

Lecture 5: OpenGL Basics

Sep 16, 2025

Won-Ki Jeong

(wkjeong@korea.ac.kr)



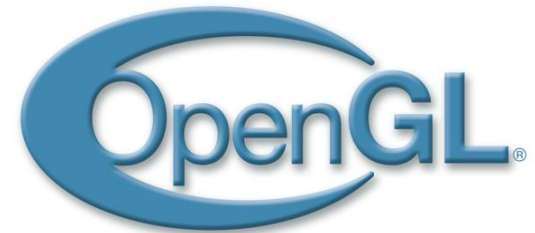
Outline

- OpenGL History
- How to create OpenGL application
- How to set up OpenGL camera
- How to pass the data to the GPU
- How to display onto screen
- How to make realistic 3D rendering



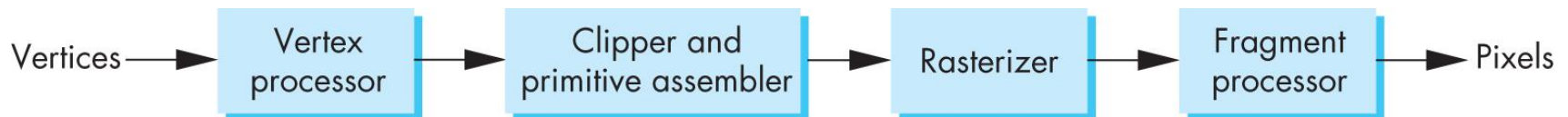
OpenGL

- OpenGL is a platform-independent API for graphics, first introduced in 1992
 - Interface for graphics hardware
- OpenGL is concerned only with rendering into a framebuffer and reading values in it
 - Easy to use
 - Platform independent
 - User interface is omitted
 - Windows system independent



Modern OpenGL

- Performance is achieved by using GPU rather than CPU
- Control GPU through programs called shaders
- Application's job is to send data to GPU
- GPU does all rendering



OpenGL Evolution

- OpenGL 1.x : fixed pipeline
- OpenGL 2.x : programmable pipeline
- OpenGL 3.x : deprecated APIs
- The latest version is 4.6
 - Most of fixed stages and global state model are removed
 - Still available through GL_ARB_compatibility extension



Other GL Versions

- OpenGL ES
 - Embedded systems (Android, iPad,...)
 - Version 1.0 simplified OpenGL 2.1
 - Version 2.0 simplified OpenGL 3.1
 - Shader based
 - Version 3.0 simplified OpenGL 4.3
- WebGL
 - Javascript implementation of OpenGL ES
 - Supported on newer browsers



GLUT

- OpenGL Utility Toolkit (GLUT)
 - Platform-independent window management library
 - Create a GL window
 - Get input from mouse and keyboard
 - Menus
- freeglut adds new functionalities to original GLUT

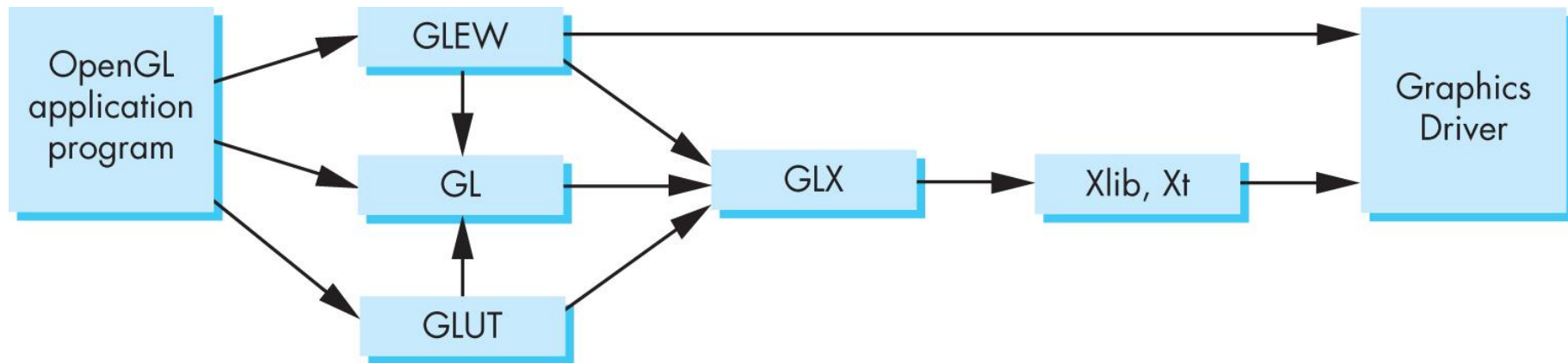


GLEW

- OpenGL Extension Wrangler Library
- Makes it easy to access OpenGL extensions available on a particular system
- Avoids having to have specific entry points in Windows code
- Application needs only to include `glew.h` and run a `glewInit()`

Download: <http://glew.sourceforge.net>

Software Organization



OpenGL is a Client-Server Model

- Host process (CPU)
 - Client
 - Send commands and data to server
 - Interconnect: PCIe
- Graphics hardware (GPU)
 - Server
 - Performs functions directed by the client



OpenGL Elements

- States
 - Snapshot of global parameters
 - GL context
- Primitives
 - Group of vertices
- Commands
 - Execution of graphics functions



Start with Skeleton Code

- MS Visual Studio
 - <https://visualstudio.microsoft.com/ko/>
 - Visual Studio Download
- Download from LMS (Board→강의자료실)
 - glew
 - CMake
 - OpenGL Skeleton Code



