
Image Formation and OpenGL

Sang Il Park
Dept. of Software

Objectives

- Introduction
 - image formation
 - “classic” OpenGL vs. “new” OpenGL
- Text book: Chapter 1.3~1.8

Image Formation

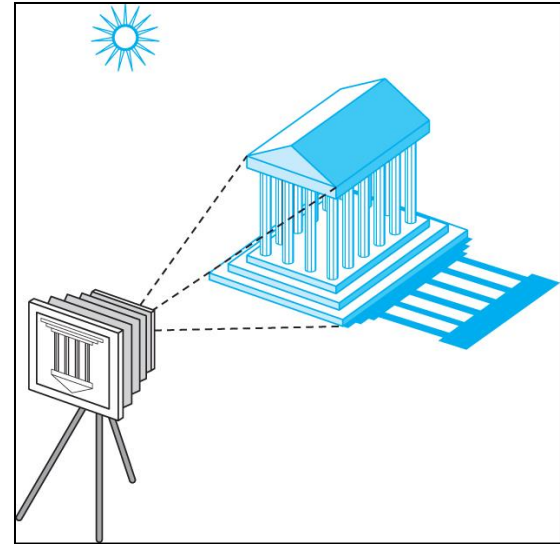
Elements of Image Formation

- What do we need to see the following?



Elements of Image Formation

- Objects
- Viewer
- Light source(s)
- Attributes (materials)
 - govern how light interacts with the materials in the scene

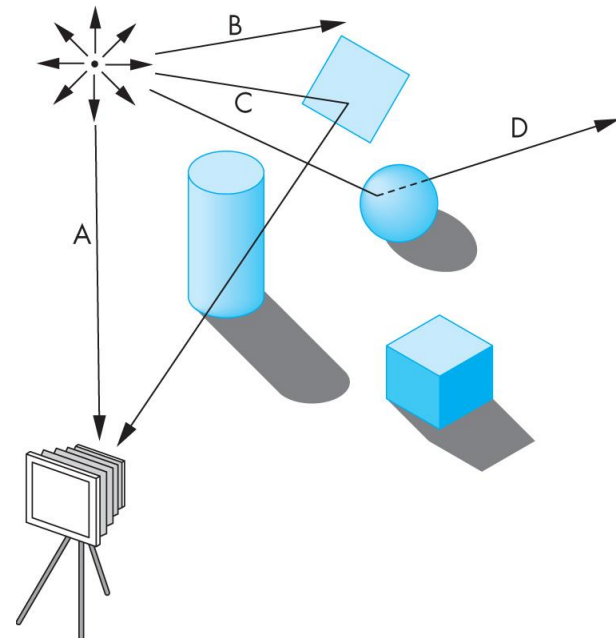
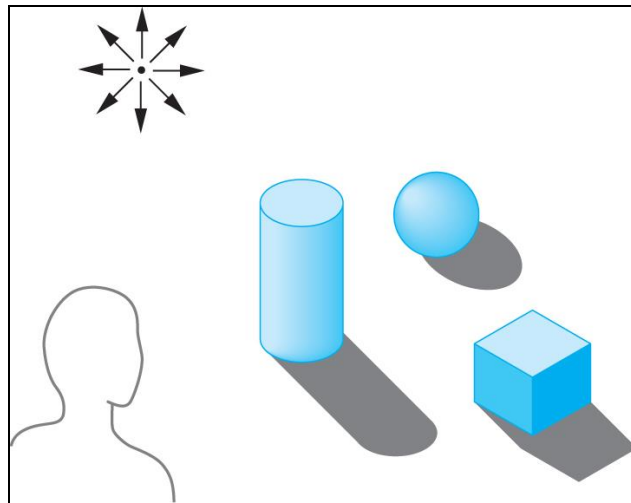


Separation is Important

- Separation of objects, viewer, light sources, and attributes.
- Leads to simple software API
 - Specify objects, lights, camera, attributes
 - Let implementation determine image
- Leads to fast hardware implementation

