UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4370 Fall 2023

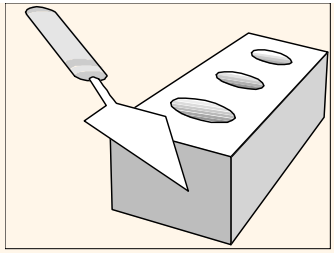
Interactive Computer Graphics

M & W 5:30 to 7:00 PM

Prof. Victoria Hilford

PLEASE TURN your webcam ON

NO CHATTING during LECTURE



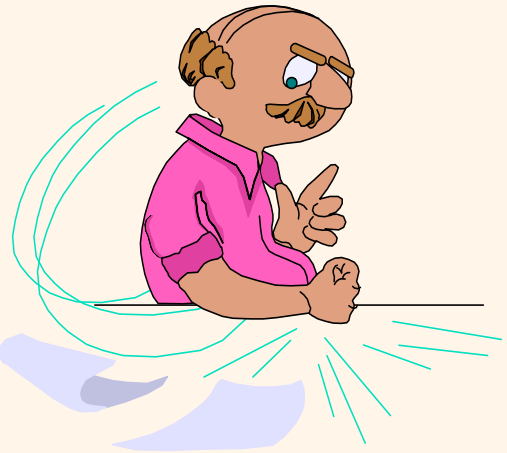
COSC 4370

5:30 to 7

PLEASE

LOG IN

CANVAS



Please close all other windows.

From 5:30 to 6:25 PM – 55 minutes.

08.28.2023 (M 5:30 to 7) (3)	Homework 1	Lecture 2
08.30.2023 (W 5:30 to 7) (4)		Math Review 1
09.06.2023 (W 5:30 to 7) (5)	Homework 2	Lecture 3
09.11.2023 (M 5:30 to 7) (6)		Math Review 2
09.13.2023 (W 5:30 to 7) (7)	Homework 3	Lecture 4
09.18.2023 (M 5:30 to 7) (8)		PROJECT 1
09.20.2023 (W 5:30 to 7) (9)		EXAM 1 REVIEW
09.25.2023 (M 5:30 to 7) (10)		EXAM 1

COSC 4370 – Computer Graphics

Lecture 2

Chapter 2

Objectives

- Development of the OpenGL API
- OpenGL Architecture

  OpenGL as a **state machine**
- API Functions

  Types

  Formats

1. Primitive functions
2. Attribute functions
3. Viewing functions
4. Transformation functions
5. Input functions
6. Control functions
7. Query functions

- Sample programs

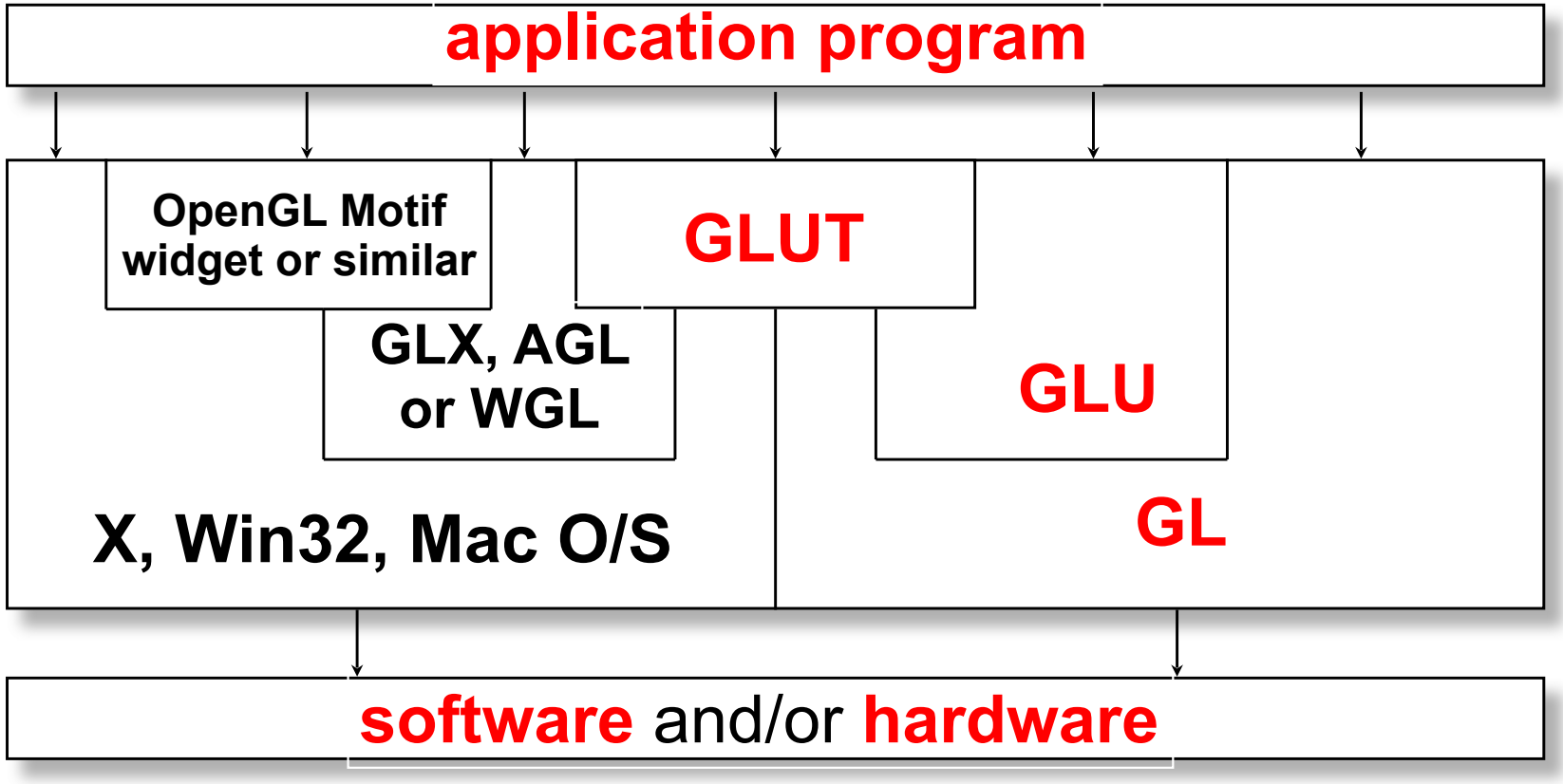
GLUT

OpenGL Utility Toolkit (GLUT)

Provides functionality common to all **window systems**

- Open a window
- **Get input from mouse and keyboard**
- Menus
- **Event-driven**