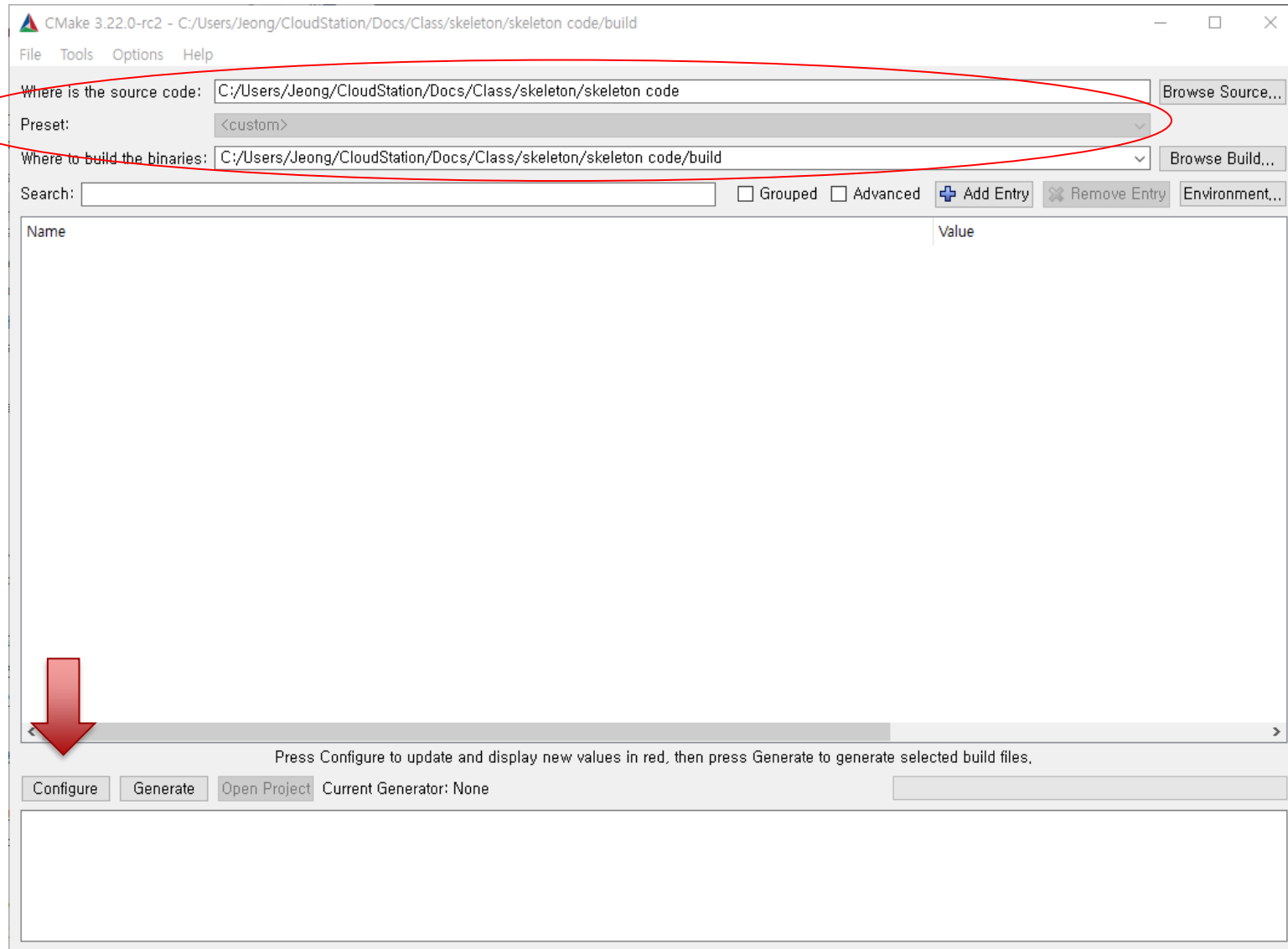
Check OpenGL Version

- glewinfo.exe → glewinfo.txt

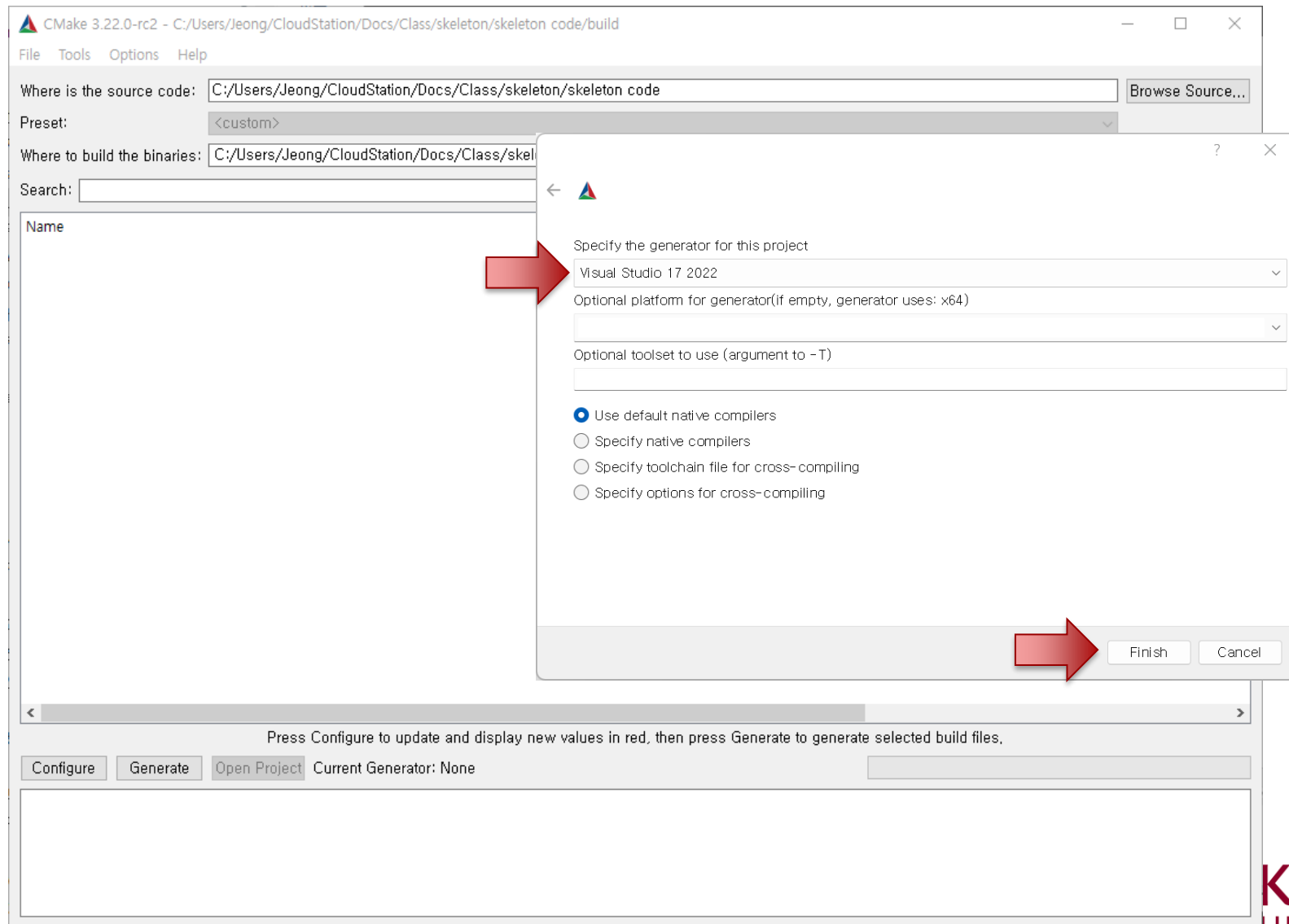
```
-----  
GLEW Extension Info  
-----  
  
GLEW version 2.1.0  
Reporting capabilities of pixelformat 3  
Running on a Intel(R) HD Graphics 620 from Intel  
OpenGL version 4.4.0 - Build 21.20.16.4550 is supported  
  
GL_VERSION_1_1: OK  
-----  
  
GL_VERSION_1_2: OK  
-----  
    glCopyTexSubImage3D: OK  
    glDrawRangeElements: OK  
    glTexImage3D: OK  
    glTexSubImage3D: OK  
  
GL_VERSION_1_2_1: OK  
-----  
  
GL_VERSION_1_3: OK  
-----  
    glActiveTexture: OK  
    glClientActiveTexture: OK  
    glCompressedTexImage1D: OK  
    glCompressedTexImage2D: OK
```



