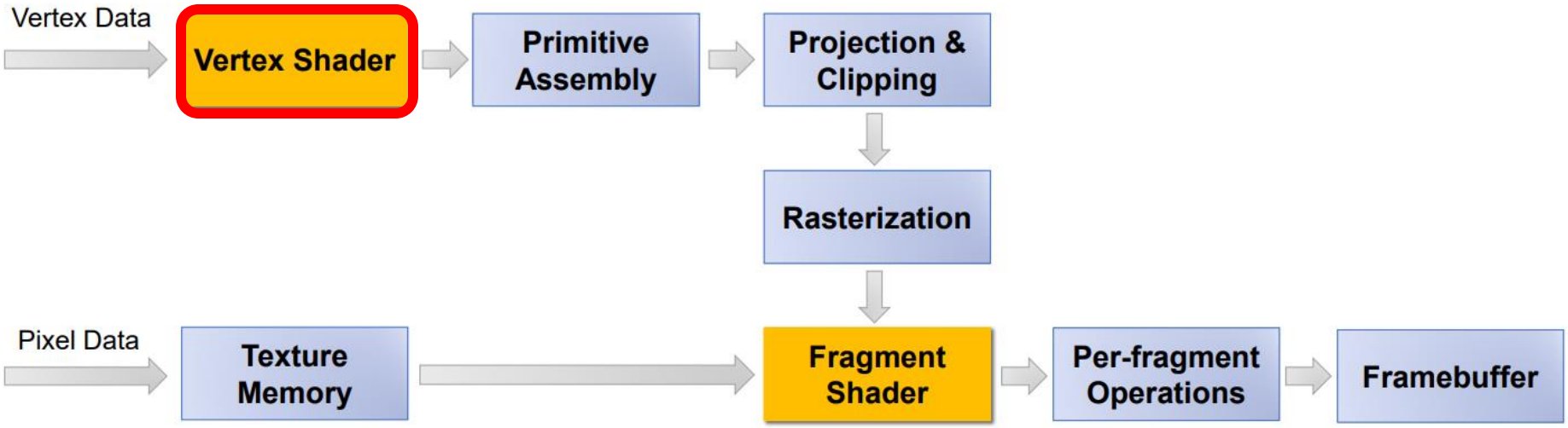
Computer Graphics
- Introduction to OpenGL