Software Organization



Abstractions

GL

- Primitives - points, line, polygons
- Shading and Colour
- Translation, rotation, scaling
- Viewing, Clipping, Texture
- Hidden surface removal

Abstractions

GLU

- Viewing –perspective/orthographic
- Image scaling, polygon tessellation
- Sphere, cylinders, quadratic surfaces

GL

- Primitives - points, line, polygons
- Shading and Colour
- Translation, rotation, scaling
- Viewing, Clipping, Texture
- Hidden surface removal

Abstractions

GLUT

- Windowing toolkit (key, mouse handler, window events)

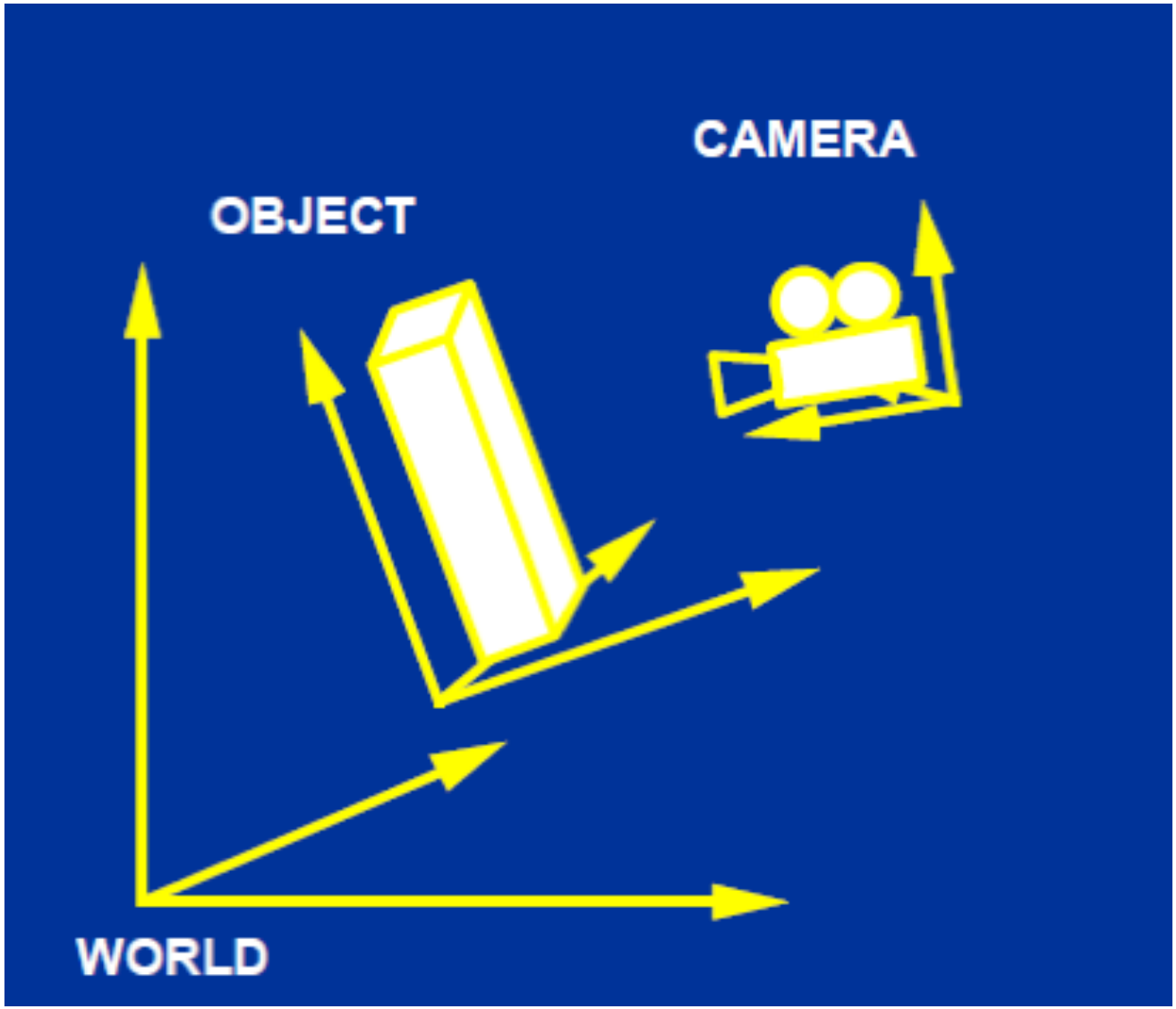
GLU

- Viewing –perspective/orthographic
- Image scaling, polygon tessellation
- Sphere, cylinders, quadratic surfaces

GL

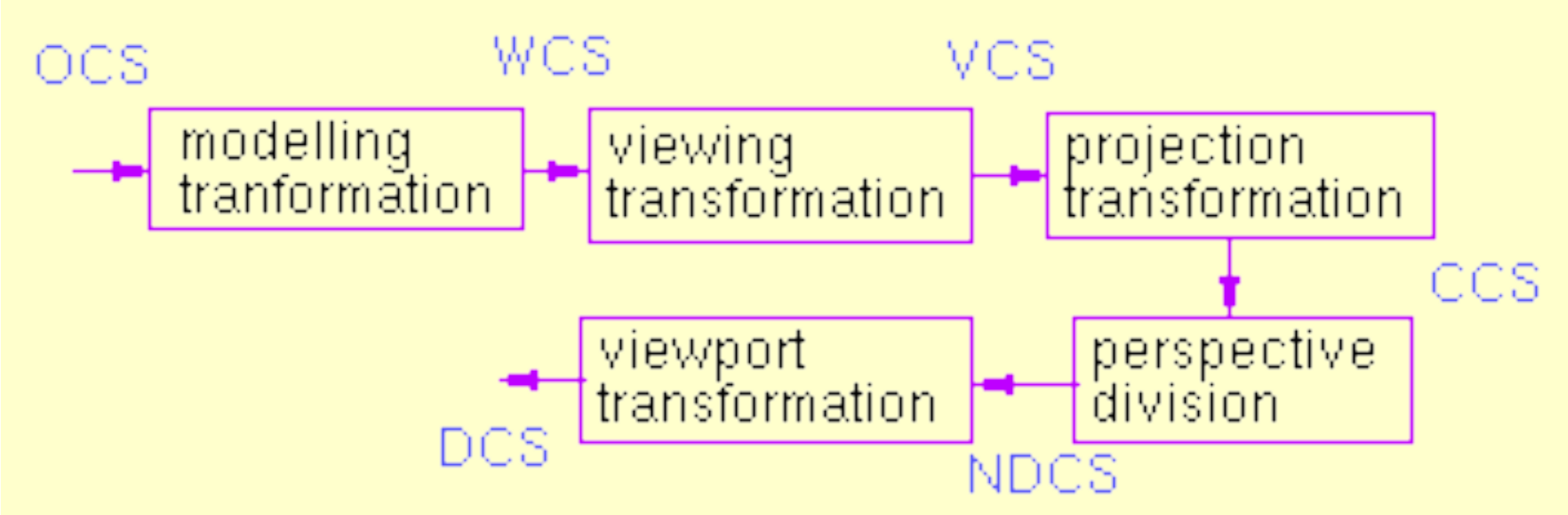
- Primitives - points, line, polygons
- Shading and Colour
- Translation, rotation, scaling
- Viewing, Clipping, Texture
- Hidden surface removal