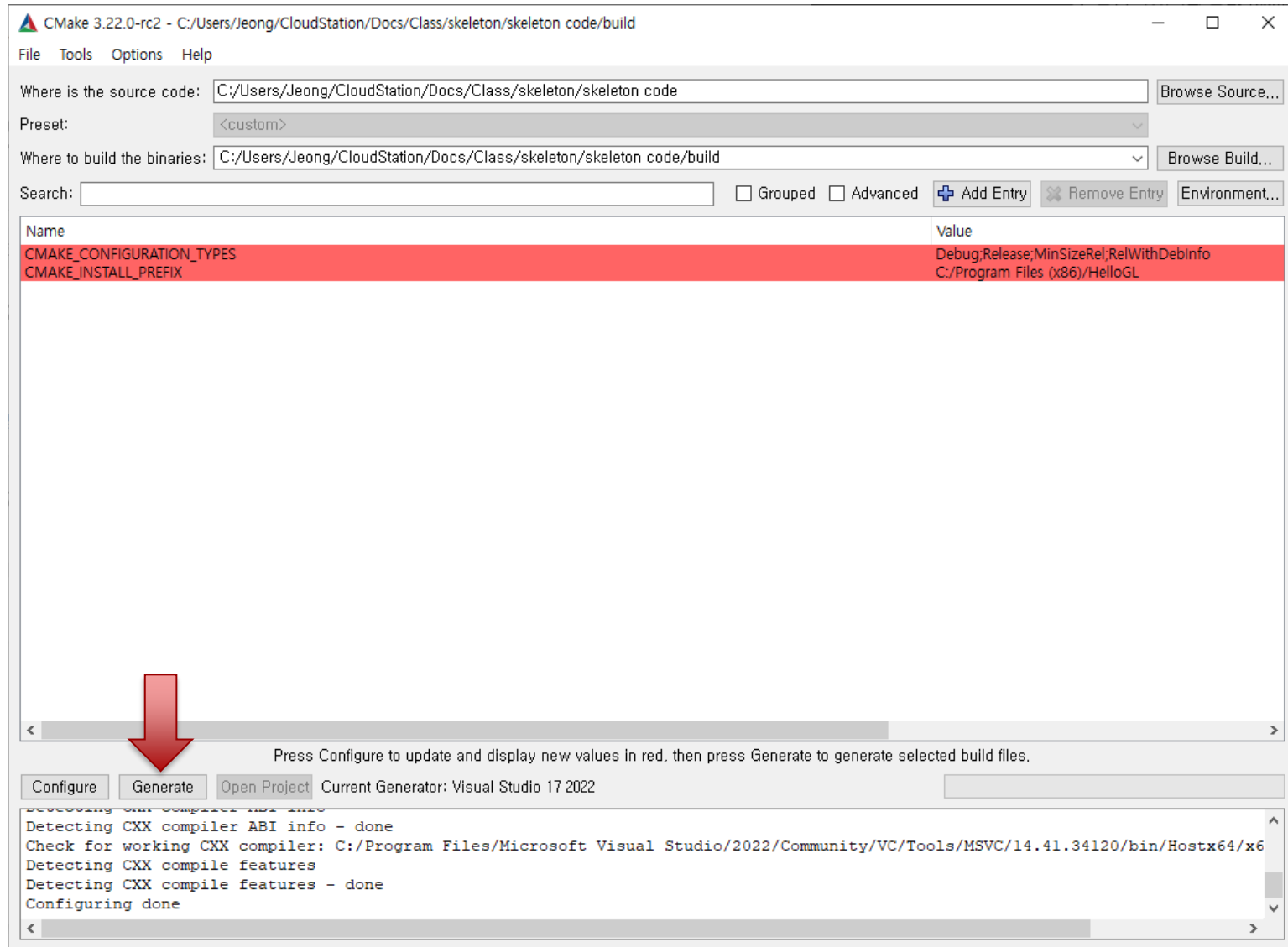
Run CMake to Create VS Project



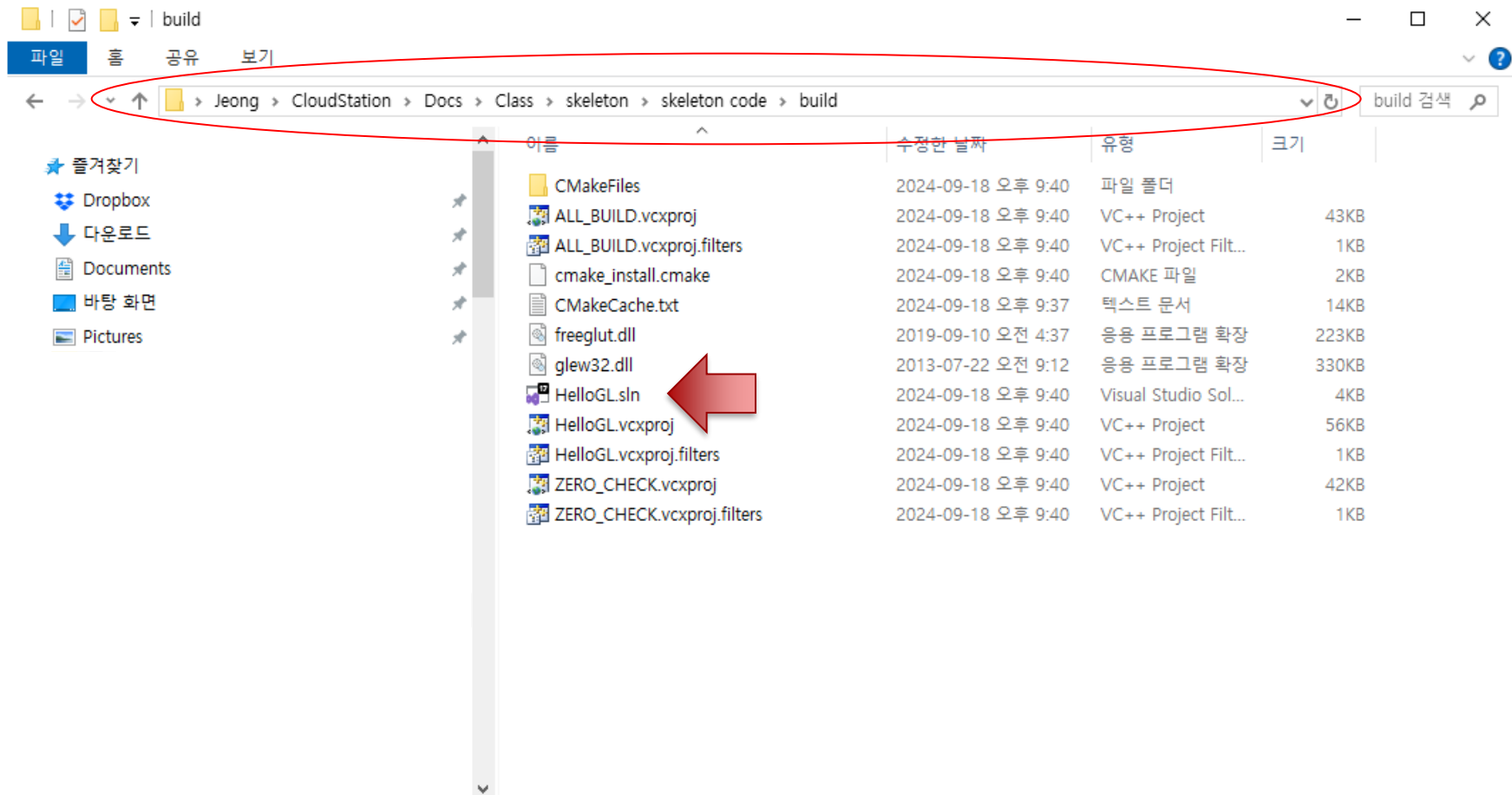
Run CMake to Create VS Project



Run CMake to Create VS Project



Location of .sln File



파일(F) 편집(E) 보기(V) Git(G) 프로젝트(P) 빌드(B) 디버그(D) 테스트(S) 분석(N) 도구(T) 확장(X) 창(W) 도움말(H) HelloGL 로그인

Debug x64 로컬 Windows 디버거

새로운 기능 main.cpp CMakeLists.txt

HelloGL (전역 범위) reshape(int w, int h)

```
1  /*
2   * Skeleton code for COSE436 Interactive Visualization
3   *
4   * Won-Ki Jeong, wkjeong@korea.ac.kr
5   */
6
7  #include <stdio.h>
8  #include <GL/glut.h>
9
10 void reshape(int w, int h)
11 {
12     glLoadIdentity();
13     glViewport(0, 0, w, h);
14     glOrtho(0, 100, 0, 100, 100, -100);
15 }
16
17
18 void display(void) {
19
20     glClear(GL_COLOR_BUFFER_BIT);
21
22     // Draw something here!
23     glColor3f(0, 0, 1);
24     glRectf(30, 30, 70, 70);
25
26     glColor3f(1,0,1);
27     glRectf(40,40,60,60);
28
29     glutSwapBuffers();
30 }
31
32 int main(int argc, char **argv) {
33
34     // init GLUT and create Window
35     glutInit(&argc, argv);
36     glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
37     glutInitWindowPosition(100,100);
38     glutInitWindowSize(800,800);
```

100 % 문제 가 검색되지 않음 줄: 14 문자: 37 열: 40 혼합 CRLF

출력 출력 보기 선택(S):

출루션 탐색기

출루션 탐색기 검색(Ctrl+)

출루션 'HelloGL' (3 프로젝트의 3)

- ALL_BUILD
- HelloGL** Set as Startup Project
- 참조
- 외부 종속성
- Source Files
 - main.cpp
 - CMakeLists.txt
- ZERO_CHECK

출루션 탐색기 Git 변경 내용

↑ 소스 제어에 추가 리포지토리 선택

준비

6

ATY

Hello World GL

- Use GLUT to handle GL windows and event
 - You do not need to use OS specific windows handling APIs
- GLUT is event-driven, callback function is called
 - When your screen is changed
 - When you press a keyboard
 - When you move mouse
 - When you are not doing anything



Create GL Window using GLUT

- Main window management and registering callback functions

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    glutReshapeFunc(reshape);           // reshape callback
    glutDisplayFunc(display);           // display callback
    glutKeyboardFunc(keyboard);         // keyboard callback
    glutMouseFunc(mouse);               // Mouse callback
    glutMainLoop();
    return 0;
}
```



GLUT Functions

- **glutInit** allows application to get command line arguments and initializes system
- **gluInitDisplayMode** requests properties for the window (the *rendering context*)
 - RGB color
 - Double buffering
 - Use depth buffer
- **glutWindowSize** window size in pixels
- **glutWindowPosition** top-left corner position of display
- **glutCreateWindow** create window with title
- **glutDisplayFunc** display callback
- **glutMainLoop** enter infinite event loop



GLUT Callback Example

- `glutKeyboardFunc(void(*f)(key, x, y))`

```
void keyboard(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'Q' :
        case 'q' :
        case 27 : // ESC
            exit(0);
            break;
    }
}
```



Example: Hello World GL

```
#include <glut.h>

void reshape(int w, int h)
{
    glLoadIdentity();
    glViewport(0,0,w,h);
    glOrtho(0, 100, 0, 100, 100, -100);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0,0,1);
    glRectf(30,30,70,70);
    glColor3f(1,0,1);
    glRectf(40,40,60,60);
    glutSwapBuffers();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("Hello World");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```

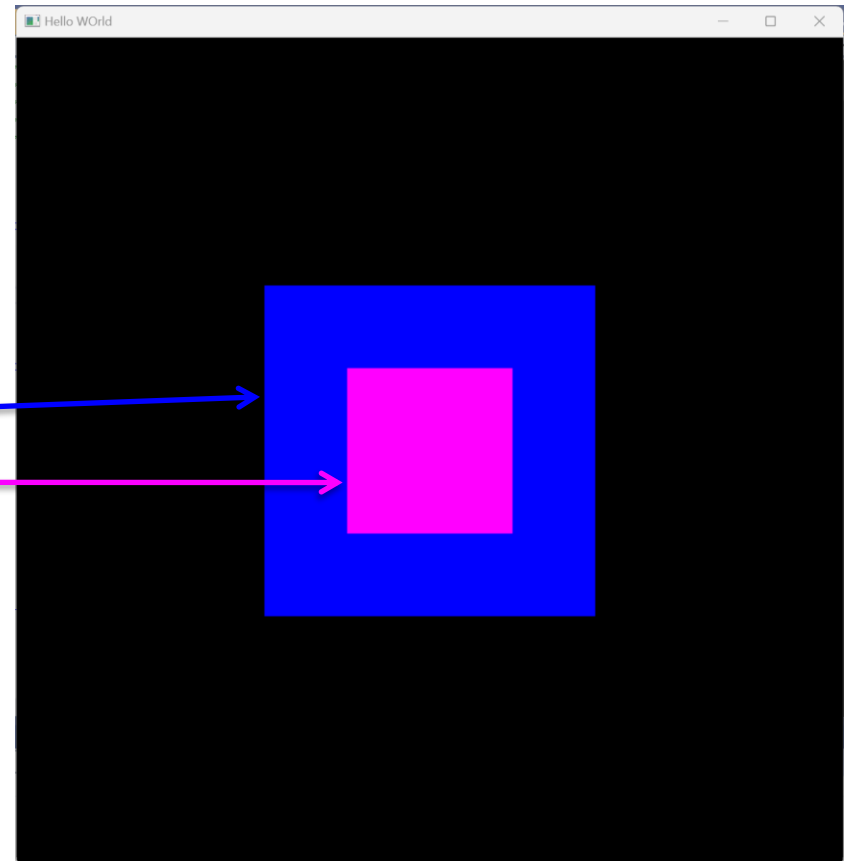
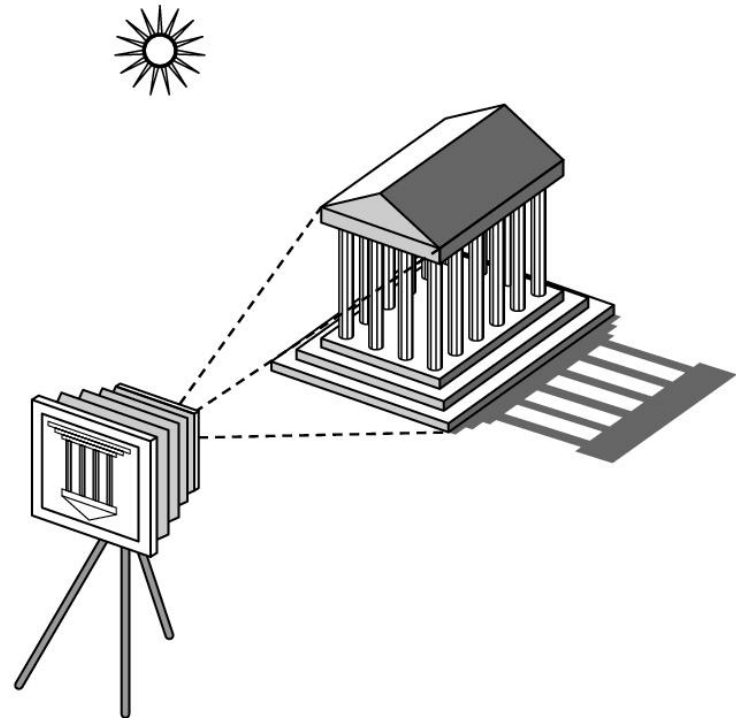