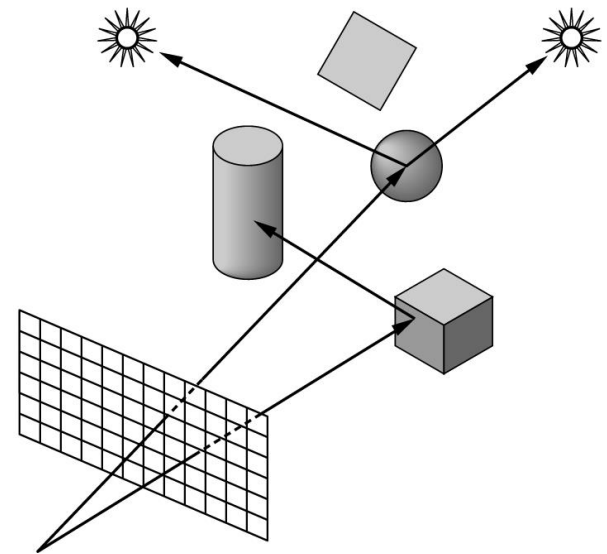
Determine the color

- One way to form an image is to follow rays of light from the lights
- each ray of light may have multiple interactions with objects



Ray tracing

- **Ray tracing:** follow rays of light from center of projection until they either are absorbed by objects or go off to infinity
 - Can handle global effects
 - Multiple reflections
 - Translucent objects
 - Slow
 - Must have whole data base available at all times

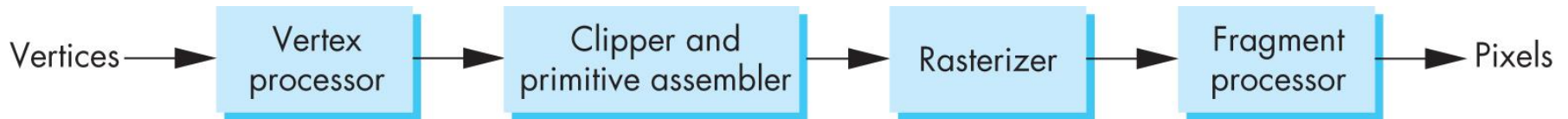


Why not ray tracing?

- Ray tracing seems more physically based so why don't we use it to design a graphics system?
- Possible and is actually simple for simple objects such as polygons and quadrics with simple point sources
- In principle, can produce global lighting effects such as shadows and multiple reflections but ray tracing is slow and not well-suited for interactive applications
- *Now, Ray tracing with GPUs is close to real time*
 - <http://www.rigidgems.sakura.ne.jp>

“Simplified” Practical Approach

- Process objects one at a time in the order they are generated by the application
 - Can consider only local lighting
- Pipeline architecture



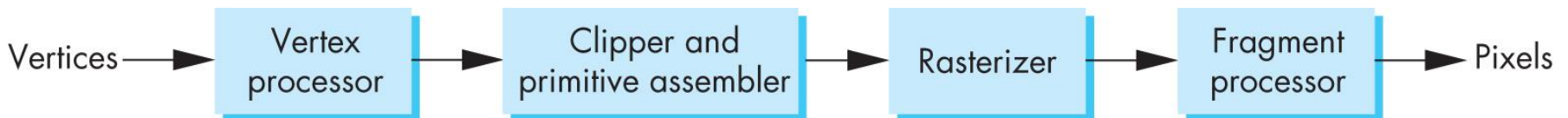
- All steps can be implemented in hardware on the graphics card

Vertex (vertices) Processing

- Much of the work in the pipeline is in converting object representations from one coordinate system to another
 - Object coordinates
 - Camera (eye) coordinates
 - Screen coordinates
- Every change of coordinates is equivalent to a matrix transformation