The Object, the World and Camera Frames



Review Transformations

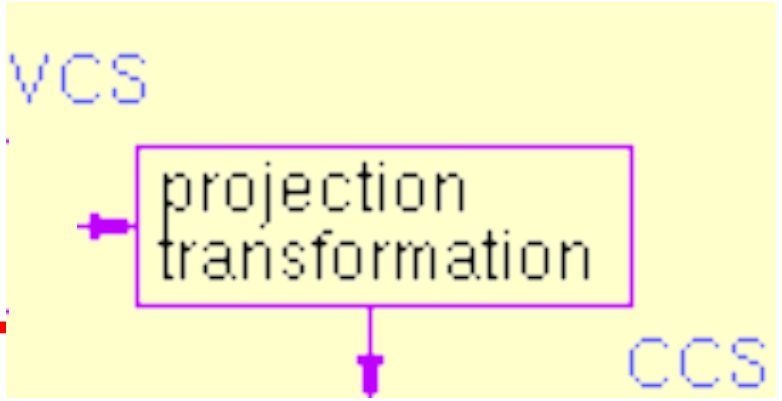
Stages of Vertex Transformations



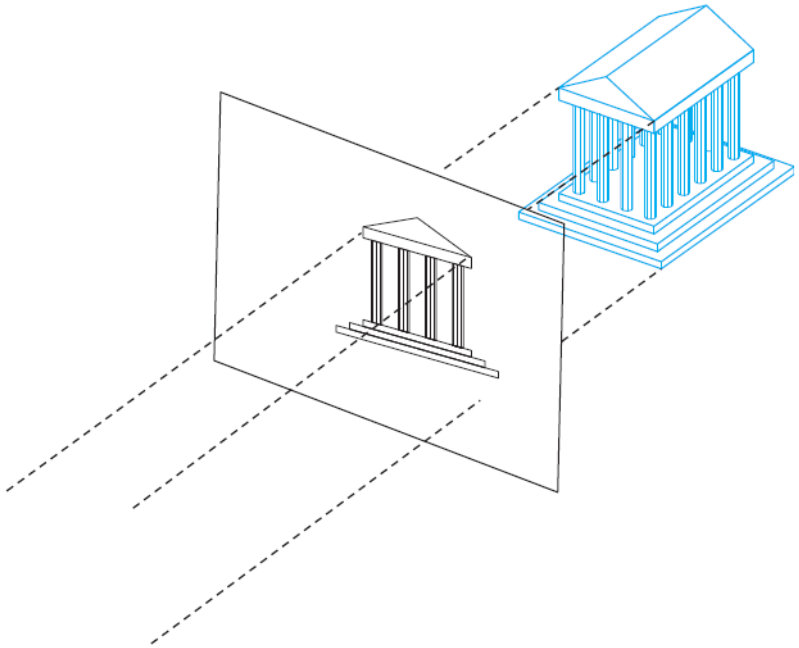
OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