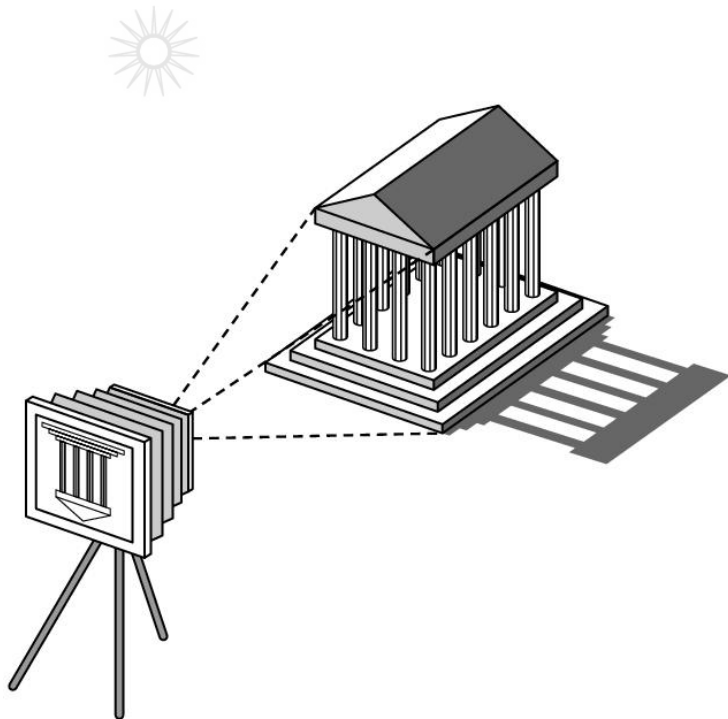
Image Formation

- Object
- Viewer
- Light sources



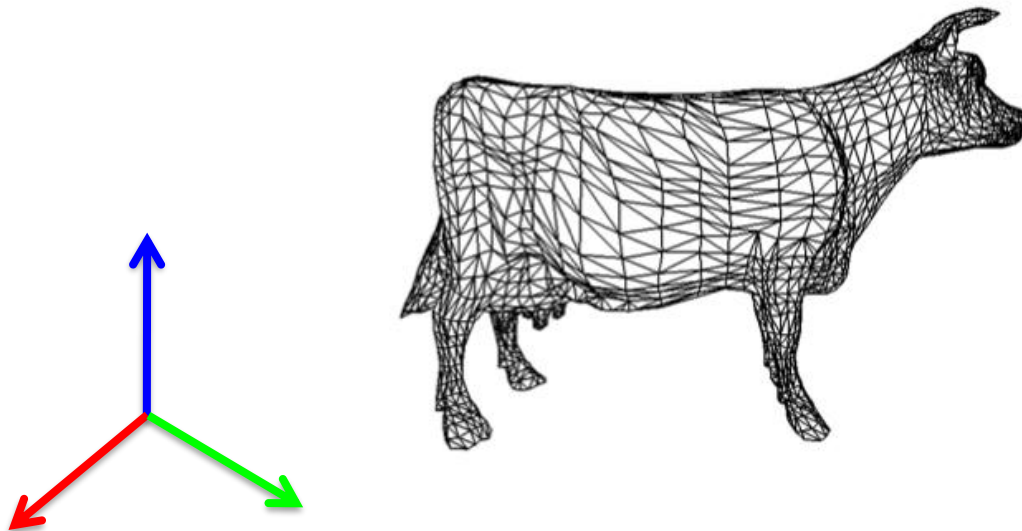
Today: Image Formation in GL

- Object
- Viewer
- Light sources



Object in OpenGL

- How to place objects in 3D space?
 - Represent object as 3D polygons
 - Assign coordinate per vertex



Handling GL Data

- Immediate mode
 - Transfer data when rendering
 - Copy from CPU to GPU per render call
 - Expensive for large datasets
- Buffer objects
 - Data reside in GPU memory
 - Fast and flexible



Vertex Array

- Use array as vertex attributes
- Single GL draw command
- Stored in **client(CPU)**'s memory space
- Transfer data from client(CPU) to host(GPU) for rendering
 - *Immediate mode*

```
GLfloat vertices[] = {...};  
...  
glEnableClientState(GL_VERTEX_ARRAY);  
glVertexPointer(3, GL_FLOAT, 0, vertices);  
  
glDrawArrays(GL_TRIANGLES, 0, n);  
  
glDisableClientState(GL_VERTEX_ARRAY);
```



Primitives

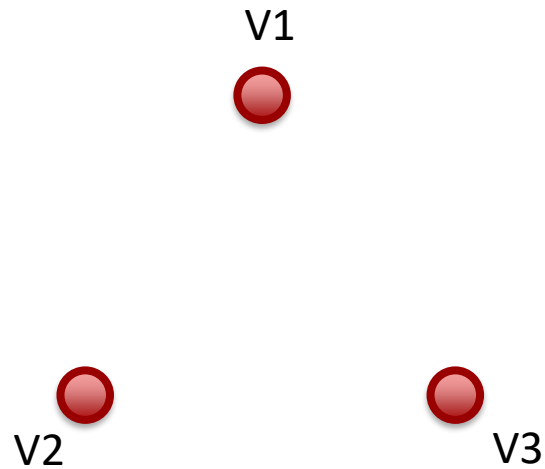
- From vertices to 3D objects
- How to tessellate (connect) vertices
- Point, Line, Polygon, 3D shapes

```
glDrawArrays (GL_TRIANGLES, 0, n) ;
```