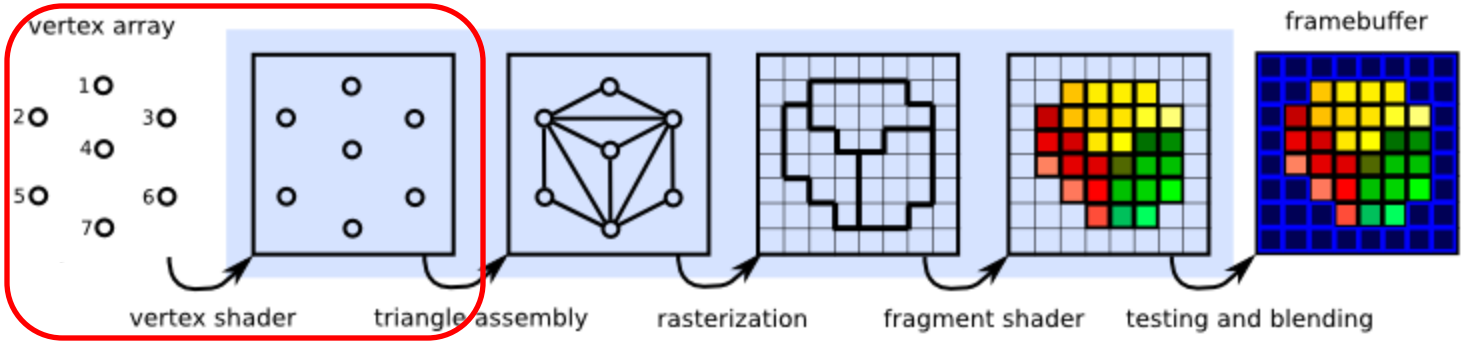
Sung Soo Hwang

What is OpenGL

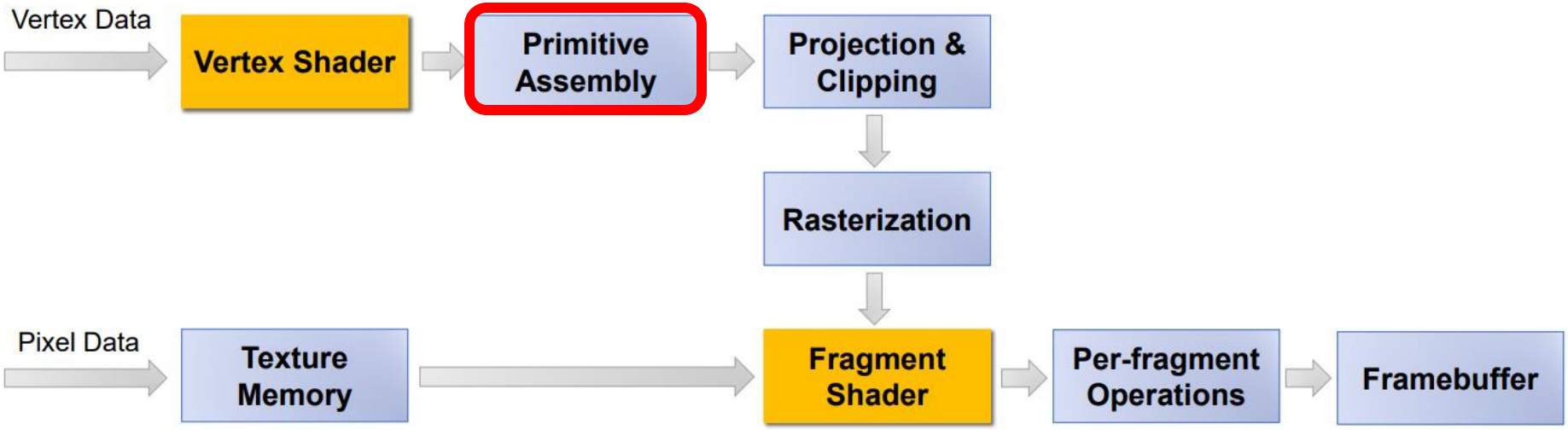
- OpenGL is A low-level graphics library specification
- OpenGL makes available to the programmer a small set of geometric primitives(points, lines, polygons, images, and bitmaps)
- OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn).