Orthographic Projection

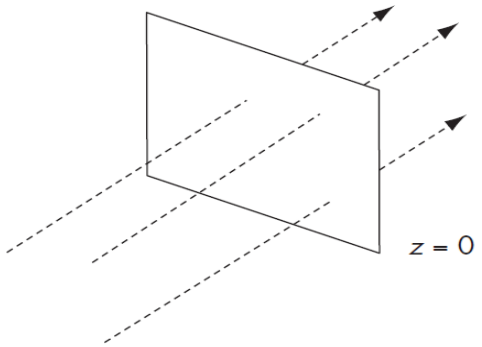
The University of New Mexico



VCS - viewing coordinate system
CCS - clipping coordinate system

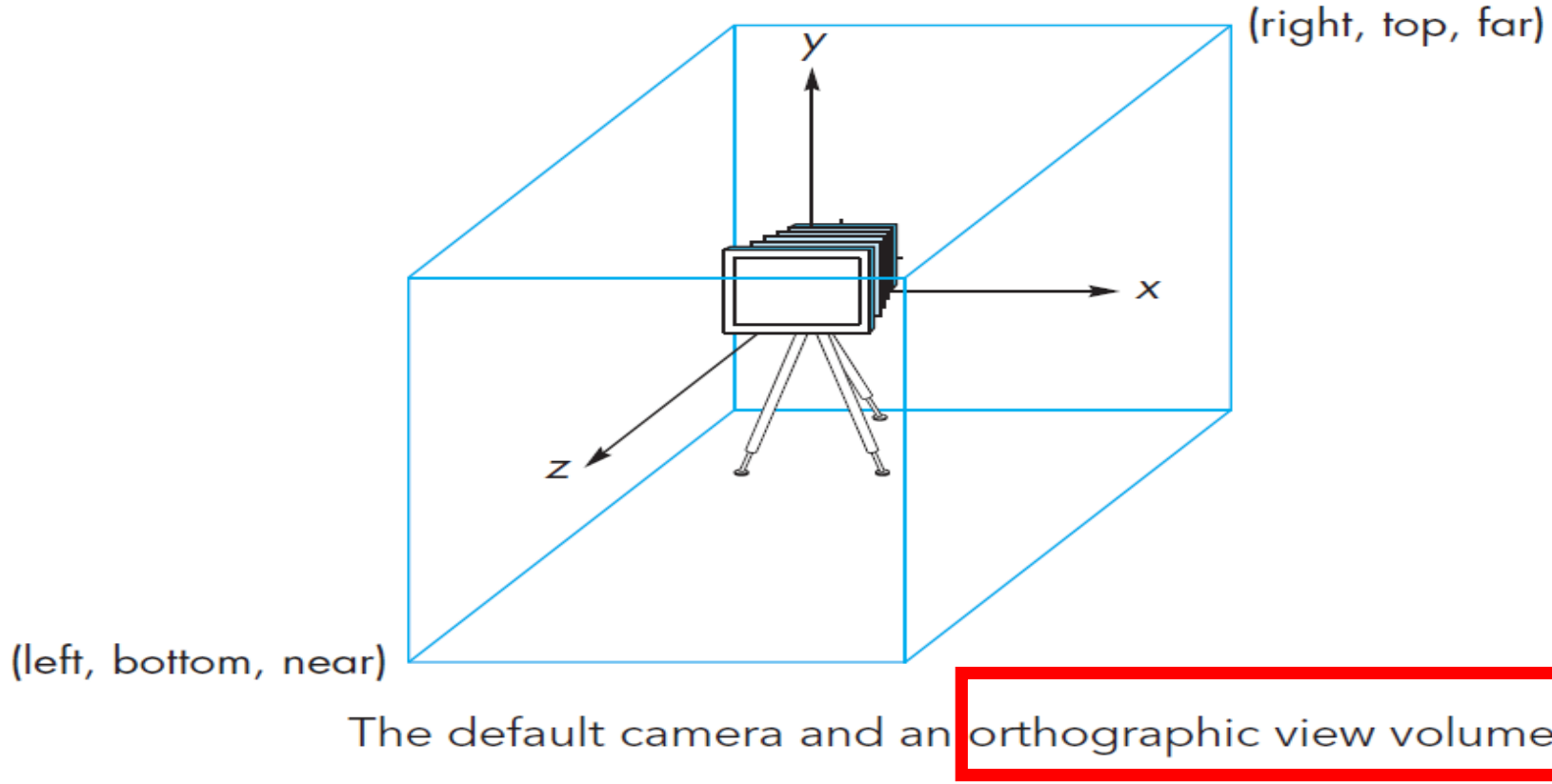


Creating an orthographic view by moving the camera away from the projection plane.



Orthographic projectors with projection plane $z = 0$.