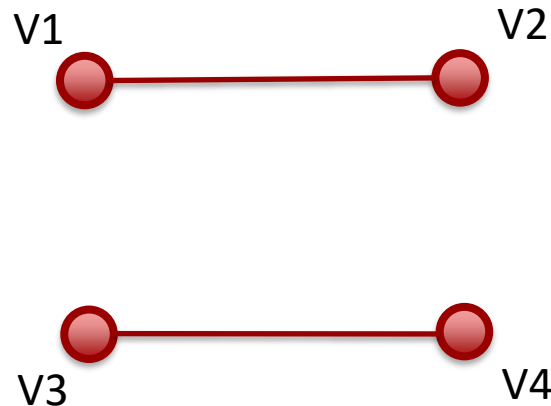
Point

- GL_POINTS



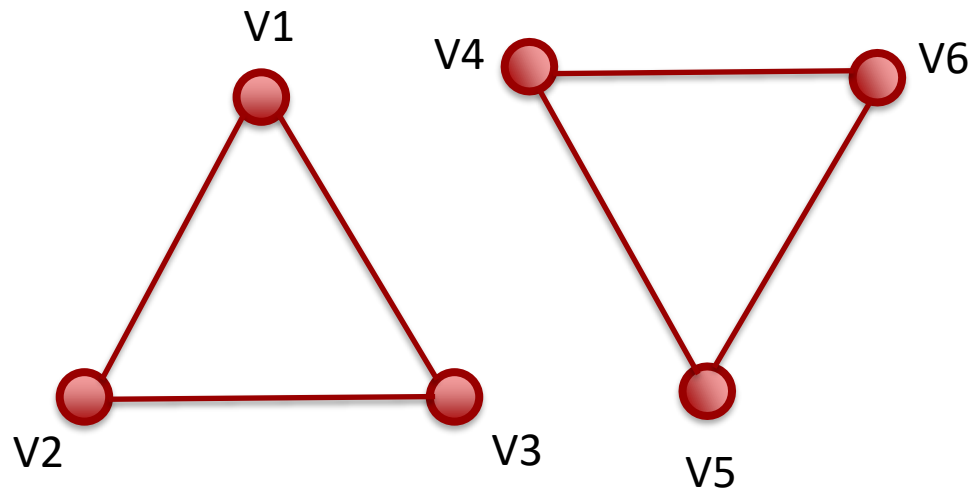
Line

- GL_LINES
 - Group every two vertices for a line segment



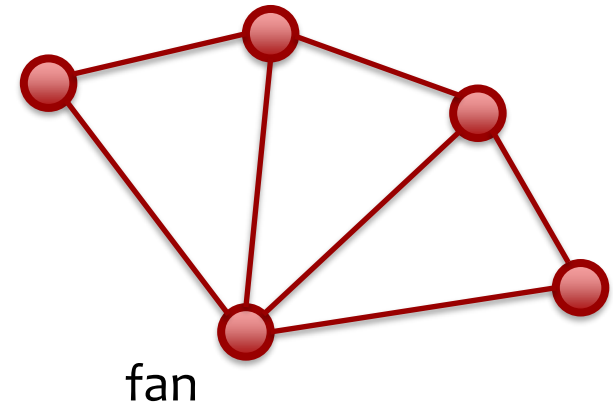
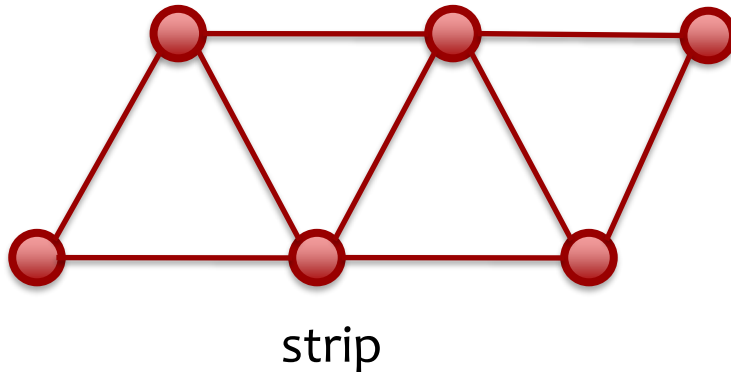
Triangle

- GL_TRIANGLES
 - Group every three vertices for a triangle



Other Primitives

- GL_LINE_STRIP, GL_LINE_LOOP
- GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN
- GL_QUADS, GL_QUAD_STRIP
- GL_POLYGON



Handling Vertex Attributes

```
// two coords (x,y) per 2D vertex
float coords[6] = { 25,25, 75,50, 50,75 };

// three RGB values per vertex
float colors[9] = { 1,0,0, 0,1,0, 0,0,1 };

// Set data type and location
glVertexPointer( 2, GL_FLOAT, 0, coords );
glColorPointer( 3, GL_FLOAT, 0, colors );

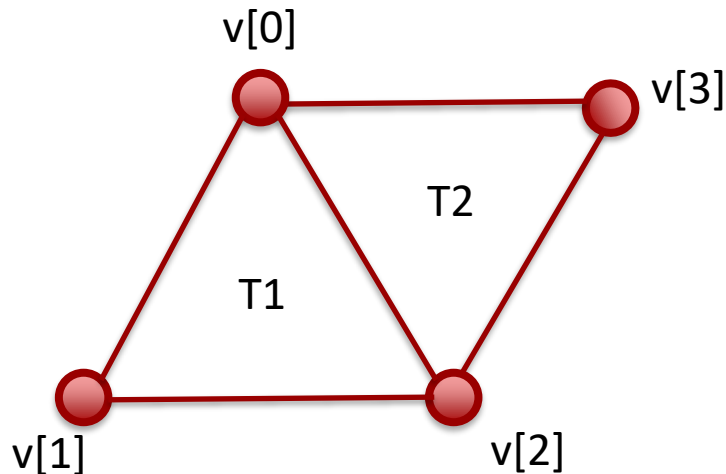
// Enable arrays
glEnableClientState( GL_VERTEX_ARRAY );
glEnableClientState( GL_COLOR_ARRAY );

// Draw triangle using 3 vertices, start with vertex 0
glDrawArrays( GL_TRIANGLES, 0, 3 );
```



Handling Redundant Vertices

- Use index array



```
// Coords for 3D vertices
```

```
float v[] = { 1.0,1.0,1.0,  
              2.0,4.0,1.0,  
              3.0,1.0,1.0,  
              4.0,4.0,1.0 };
```

```
// Two triangles (T1, T2)
```

```
int idx = { 0,1,2, 0,2,3 };
```

Handling Vertex Index

```
// Cube example
// 8 vertices (3 coords/colors per vertex)
// 6 faces (4 indices per face)
float vertexCoords[24] = { ... };
float vertexColors[24] = { ... };
int indexArray[24] = { ... };

glVertexPointer( 3, GL_FLOAT, 0, vertexCoords );
glColorPointer( 3, GL_FLOAT, 0, vertexColors );

glEnableClientState( GL_VERTEX_ARRAY );
glEnableClientState( GL_COLOR_ARRAY );

glDrawElements( GL_QUADS, 24, GL_UNSIGNED_INT, indexArray );
```



Vertex/Index Buffer Object (VBO/IBO)

- Store vertices in **server(GPU)**'s memory space
- No transfer between client to server for rendering
- Most efficient method



Vertex Buffer Object

- Create buffer, bind buffer, copy data

```
float vertices[] = {.....}; // n vertices
```

```
GLuint buffer;
```

```
glGenBuffers(1, &buffer);
```

```
glBindBuffer(GL_ARRAY_BUFFER, buffer);
```

```
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,  
             GL_STATIC_DRAW);
```

```
glEnableClientState(GL_VERTEX_ARRAY);
```

```
glVertexPointer(3, GL_FLOAT, 0, 0);
```

use currently bound buffer

```
glDrawArrays(GL_TRIANGLES, 0, n);
```

of vertices to draw



Index Buffer Object

- Create buffer, bind buffer, copy data

```
GLuint indices[] = {....}; // indices for n vertices
GLuint ibo;
glGenBuffers(1, &ibo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibo);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices),
             indices, GL_STATIC_DRAW);
```

```
glDrawElements(GL_TRIANGLES, n, GL_UNSIGNED_INT, 0 );
```



of vertices to draw



use currently bound buffer



Pre-defined 3D Shapes

- glut functions (solid, wire)
 - glut***Cube
 - glut***Tetrahedron
 - glut***Icosahedron
 - glut***Sphere
 - glut***Torus
 - glut***Cone
 - glut***Teapot



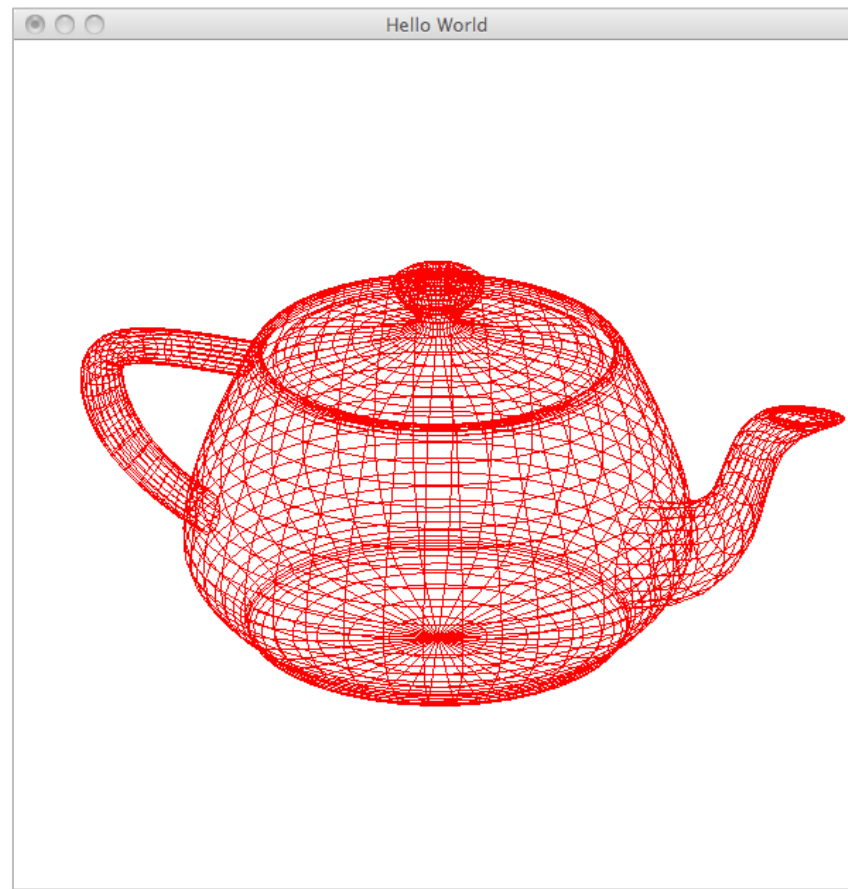
glPolygonMode(face,mode)

- Polygon rasterization mode
- face
 - GL_FRONT
 - GL_BACK
 - GL_FRONT_AND_BACK
- mode
 - GL_POINT
 - GL_LINE
 - GL_FILL