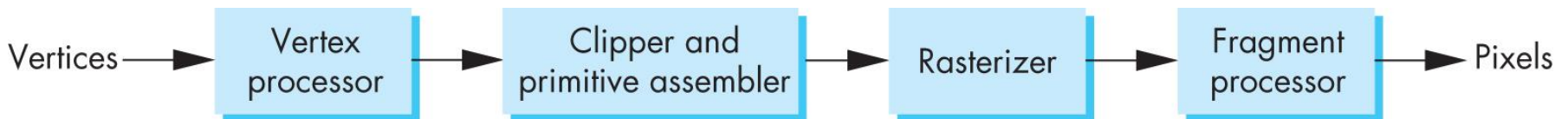


PROJECTION



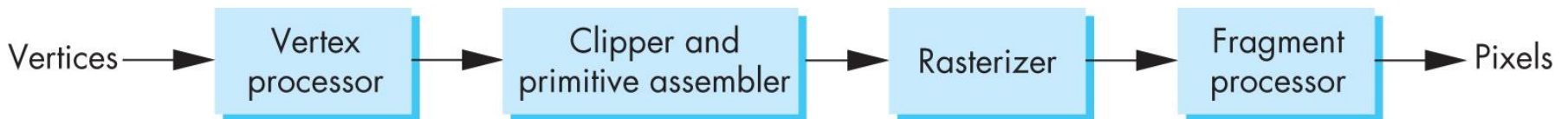
Rasterization

- If an object is visible, the appropriate pixels in the frame buffer must be assigned colors
- Rasterizer produces a set of fragments for each object
- **Fragments** are “potential pixels”
 - Have a location in frame buffer
 - Color and depth attributes
- Vertex attributes are interpolated over objects by the rasterizer



Fragment Processing

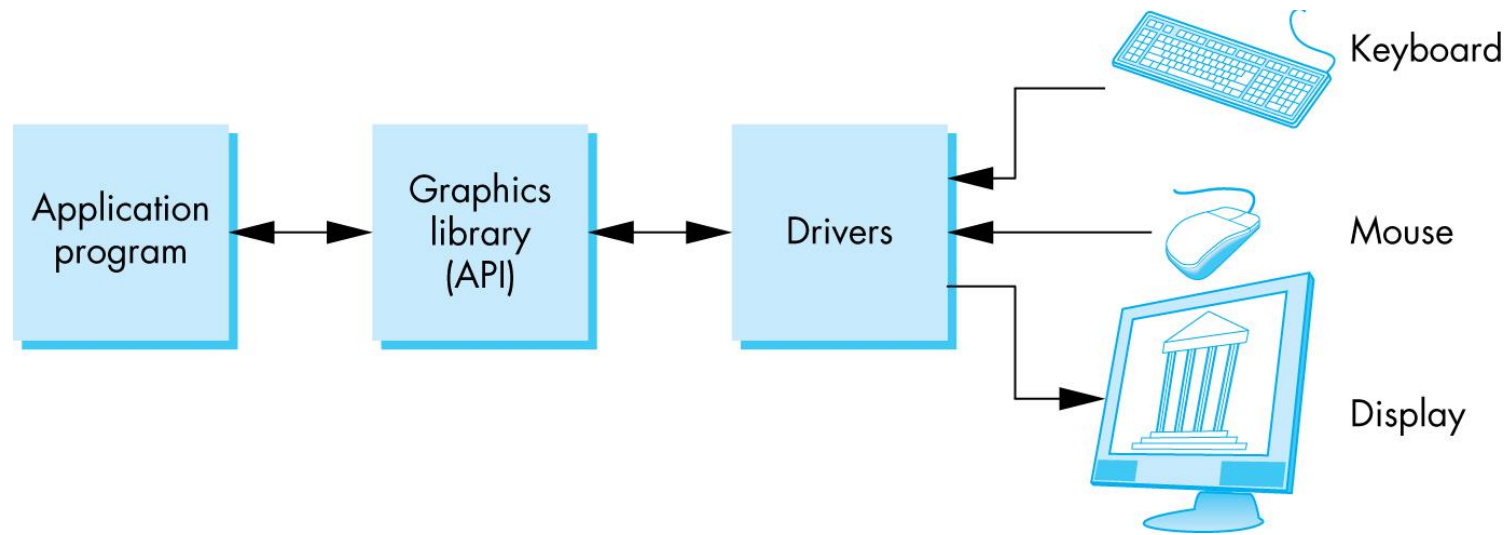
- Fragments are processed to determine the color of the corresponding pixel in the frame buffer
- Colors can be determined by texture mapping or interpolation of vertex colors
- Fragments may be blocked by other fragments closer to the camera
 - Hidden-surface removal



OpenGL: the API

The Programmer's Interface

- Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)



API Contents

- Functions that specify what we need to form an image
 - Objects
 - Viewer
 - Light Source(s)
 - Attributes (Materials)
- Other information
 - Input from devices such as mouse and keyboard
 - Capabilities of system