Orthographic Projection



The default camera and an orthographic view volume

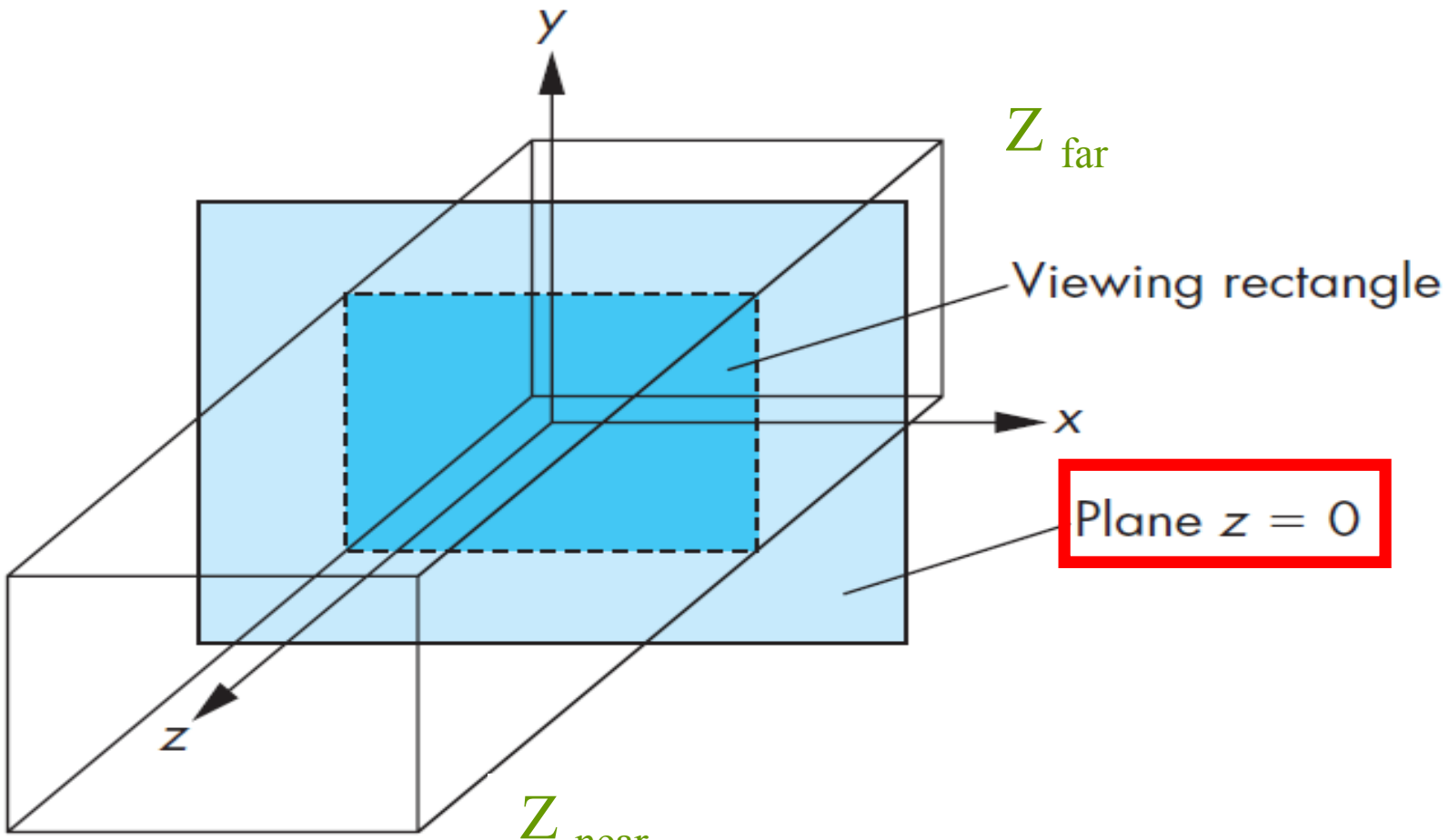
x - axis

y - axis

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

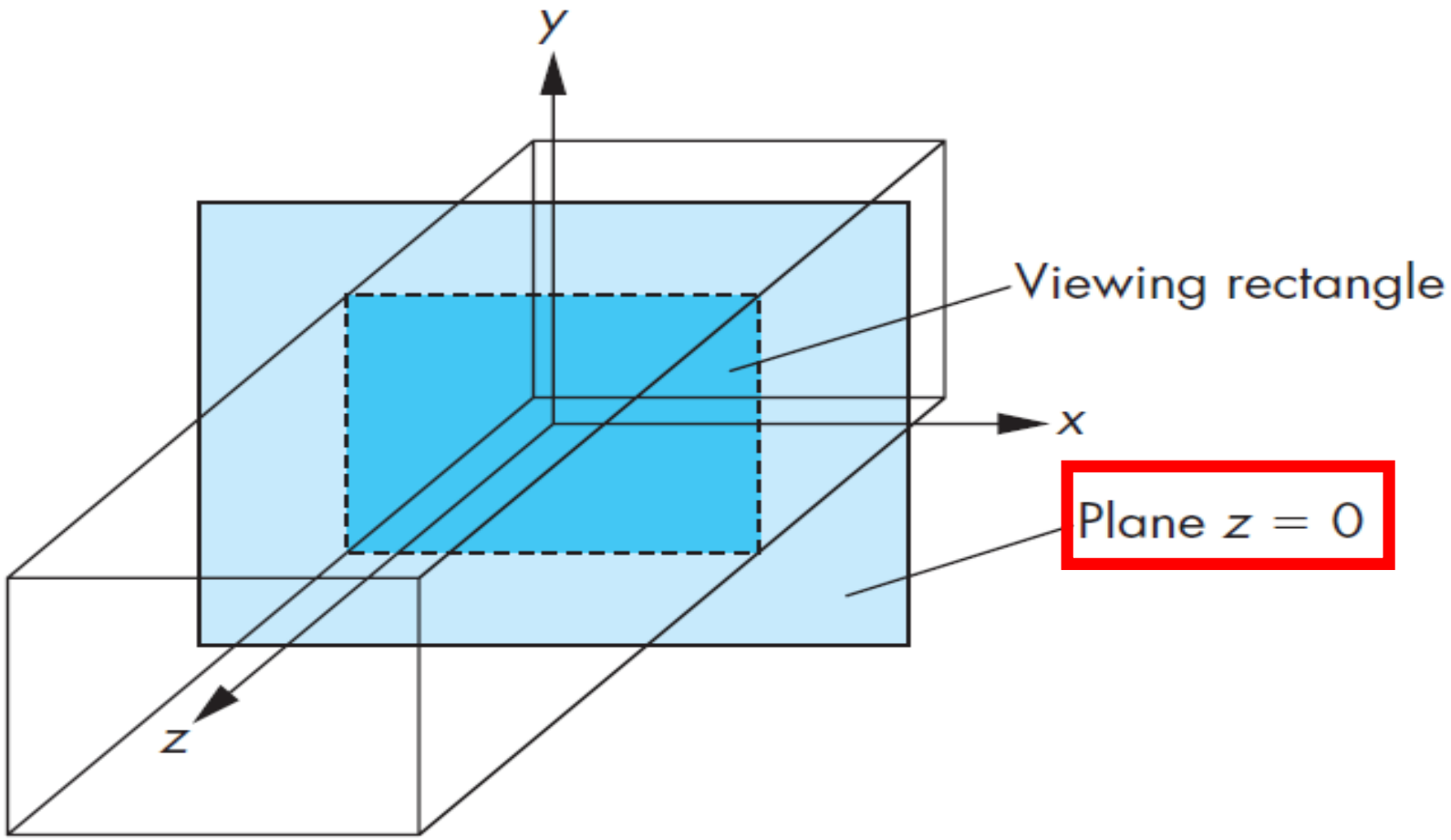
z - axis

Orthographic Projection



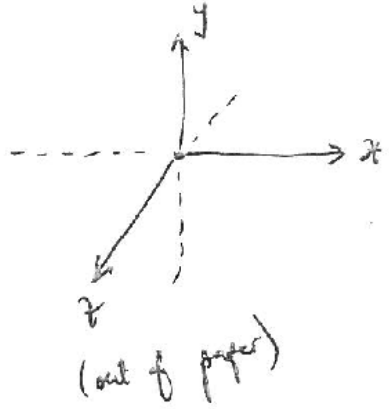
Viewing volume.

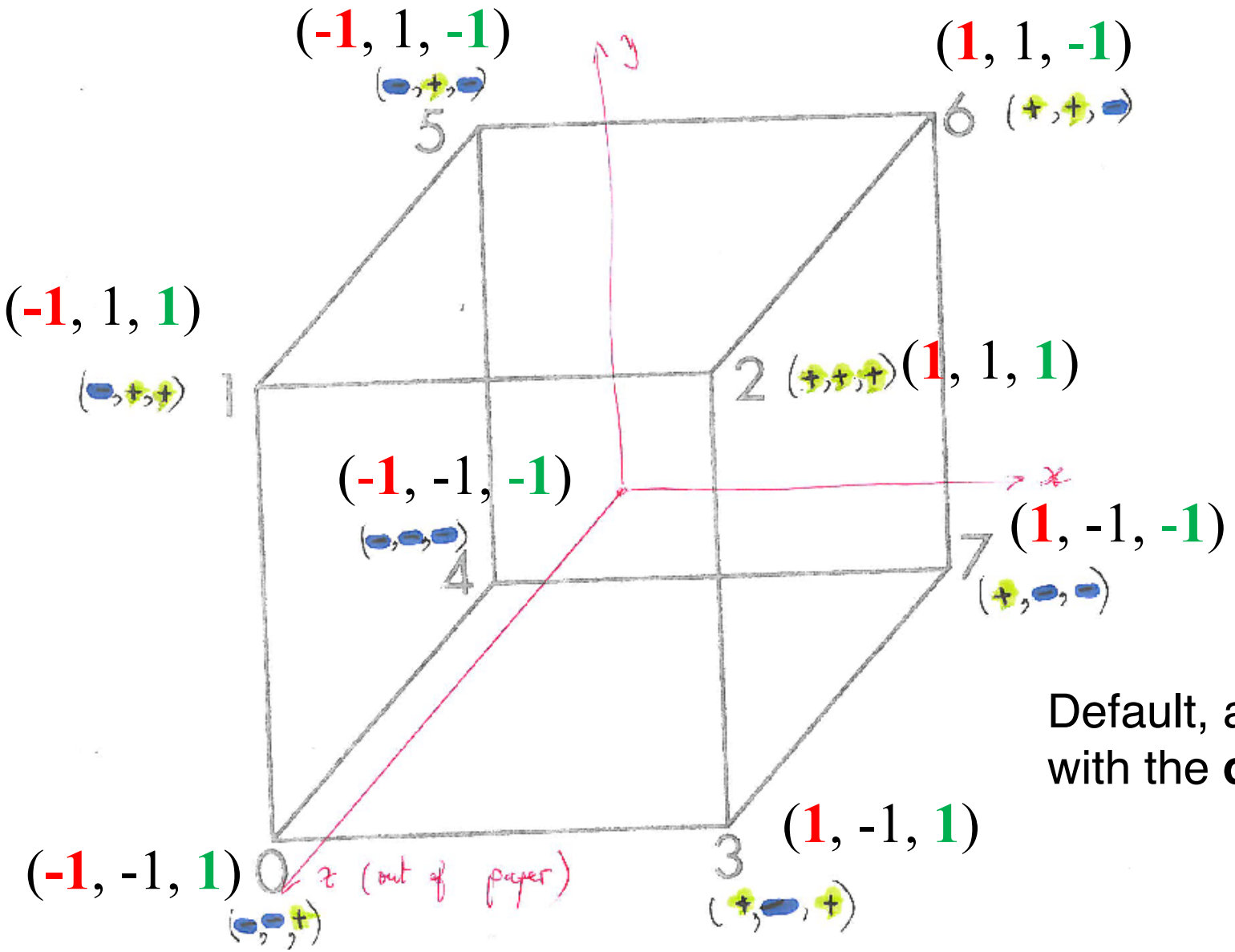
Orthographic Projection



Viewing volume.