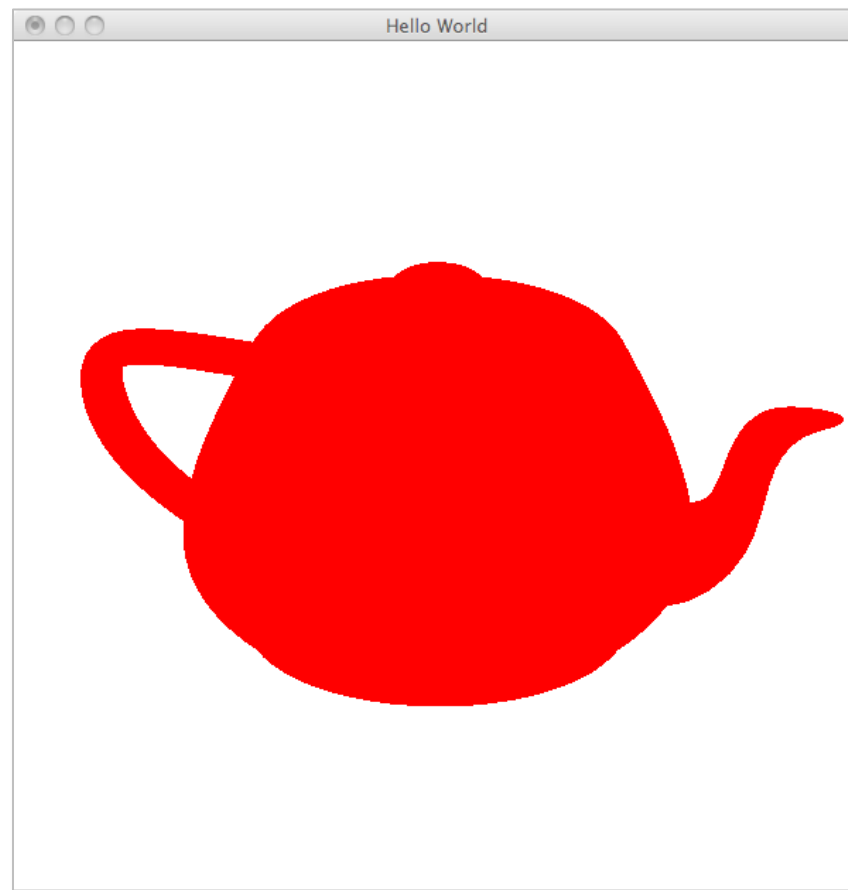
Wireframe Rendering

- `glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)`



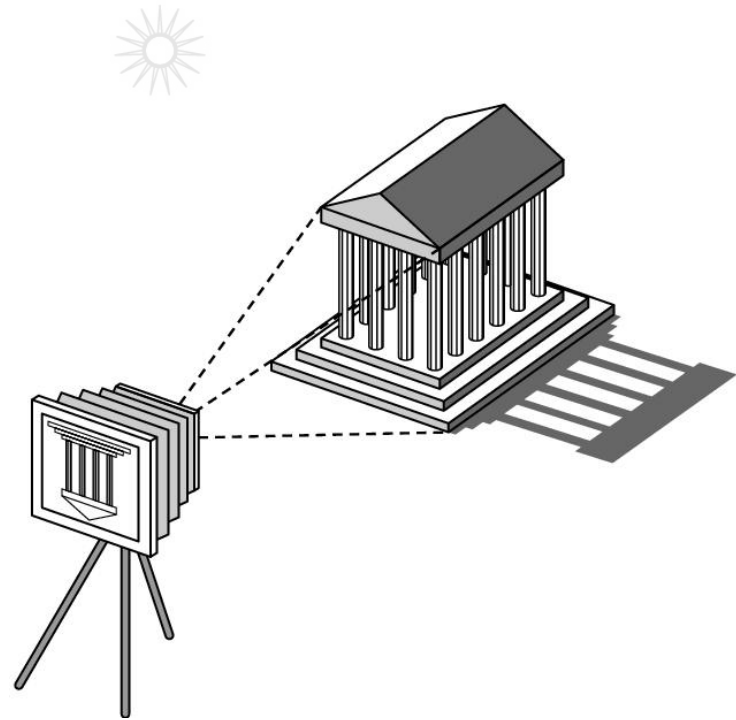
Solid Rendering

- `glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)`



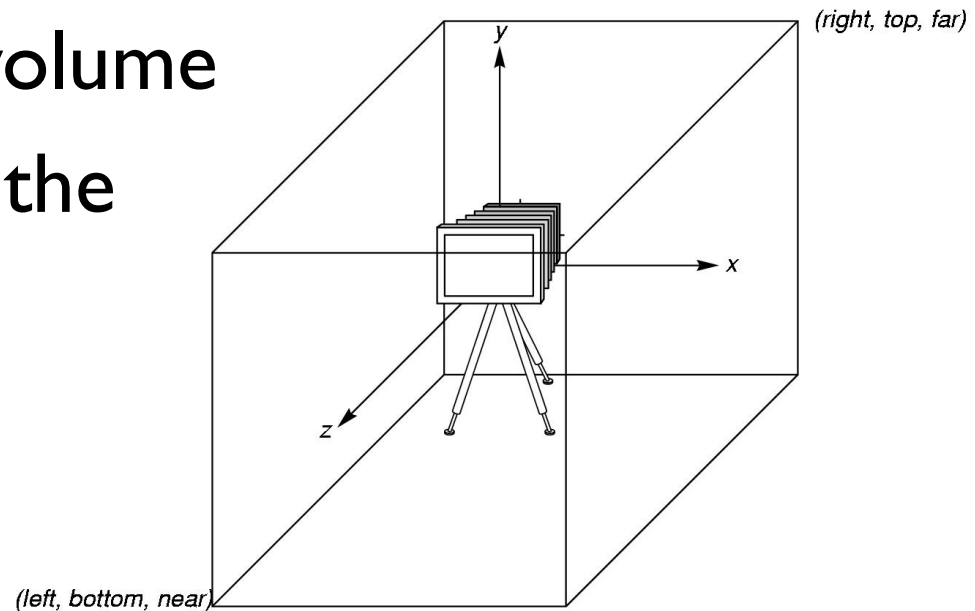
Today: Image Formation in GL

- Object
- Viewer
- Light sources



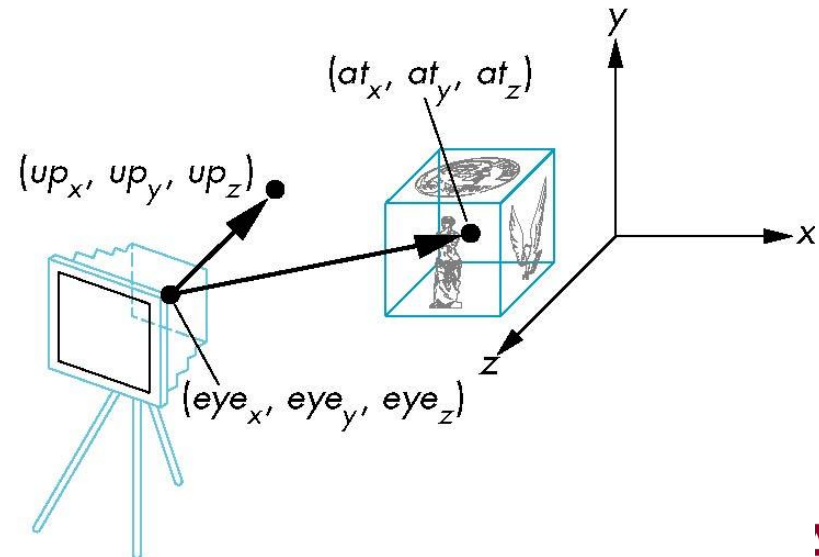
OpenGL Camera

- OpenGL places a camera at the **origin** in object space pointing in the **negative z** direction
- The default viewing volume is a box centered at the origin with sides of length 2



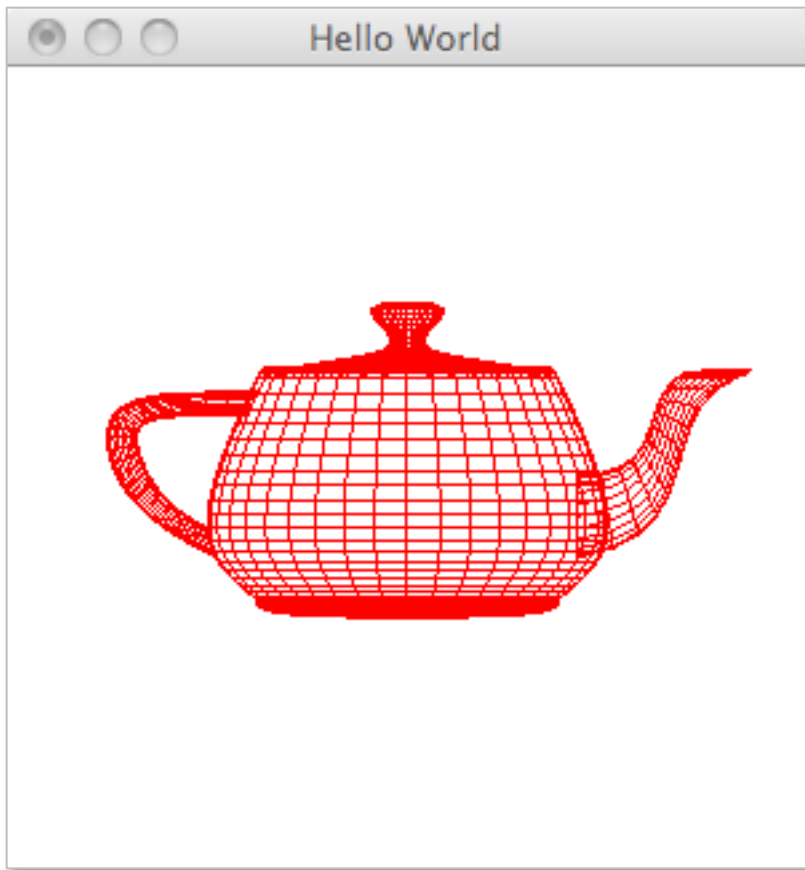
OpenGL Camera Commands

- Orientation
 - LookAt
- Field of view
 - Orthogonal/Perspective Projection
- Film/CCD
 - Buffers