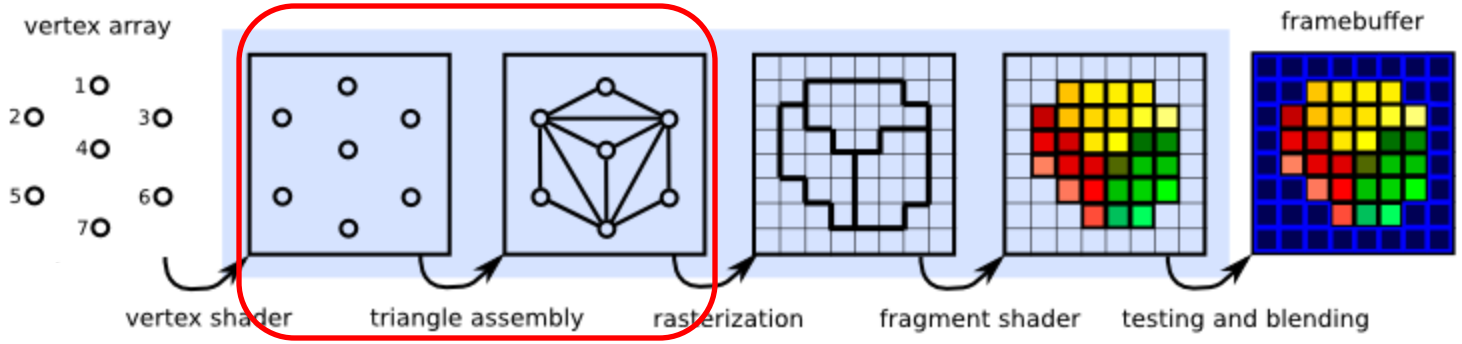
OpenGL's Rendering Pipeline



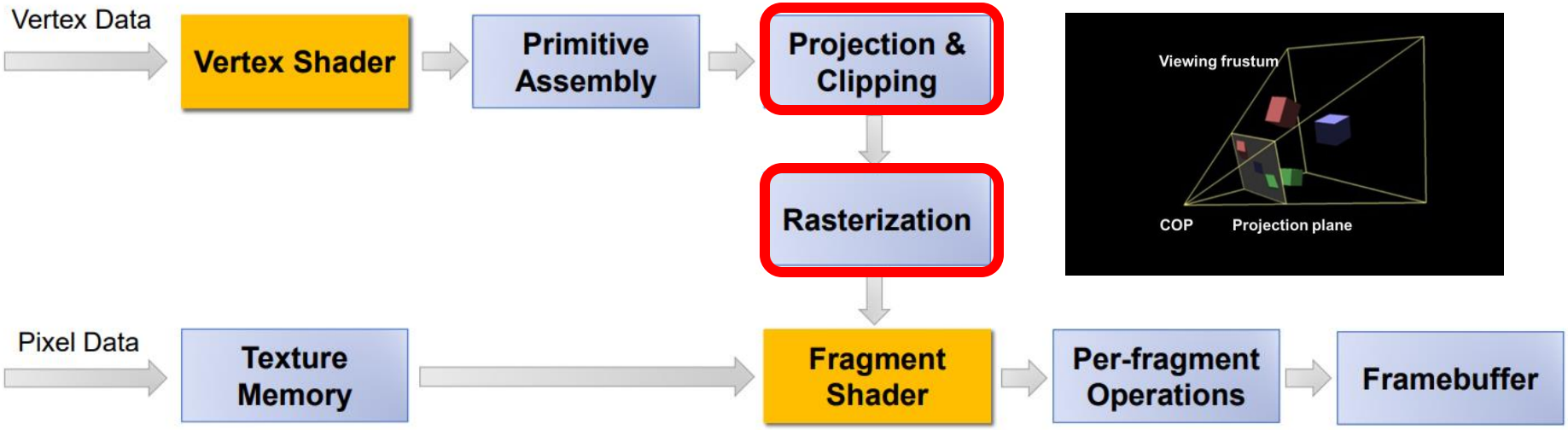


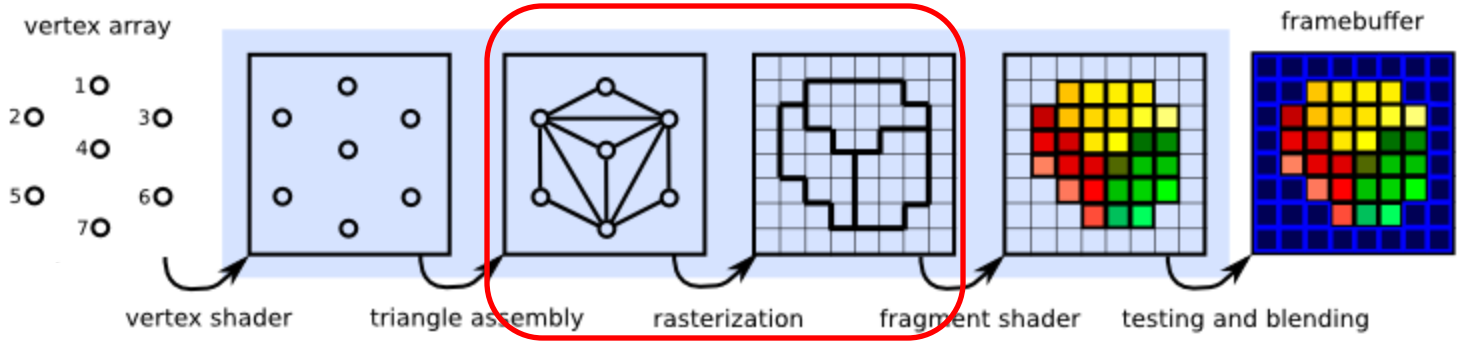
OpenGL's Rendering Pipeline



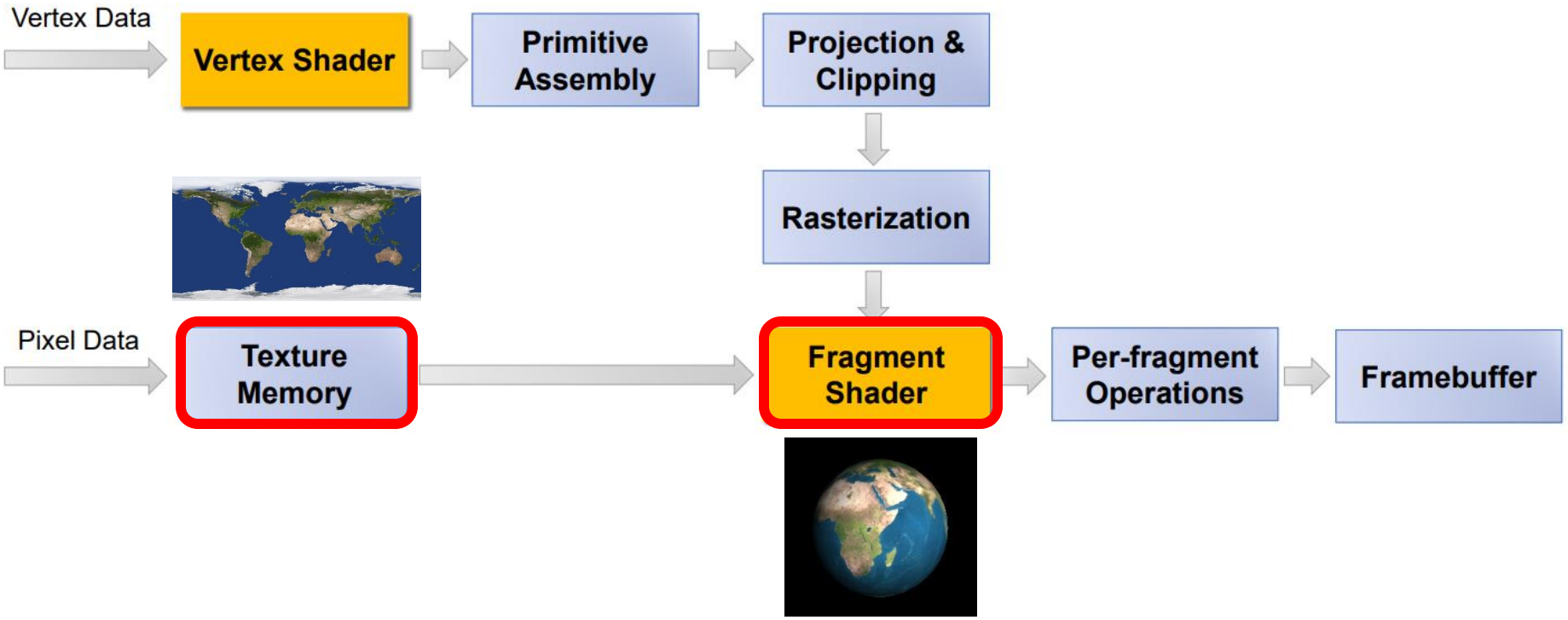


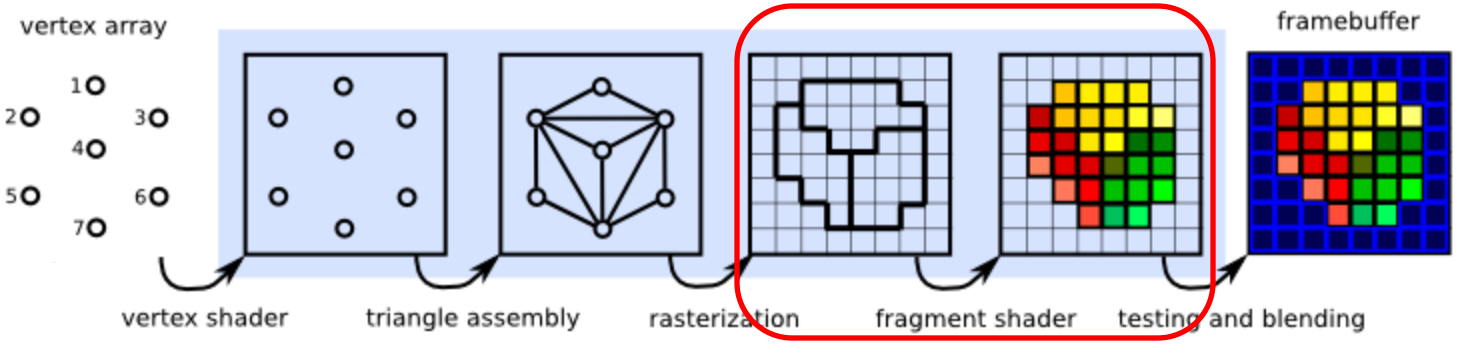
OpenGL's Rendering Pipeline



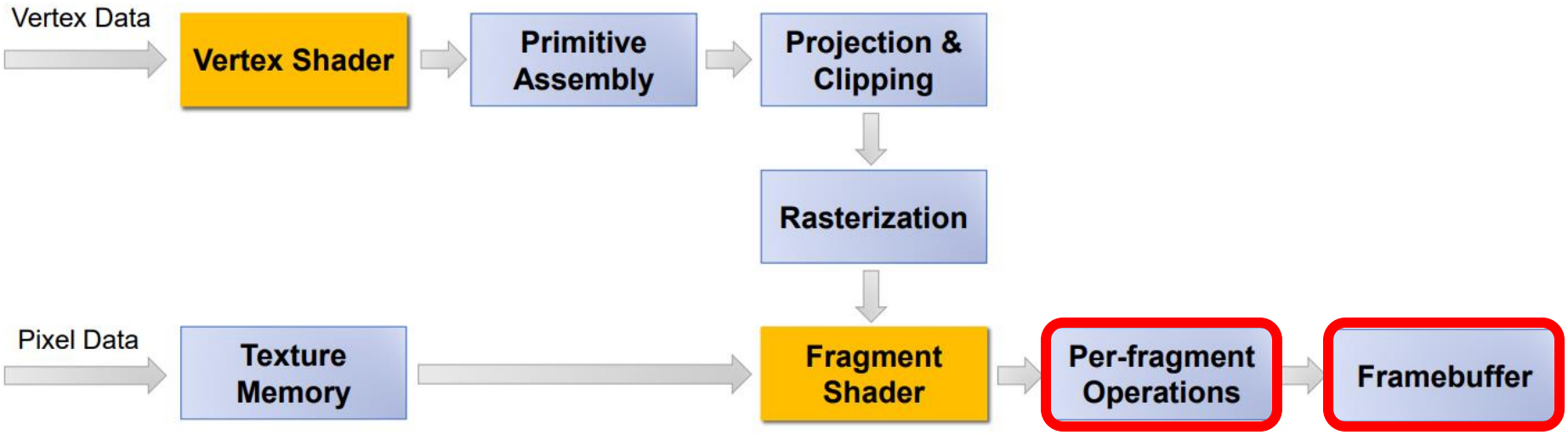


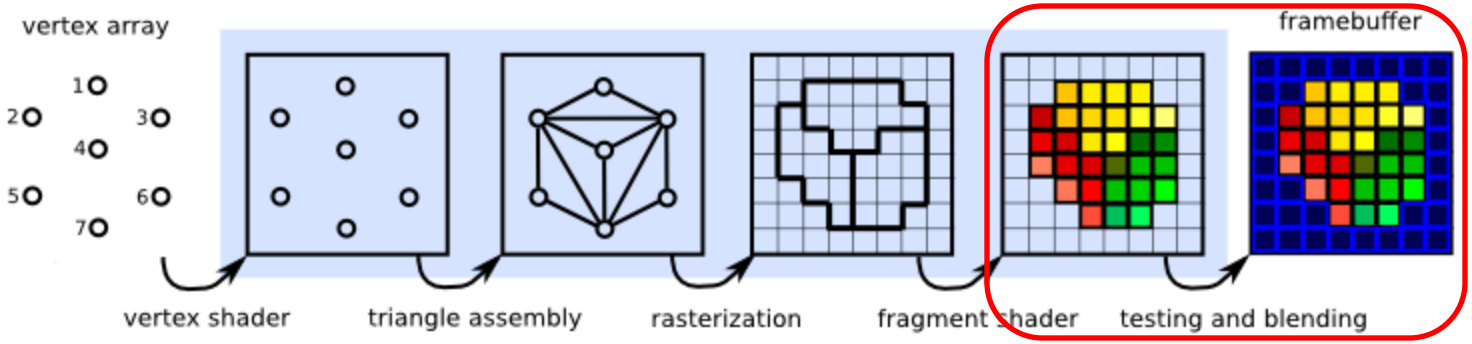
OpenGL's Rendering Pipeline





OpenGL's Rendering Pipeline





For more details, visit https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

OpenGL Syntax

- Constants
  - Begin with “GL” ex) GL_COLOR_BUFFER_BIT
- Functions
  - Begin with “gl” ex) glClearColor()
- OpenGL has no overloaded functions.

Data types

Type name	Description	Type constant
GLboolean	Boolean value, either GL_TRUE or GL_FALSE	
GLbyte	8-bit integer value	GL_BYTE
GLshort	16-bit integer value	GL_SHORT
GLint	32-bit integer value	GL_INT
GLuint	32-bit unsigned integer value	GL_UNSIGNED_INT
GLfloat	32-bit floating-point value	GL_FLOAT

For more details : https://www.khronos.org/opengl/wiki/OpenGL_Type