Orthographic Projection
OpenGL Default





Default, a **2 × 2 × 2 cube**, with the **origin in the center**.

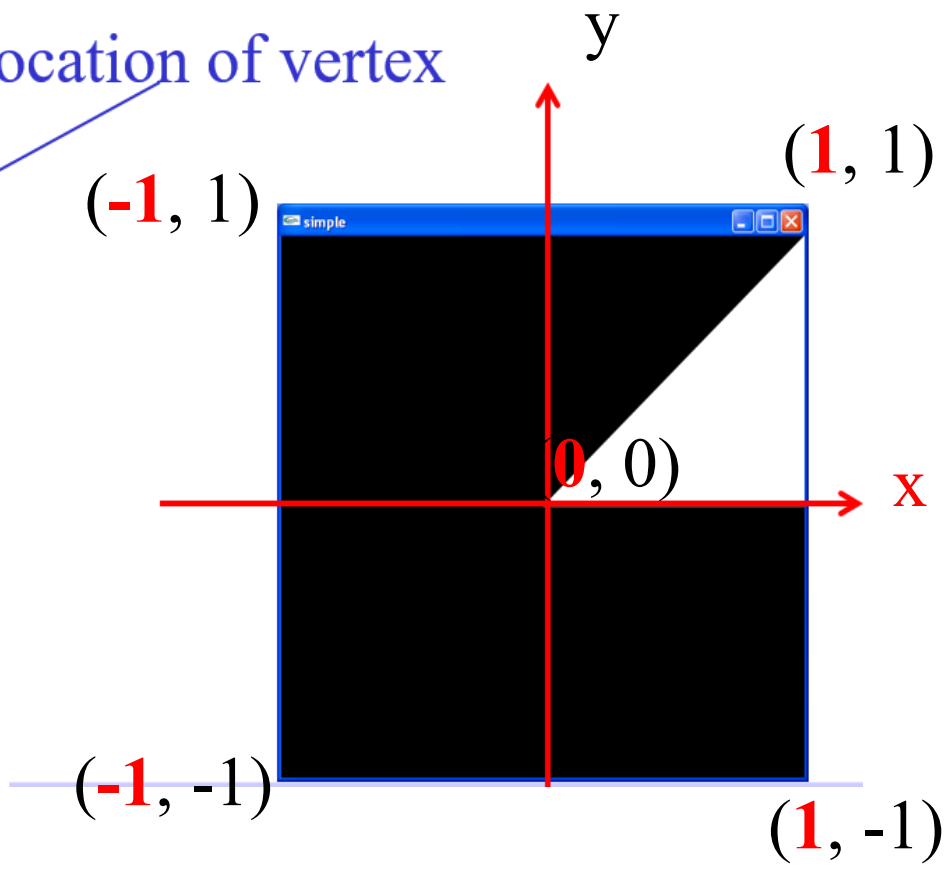
Orthographic Projection
OpenGL Default (2D)

type of object

location of vertex

```
glBegin(GL_TRIANGLES);
 glVertex2f(0.0, 0.0);
 glVertex2f(1.0, 1.0);
 glVertex2f(1.0, 0.0);
glEnd( );
```