LookAt

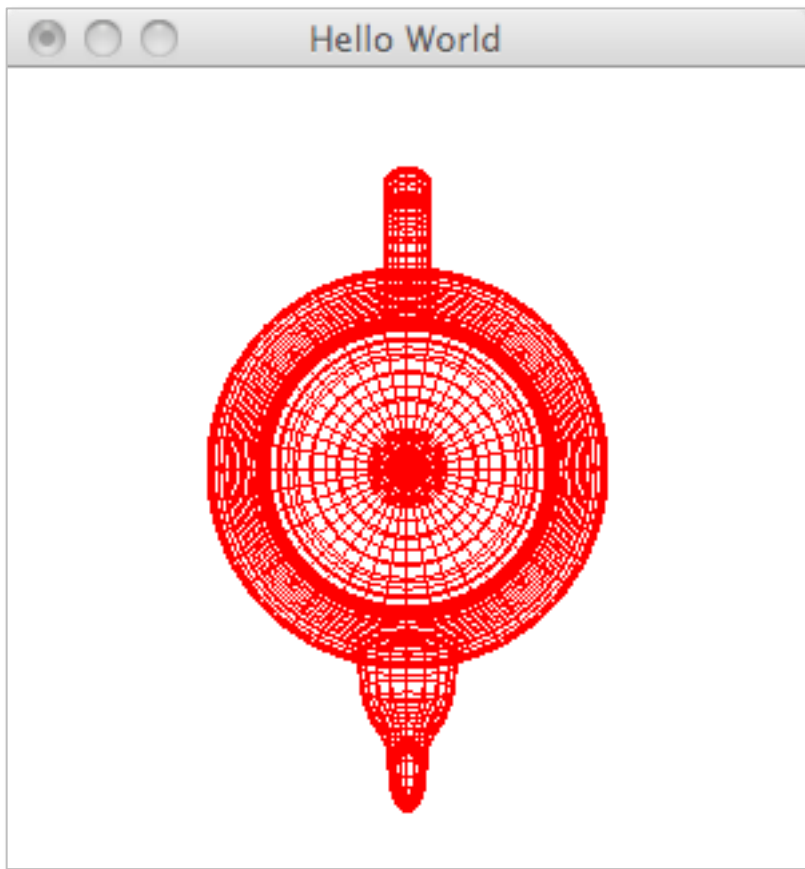
- Eye position, viewing direction, up vector



```
LookAt(0,0,100, 0,0,0, 0,1,0);
```

LookAt

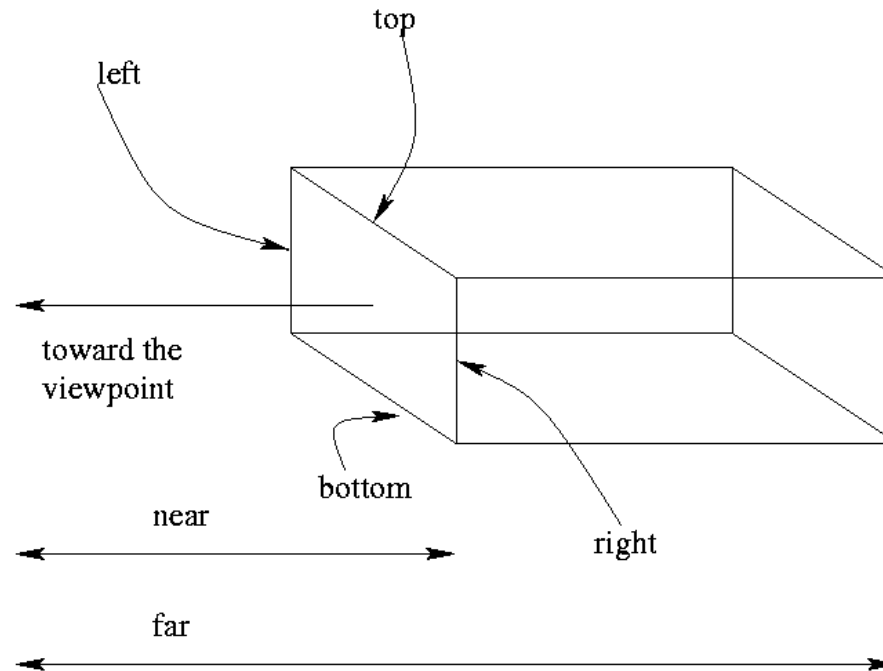
- Eye position, viewing direction, up vector



```
LookAt(0,100,0, 0,0,0, -1,0,0);
```

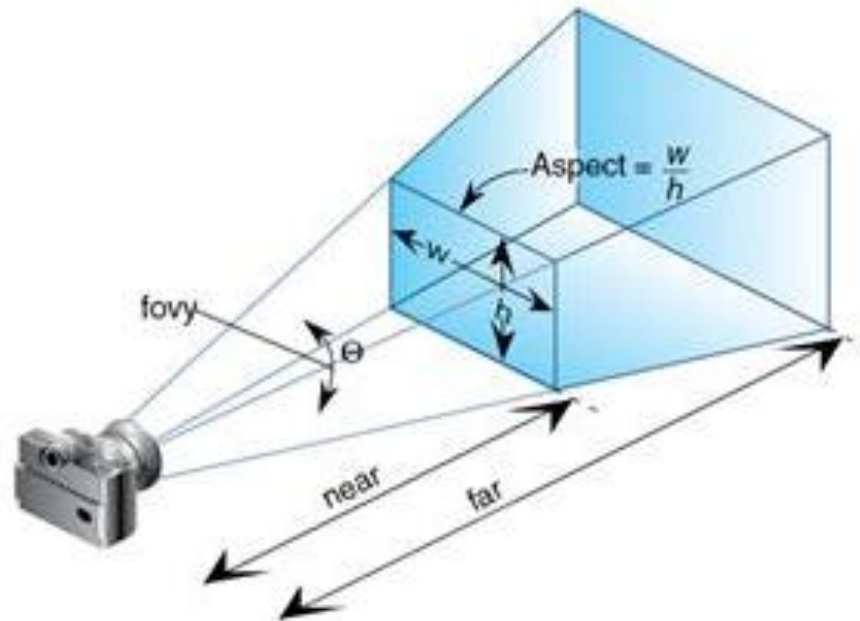

Orthogonal Projection

- left, right, bottom, top, near, far



Perspective Projection

- fovy, aspect, near, far



OpenGL Framebuffer

- Color buffers
 - Store pixel colors
 - Front, back, left, right, auxiliary
- Depth buffer
 - Store per-pixel depth value
- Stencil buffer
 - Mask for stencil test
- Accumulation buffer
 - Blend pixel values



Clear Buffers

- Set the clearing value for buffer
 - glClearColor, glClearIndex
 - glClearDepth
 - glClearStencil
 - glClearAccum
- Clear specific buffers
 - glClear(mask)
 - GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT...

```
glClearColor(1,1,1,0);  
glClear(GL_COLOR_BUFFER_BIT);  
  
glClearDepth(1.0);  
glClear(GL_DEPTH_BUFFER_BIT);
```



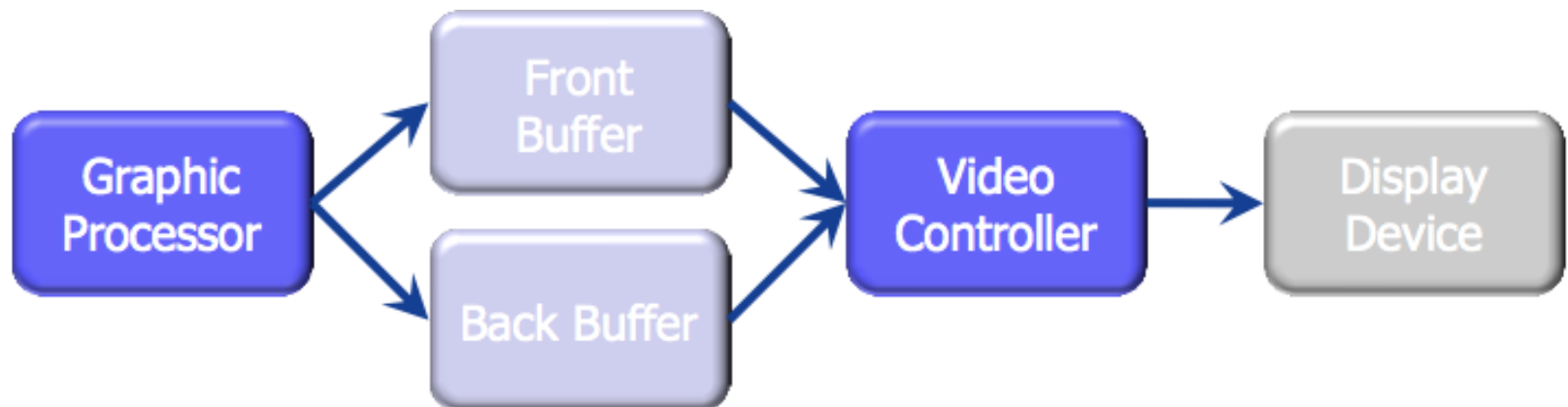
Select Buffers

- `glDrawBuffer(mode)`
 - Select target buffer to render onto
- `glReadBuffer(mode)`
 - Select buffer to read