Function suffix

- Provides information about the arguments passed to the function.
- Typically defined in the following pattern:

FunctionName{1234}{b s i i64 ...}{v}

the # of parameters — Data type — vector

Suffix	OpenGL Type
b	GLbyte
s	GLshort
i	GLint
i64	GLint64
ub	GLubyte
us	GLushort
ui	GLuint
ui64	GLuint64
f	GLfloat
d	GLdouble

Function suffix

- Ex) glUniform*()

glUniform1i	Single integer value passed into the function: Void glUniform1i(GLint location, GLint v0);
glUniform2f	Two floating-point values passed into the function: void glUniform2f(GLintlocation, GLfloatv0, GLfloatv1);
glUniform3fv	A pointer to three floating-point values passed into the function: void glUniform3fv(GLintlocation,GLsizeicount, constGLfloat*value);

OpenGL Shading Language(GLSL)

- Specially designed for writing the code for shaders
- Very similar to C programming language

- Example

```
#version 330 core
layout (location = 0) in vec3 aPos; // the position variable has attribute position 0

out vec4 vertexColor; // specify a color output to the fragment shader

void main()
{
    gl_Position = vec4(aPos, 1.0); // see how we directly give a vec3 to vec4's constructor
    vertexColor = vec4(0.5, 0.0, 0.0, 1.0); // set the output variable to a dark-red color
}
```

Auxiliary Libraries

- To support window interfaces
  - GLUT
  - FreeGLUT

- To load OpenGL extensions
  - GLEW

GLUT(OpenGL Utility Toolkit)

- Introduction
  - A library for OpenGL apps that provides a simple windowing API
  - Well-suited to learning OpenGL and developing simple OpenGL applications

- Limitations
  - The most recent version is too old dating back to August 1998.
  - The license does not allow anyone to distribute modified library code.

FreeGLUT

- Introduction
  - An alternative to the GLUT library released under the X-Consortium license (which puts only very limited restriction on reuse)

- Download
  - Main website: http://freeglut.sourceforge.net/
  - Windows binaries
    - Website: http://www.transmissionzero.co.uk/software/freeglut-devel/
    - Download the binaries for MSVC (Microsoft Visual C++) if you intend to use MSVC for compiling your OpenGL applications.