end of object definition

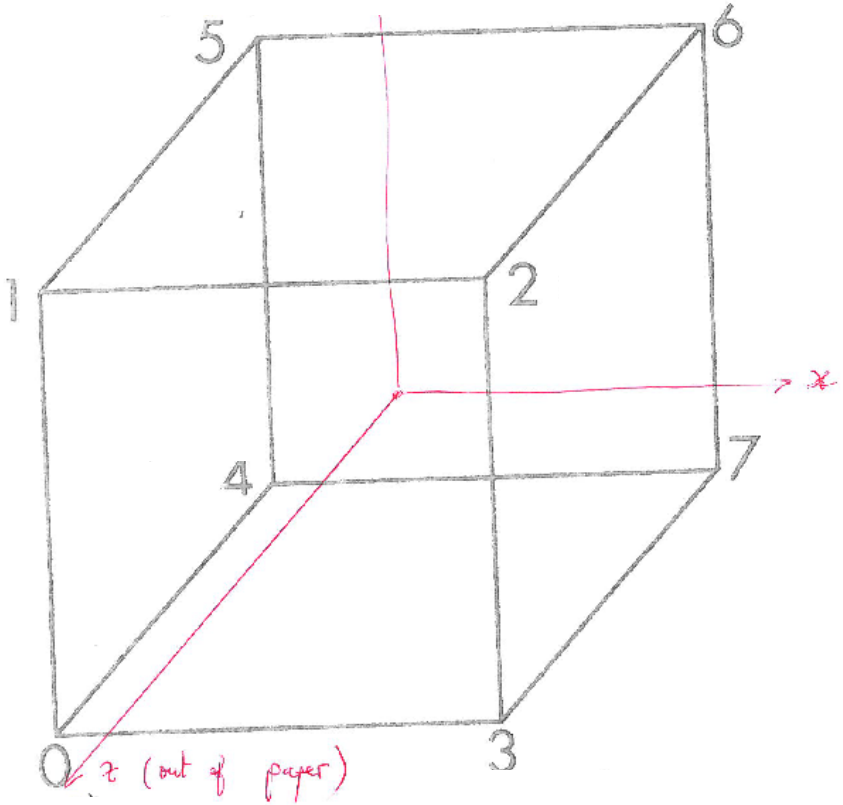


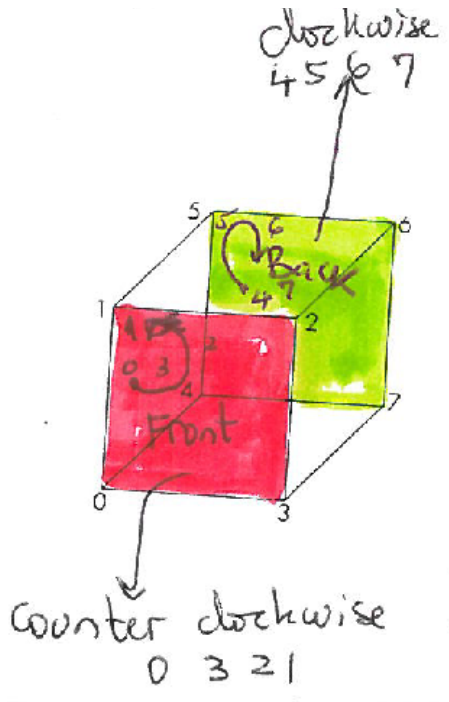

```
void gluOrtho2D(GLdouble left, GLdouble right,
   GLdouble bottom, GLdouble top)

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

Orthographic Projection
OpenGL Default

Each **face** of the cube is using a polygon of the following colors:

back side (`glColor3f(0.0, 1.0, 0.0); //green`





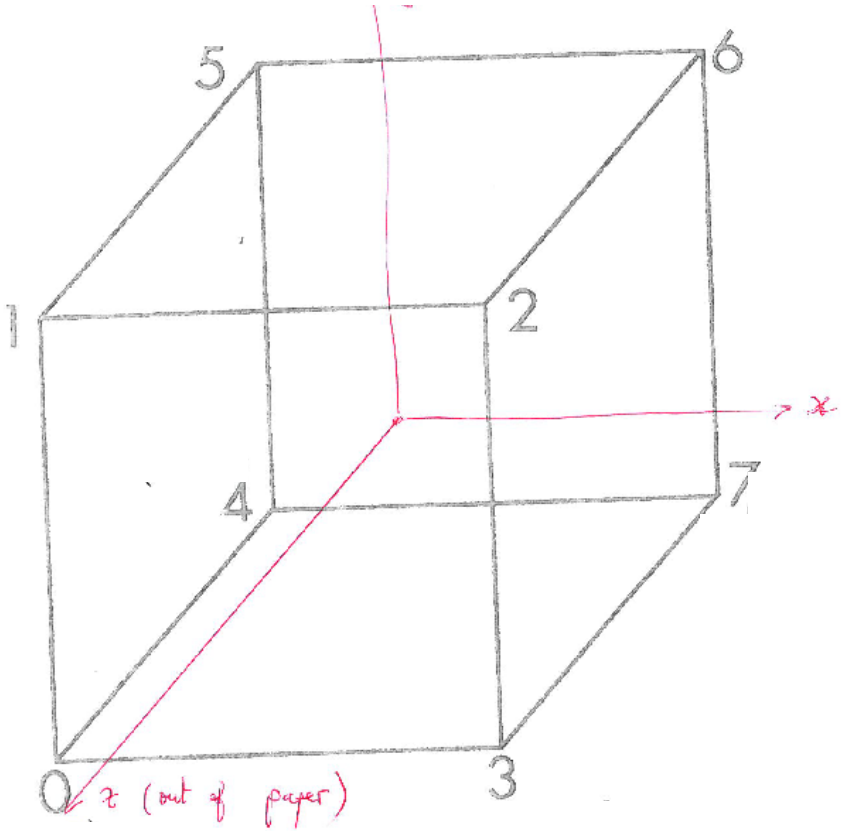
front side (`glColor3f(1.0, 0.0, 0.0); //red`

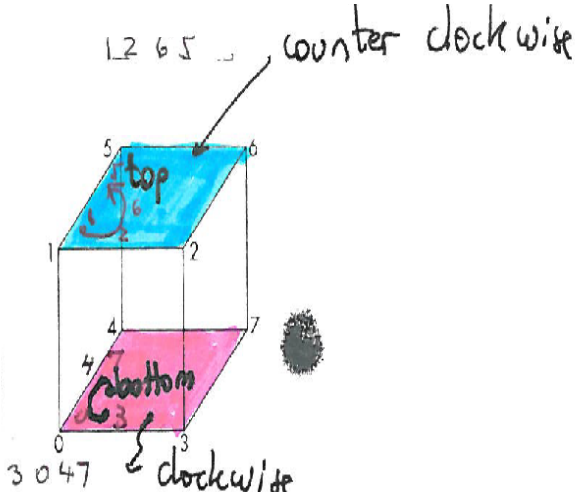
```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
    GLdouble top, GLdouble near, GLdouble far)
```

Orthographic Projection
OpenGL Default

Each **face** of the cube is using a polygon of the following colors:

top side (glColor3f(0.0, 0.75388, 1.0); //blue