Object Specification

- Most APIs support a limited set of primitives including
 - Points (0D object)
 - Line segments (1D objects)
 - Polygons (2D objects)
 - Some curves and surfaces
 - Quadrics
 - Parametric polynomials
- All are defined through locations in space or *vertices*

Object Specification (old style)

type of object

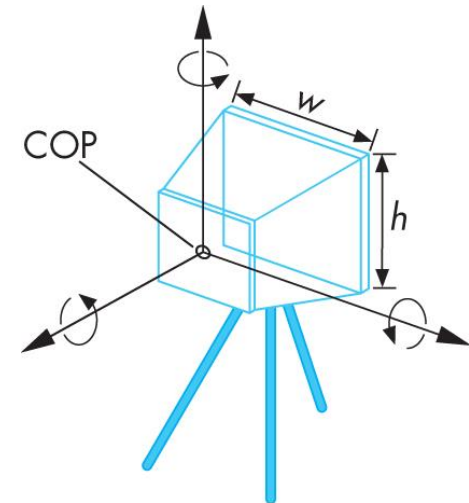
location of vertex

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

end of object definition

Camera Specification (old style)

- Six degrees of freedom
 - Position of center of lens
 - Orientation
- Lens
- Film size
- Orientation of film plane



- \rightarrow `glPerspective(...)`, `gluLookAt(...)`

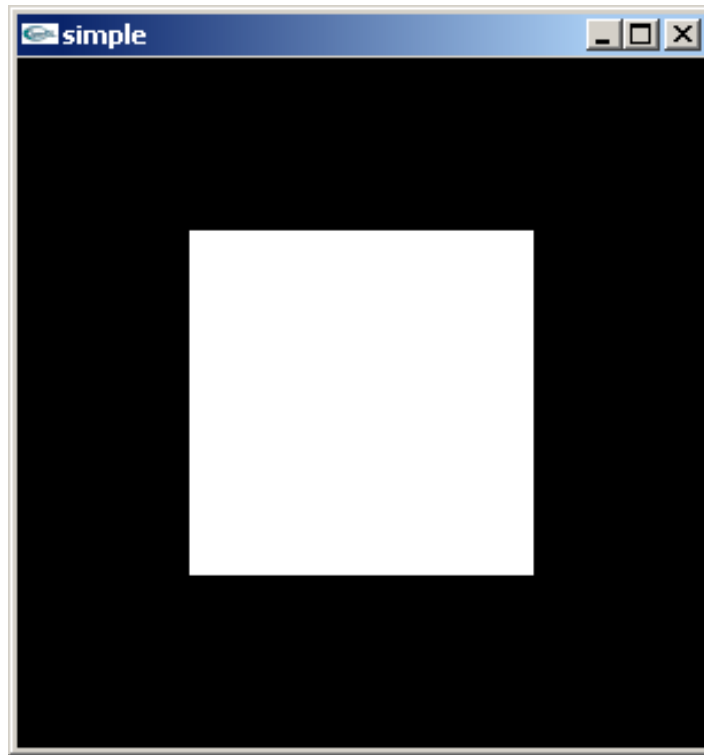
Lights and Materials (old style)

- Types of lights
 - Point sources vs distributed sources
 - Spot lights
 - Near and far sources
 - Color properties
- Material properties
 - Absorption: color properties
 - Scattering
 - Diffuse
 - Specular

→ `glLight(...)`, `glMaterial(...)`

A Simple Program (old style)

Generate a square on a solid background



Let's start to CODE it!