GLEW (OpenGL Extension Wrangler)

- Introduction
  - A cross-platform open-source OpenGL extension loading library
  - Provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform.

- Download
  - Website: http://glew.sourceforge.net/
  - Windows binaries are also provided:
    - https://sourceforge.net/projects/glew/files/glew/2.1.0/glew-2.1.0-win32.zip/download

Basic Structure of OpenGL Program

```
void MyDisplay() { };
void MyKeyboard(char key, int x, int y) { };
void MyMouse(int button, int state, int x, int y) { };
```

callback function definition

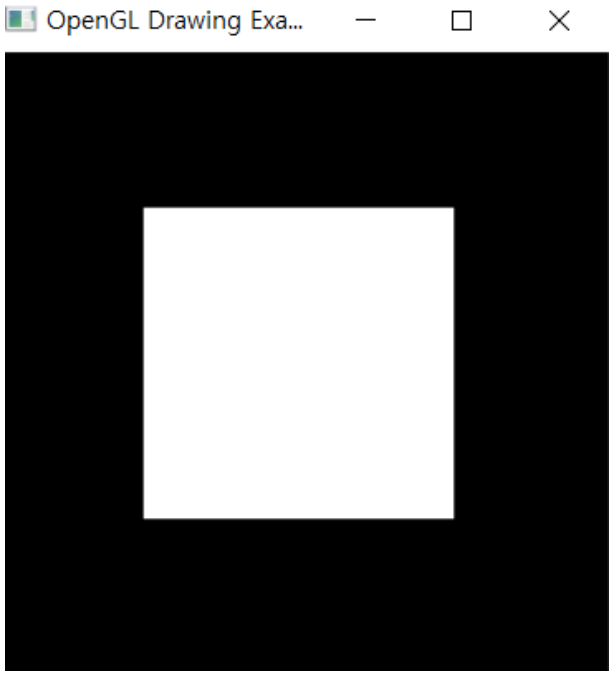
```
int main()
{
        Initialize and Open Window;
        Initialize OpenGL State;
        Register Input Callback Functions;
        {
                glutDisplayFunc(MyDisplay);
                glutKeyboardFunc(MyKeyboard);
                glutMouseFunc(MyMouse);
        }
        Enter Event Processing Loop;
}
```

Simple Code

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

void MyDisplay() {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex3f(-0.5, -0.5, 0.0);
                glVertex3f(0.5, -0.5, 0.0);
                glVertex3f(0.5, 0.5, 0.0);
                glVertex3f(-0.5, 0.5, 0.0);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow("OpenGL Drawing Example");
        glutDisplayFunc(MyDisplay);
        glutMainLoop();
        return 0;
}
```



Result

Code Review

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
```

These header files are for using the GLUT, GL, and GLU library functions respectively.

```
void MyDisplay() {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex3f(-0.5, -0.5, 0.0);
                glVertex3f(0.5, -0.5, 0.0);
                glVertex3f(0.5, 0.5, 0.0);
                glVertex3f(-0.5, 0.5, 0.0);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow("OpenGL Drawing Example");
        glutDisplayFunc(MyDisplay);
        glutMainLoop();
        return 0;
}
```

Code Review

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

void MyDisplay() {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex3f(-0.5, -0.5, 0.0);
                glVertex3f(0.5, -0.5, 0.0);
                glVertex3f(0.5, 0.5, 0.0);
                glVertex3f(-0.5, 0.5, 0.0);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow("OpenGL Drawing Example");
        glutDisplayFunc(MyDisplay);
        glutMainLoop();
        return 0;
}
```

The polygon we want to draw in is defined between glBegin and glEnd.

The polygon is composed of vertices input by glVertex3f(), and these are based on the Default Window which is the vertex coordinates of the center of the window are (0.0, 0.0, 0.0) and the upper right corner of the window as (1.0, 1.0, 1.0).

Code Review

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

void MyDisplay() {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex3f(-0.5, -0.5, 0.0);
                glVertex3f(0.5, -0.5, 0.0);
                glVertex3f(0.5, 0.5, 0.0);
                glVertex3f(-0.5, 0.5, 0.0);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow("OpenGL Drawing Example");
        glutDisplayFunc(MyDisplay);
        glutMainLoop();
        return 0;
}
```

glFlush() is an instruction that forces all instructions accumulated so far to be delivered to the processor unconditionally.

This is called when all functions for rendering are defined. If this command is not executed, the vertex is only defined and not displayed on the window.