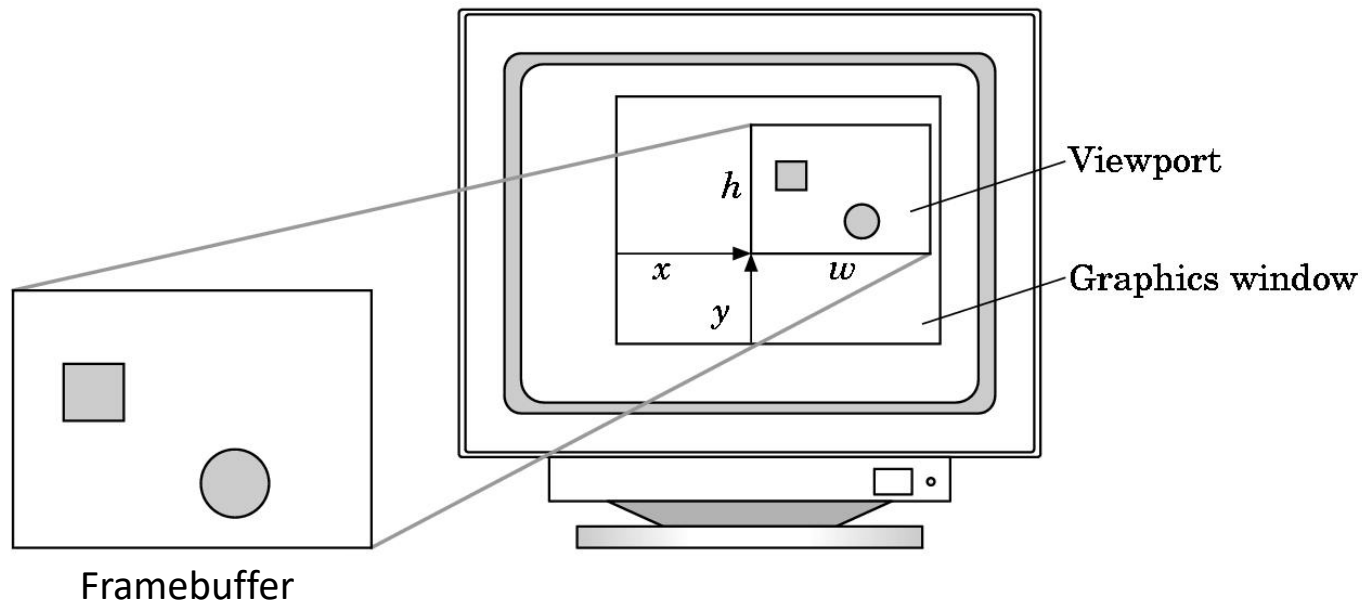
Double Buffering

- Rendering to framebuffer is slower than copying framebuffer to screen
- Use two buffers, GL_FRONT and GL_BACK
- Render to back buffer and swap
- Fast, reduce flickering and tearing



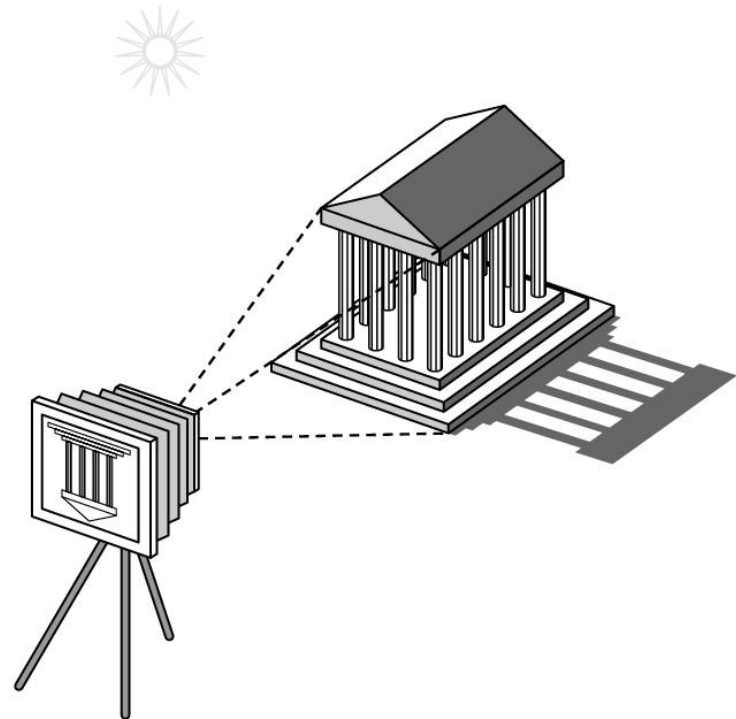
Viewport

- Mapping framebuffer to screen
 - `glViewport(x, y, w, h)`



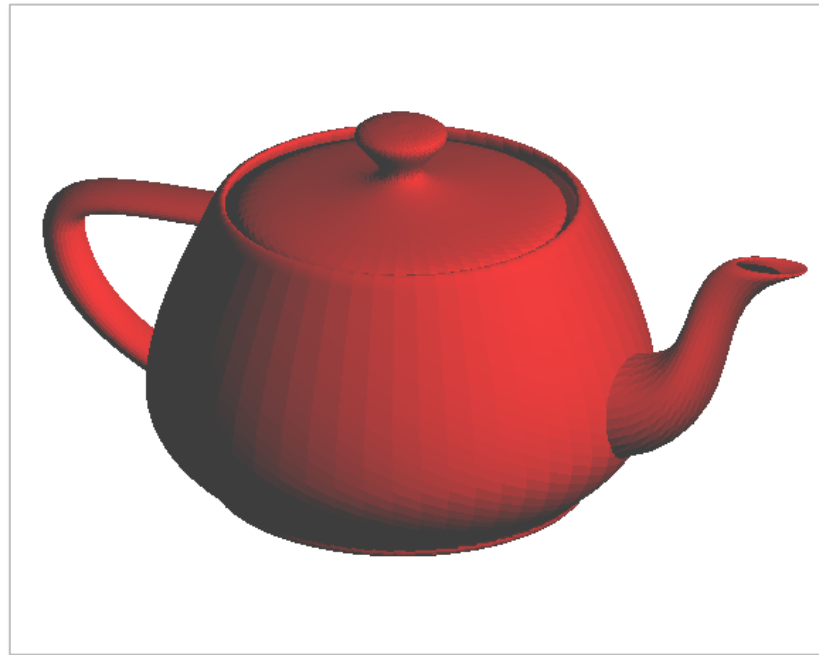
Today: Image Formation in GL

- Object
- Viewer
- Light sources



Shading Model

- Flat shading
 - Per-polygon shading
 - Constant color for a polygon



Shading Model

- Smooth shading
 - Per-vertex shading or per-pixel shading



Why?



Depth Test

- Triangle rendering order is arbitrary
- You must discard fragments that are not visible

```
// once  
glutInitDisplayMode(GLUT_DEPTH | ...);  
glEnable(GL_DEPTH_TEST);
```

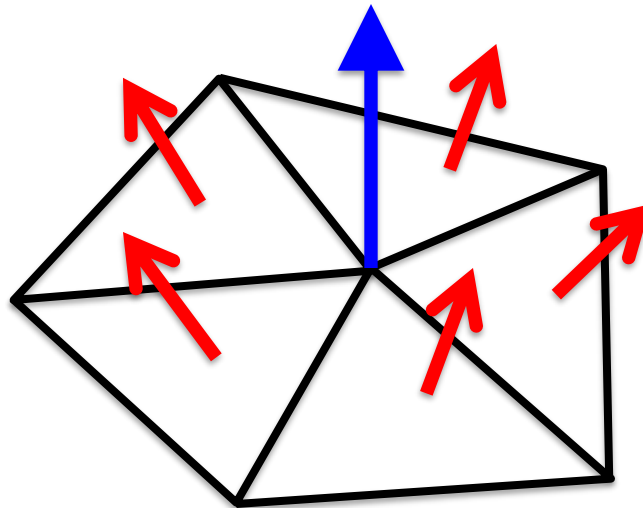
...

```
// Before drawing something  
glClearDepth(1.0);  
glClear(GL_DEPTH_BUFFER_BIT | ...)
```



Note

- You need to calculate per-vertex normals and assign them as vertex attribute
- Per-vertex normal can be an average of neighbor triangle normals



Questions?



"Brave" Image courtesy of Disney/Pixar