- Preparation

1. Download necessary libraries

- Header files: Include folder
- LIB files : lib folder
- DLL files : bin(or system32) folder

2. Change the project setting

- Directory setting

Required Libraries

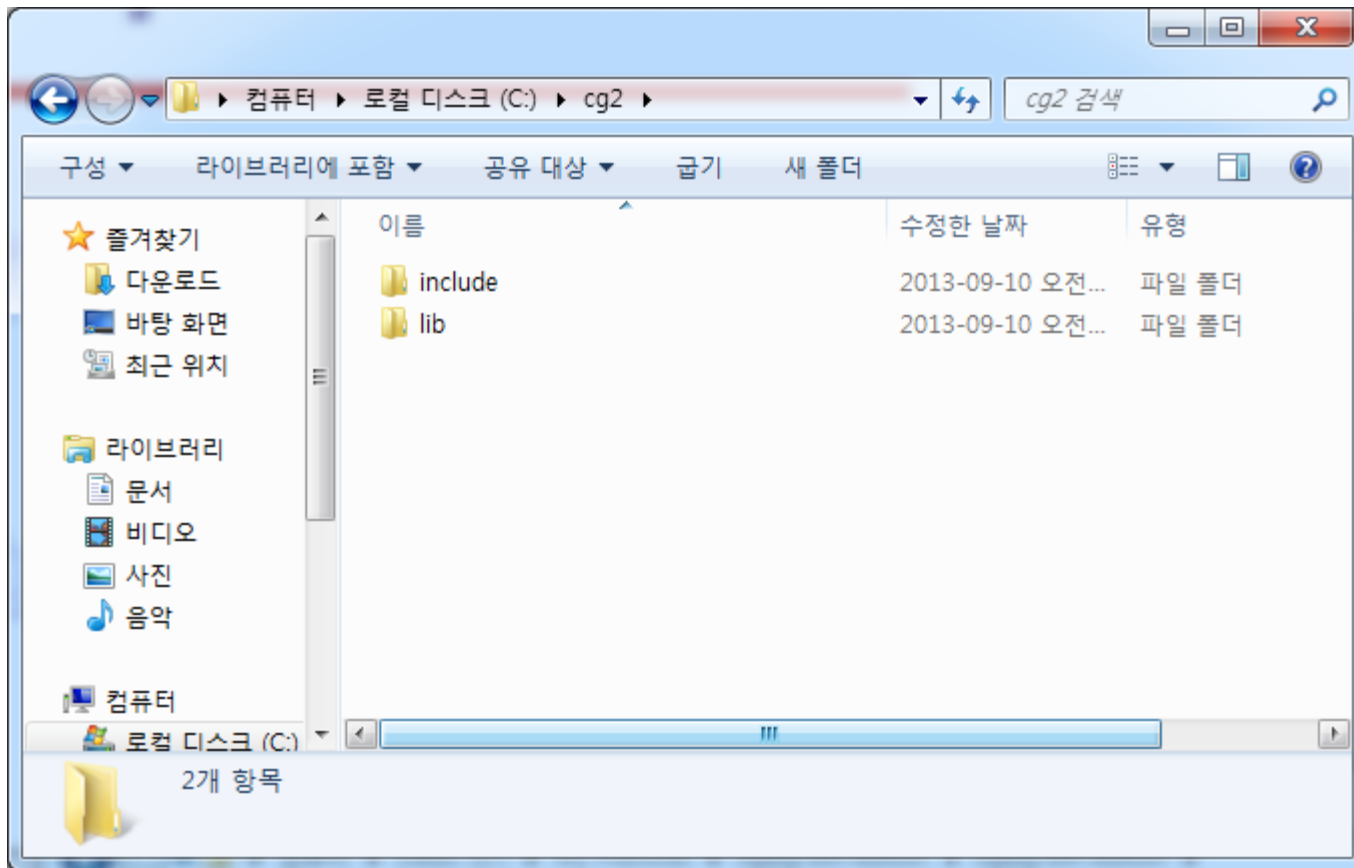
- freeglut: <http://freeglut.sourceforge.net/>
- GLEW: <http://glew.sourceforge.net/>

I put both of them and more at our homepage:

freeglut_and_glew.zip

➔ Download it and unzip it at “c:/cg2/”

What you should have in your “c:/cg2/” folder:



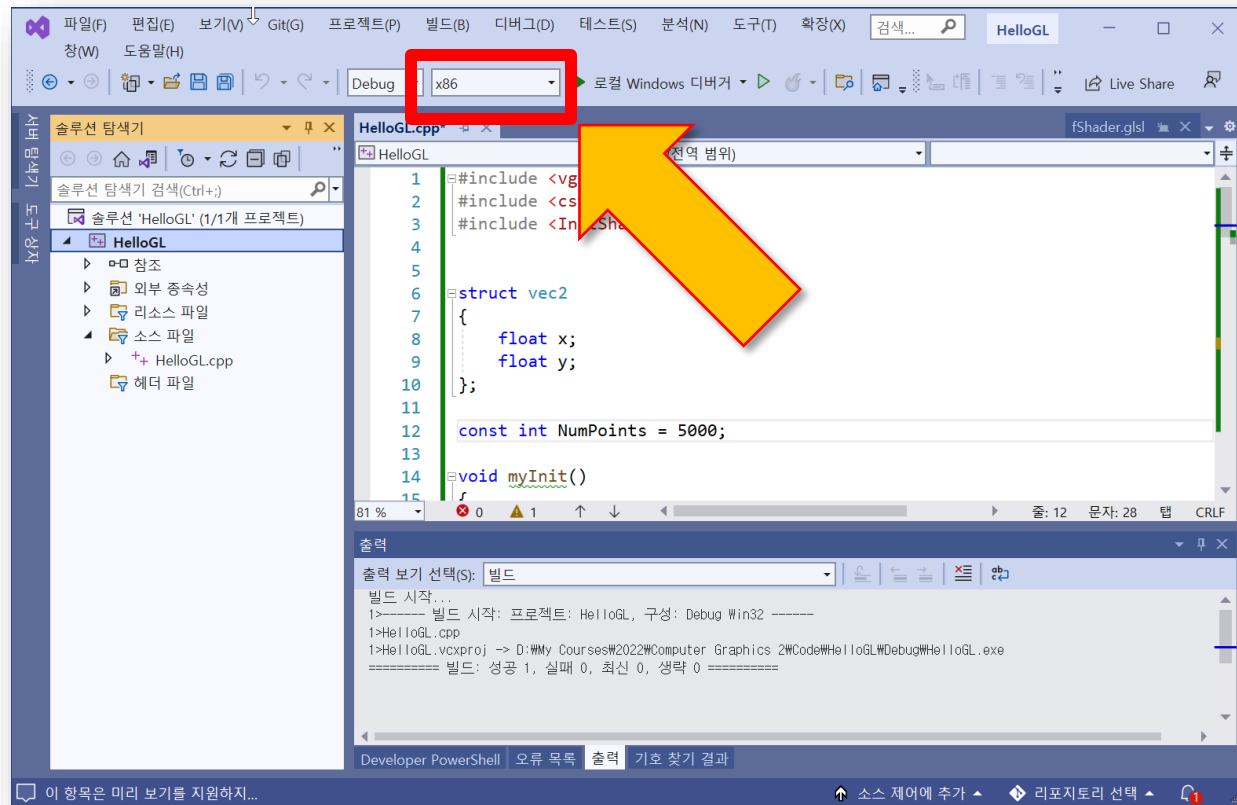
Project Setting:

- Start a new project with a “**console application**” project with an empty project option.