Code Review

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

void MyDisplay() {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex3f(-0.5, -0.5, 0.0);
                glVertex3f(0.5, -0.5, 0.0);
                glVertex3f(0.5, 0.5, 0.0);
                glVertex3f(-0.5, 0.5, 0.0);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow("OpenGL Drawing Example");
        glutDisplayFunc(MyDisplay);
        glutMainLoop();
        return 0;
}
```

All main functions are composed of GLUT functions. glutCreateWindow() is a command to GLUT to create a new window, and the string in parentheses becomes the name of the window.

Code Review

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

void MyDisplay() {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex3f(-0.5, -0.5, 0.0);
                glVertex3f(0.5, -0.5, 0.0);
                glVertex3f(0.5, 0.5, 0.0);
                glVertex3f(-0.5, 0.5, 0.0);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow("OpenGL Drawing Example");
        glutDisplayFunc(MyDisplay);
        glutMainLoop();
        return 0;
}
```

glutDisplayFunc registers a function called "MyDisplay" as a callback function for display events.

The end of the main function always ends with glutMainLoop(). This is because we need to run the event loop.

Code Review

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

void MyDisplay() {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex3f(-0.5, -0.5, 0.0);
                glVertex3f(0.5, -0.5, 0.0);
                glVertex3f(0.5, 0.5, 0.0);
                glVertex3f(-0.5, 0.5, 0.0);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow("OpenGL Drawing Example");
        glutDisplayFunc(MyDisplay);
        glutMainLoop();
        return 0;
}
```

0 is returned because the return type of the main function is declared as an integer.

Window and Viewport of GLUT

- Window: The screen to be drawn using OpenGL
- Viewport: A polygonal (normally rectangular) area in the Window that is currently being viewed.

Commands	Description
glViewport(x, y, width, height);	Set the size of viewport
glutInitWindowSize(width, height);	Set the size of window
glutInitWindowPosition(x, y);	Set the position of window

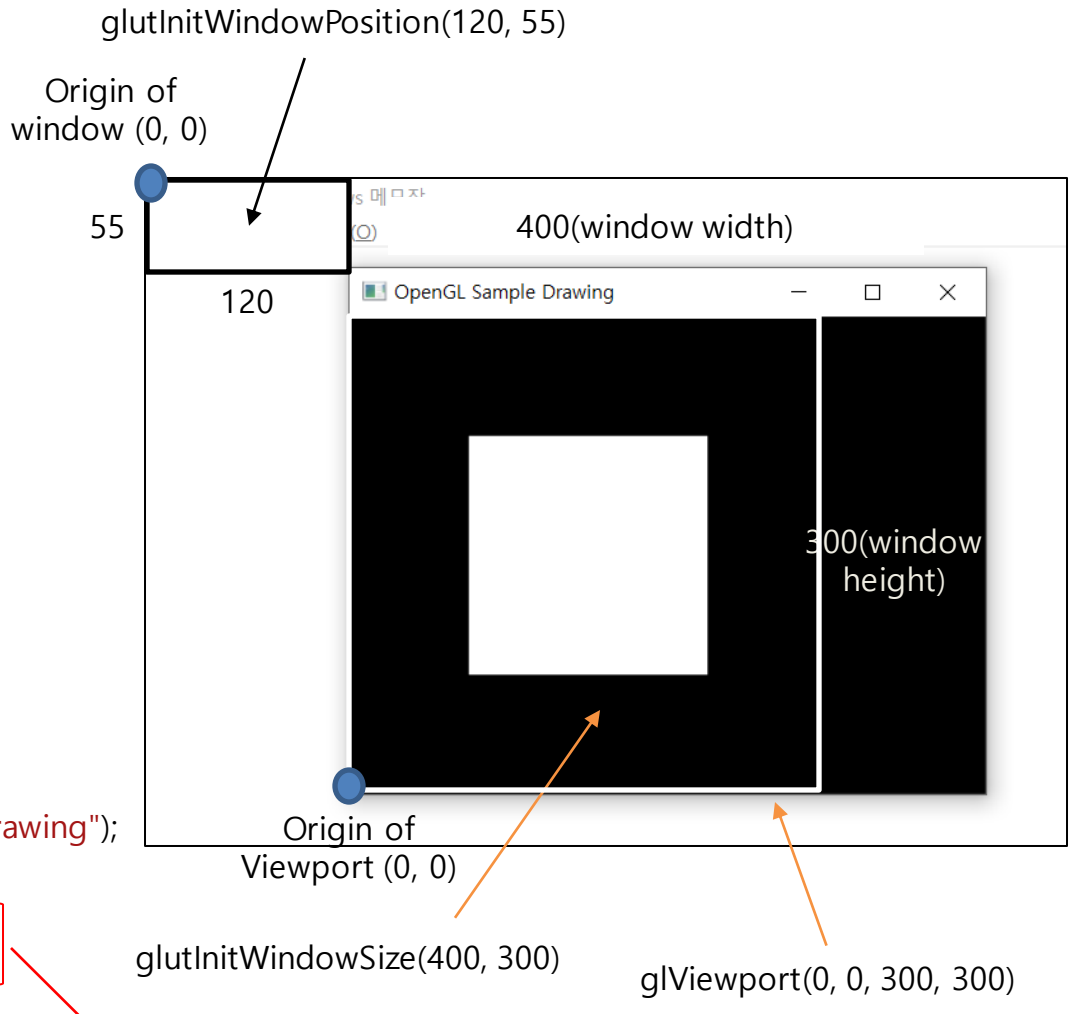
Window and Viewport of GLUT

<Example>

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

void MyDisplay() {
        glClear(GL_COLOR_BUFFER_BIT);
        glViewport(0, 0, 300, 300);
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_POLYGON);
                glVertex3f(-0.5, -0.5, 0.0);
                glVertex3f(0.5, -0.5, 0.0);
                glVertex3f(0.5, 0.5, 0.0);
                glVertex3f(-0.5, 0.5, 0.0);
        glEnd();
        glFlush();
}

int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB);
        glutInitWindowSize(400, 300);
        glutInitWindowPosition(120, 55);
        glutCreateWindow("OpenGL Sample Drawing");
        glClearColor(0.0, 0.0, 0.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
        glutDisplayFunc(MyDisplay);
        glutMainLoop();
        return 0;
}
```