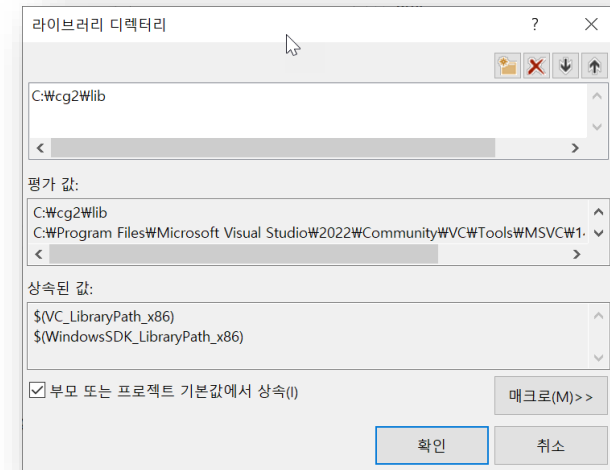
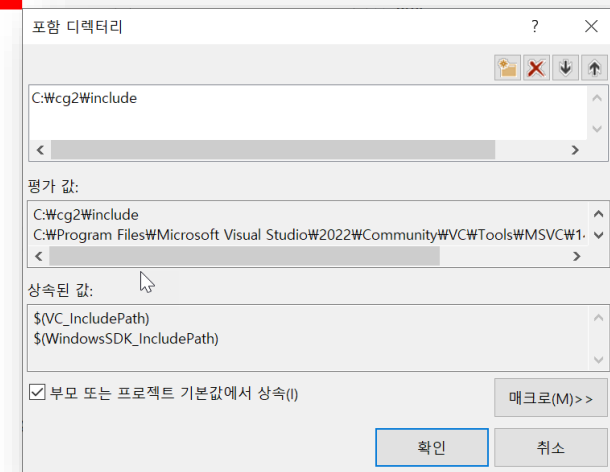
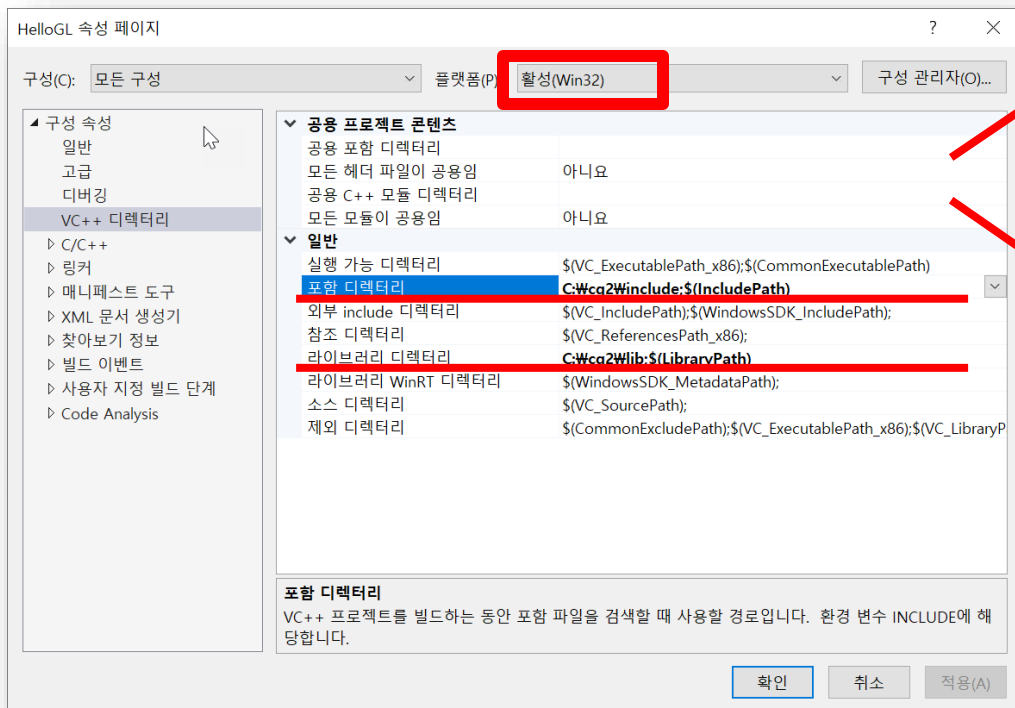
Set Target Platform: x86

- 다음과 같이 반드시 Target Platform을 x86으로 설정 (x64 아님!)



Project Setting

• Set the directories:



Create a new main.cpp file

- And add the following line at the beginning of the code:

```
#include <vgl.h>
```

Now ALL SET!!!!

Hello GL Program:

```
#include <vgl.h>
```

```
void display()
```

```
{
```

```
    glClear(GL_COLOR_BUFFER_BIT);
```

```
    glBegin(GL_TRIANGLES);
```

```
        glVertex2f(-0.5, -0.5);
```

```
        glVertex2f(0.5, -0.5);
```

```
        glVertex2f(-0.5, 0.5);
```

```
        glVertex2f(0.5, -0.5);
```

```
        glVertex2f(0.5, 0.5);
```

```
        glVertex2f(-0.5, 0.5);
```

```
    glEnd();
```

```
    glFlush();
```

```
}
```

```
int main(int argc, char** argv)
```

```
{
```

```
    glutInit(&argc, argv);
```

```
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
```

```
    glutInitWindowSize(512, 512);
```

```
    glutCreateWindow("Hello GL");
```

```
    glutDisplayFunc(display);
```

```
    glutMainLoop();
```

```
    return 0;
```

```
}
```