GLUT Modeling

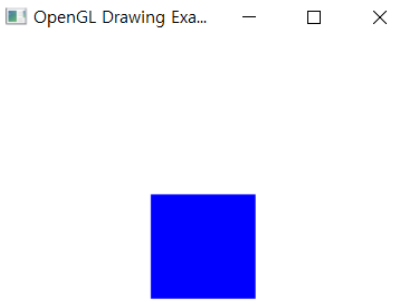
- For user convenience, GLUT provides several objects already modeled.

- To draw it, call the following commands in the display callback function

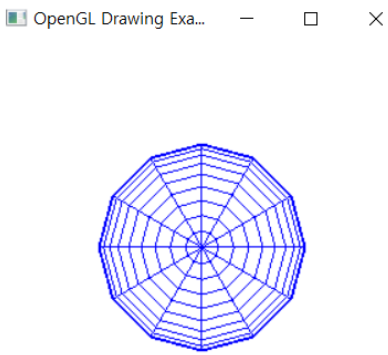
Name	Command
Cube	glutWireCube(GLdouble size); glutSolidCube(GLdouble size);
Sphere	glutWireSphere(GLdouble radius, GLint nSlices, GLint nStacks); glutSolidSphere(GLdouble radius, GLint nSlices, GLint nStacks);
Cone	glutWireCone(GLdouble baseRadius, GLdouble height, GLint nSlices, GLint nStacks); glutSolidCone(GLdouble baseRadius, GLdouble height, GLint nSlices, GLint nStacks);
Torus	glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nSlices, GLint rings); glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nSlices, GLint rings);
Teapot	glutWireTeapot(GLdouble size); glutSolidTeapot(GLdouble size);

GLUT Modeling

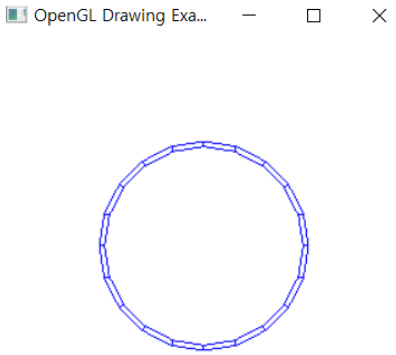
- Examples



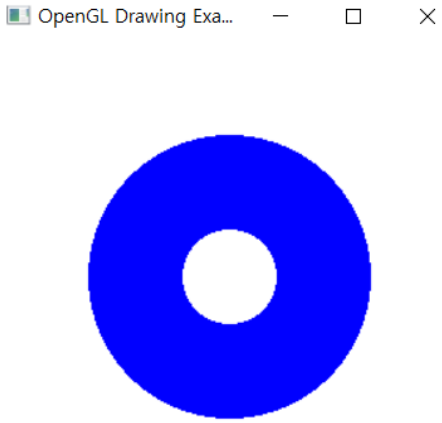
glutSolidCube(0.5);



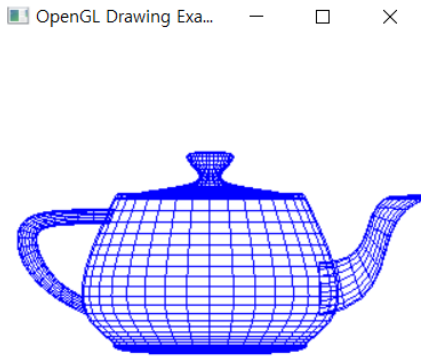
glutWireSphere(0.5, 12, 20);



glutWireCone(0.5, 20, 20, 20);



glutSolidTorus(0.2, 0.4, 60, 60);



glutWireTeapot(0.6);