bottom side (glColor3f(1.0, 0.2, 0.7); //deep pink

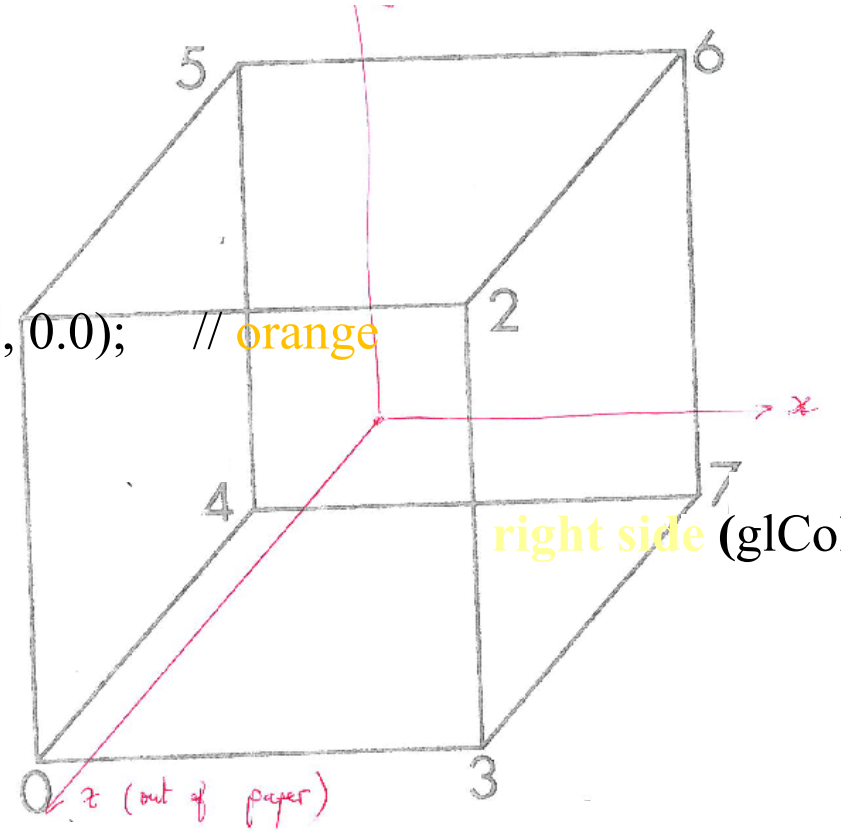
```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
    GLdouble top, GLdouble near, GLdouble far)
```

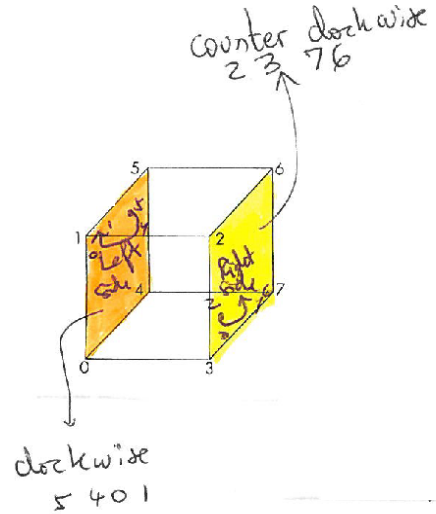
The University of New Mexico

Orthographic Projection
OpenGL Default

Each **face** of the cube is using a polygon of the following colors:

left side (glColor3f(1.0, 0.75, 0.0); // orange

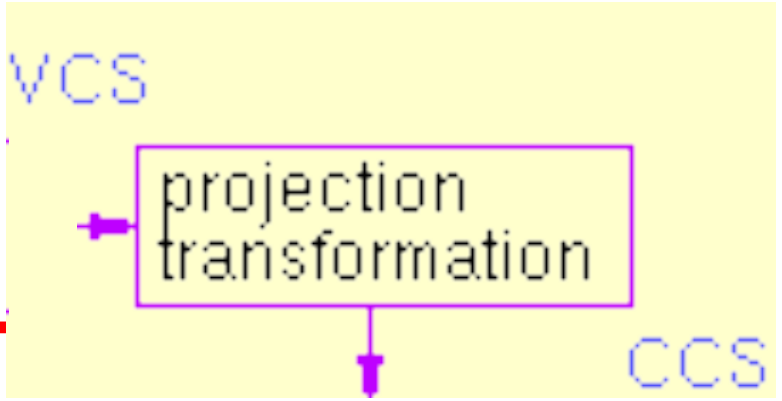




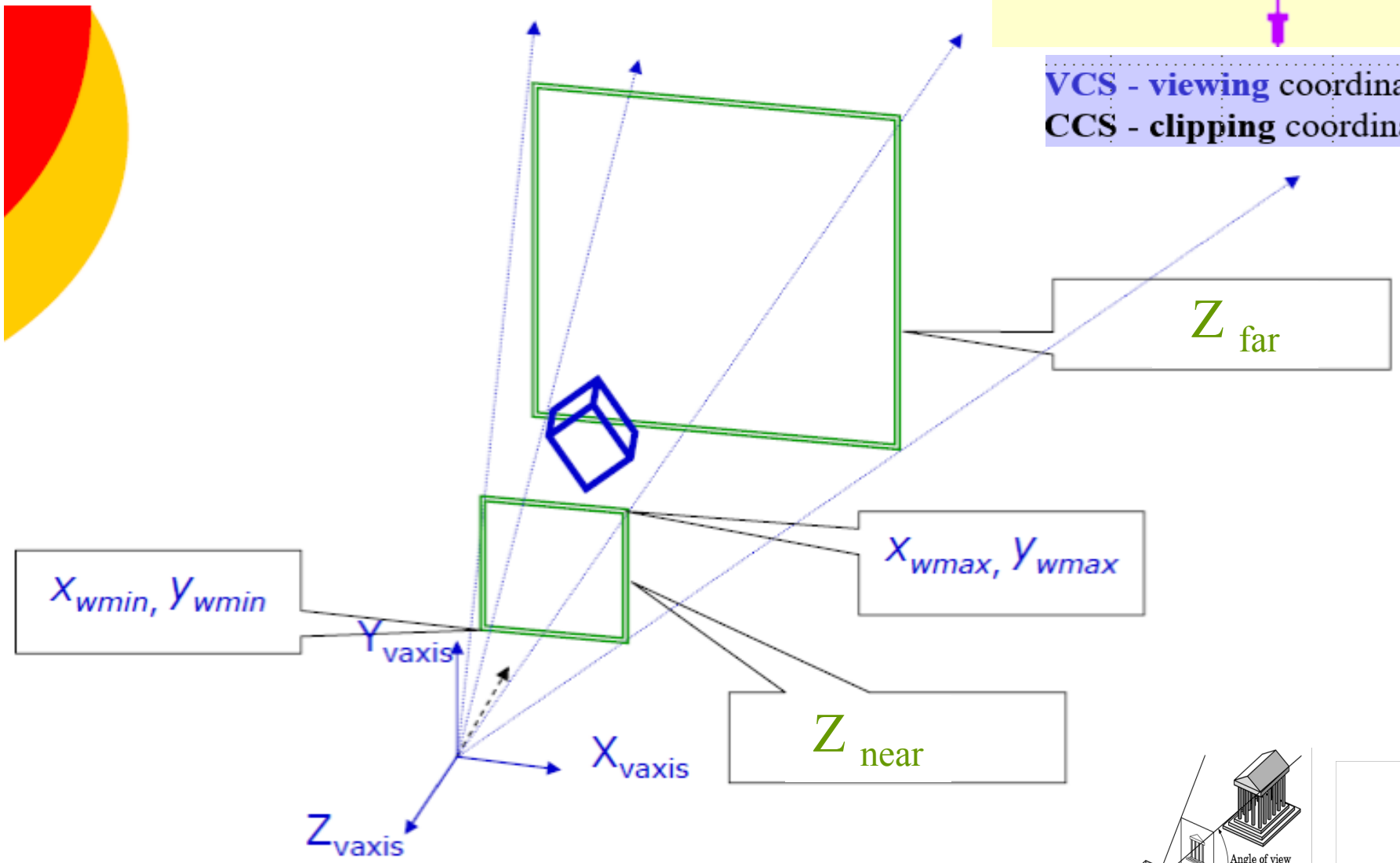
right side (glColor3f(1.0, 1.0, 0.0); //Yellow

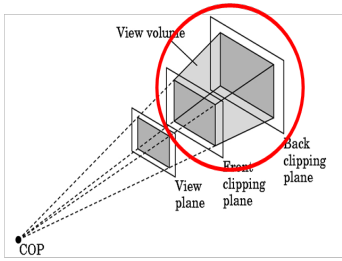
```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
    GLdouble top, GLdouble near, GLdouble far)
```

Perspective Projection



VCS - viewing coordinate system
CCS - clipping coordinate system

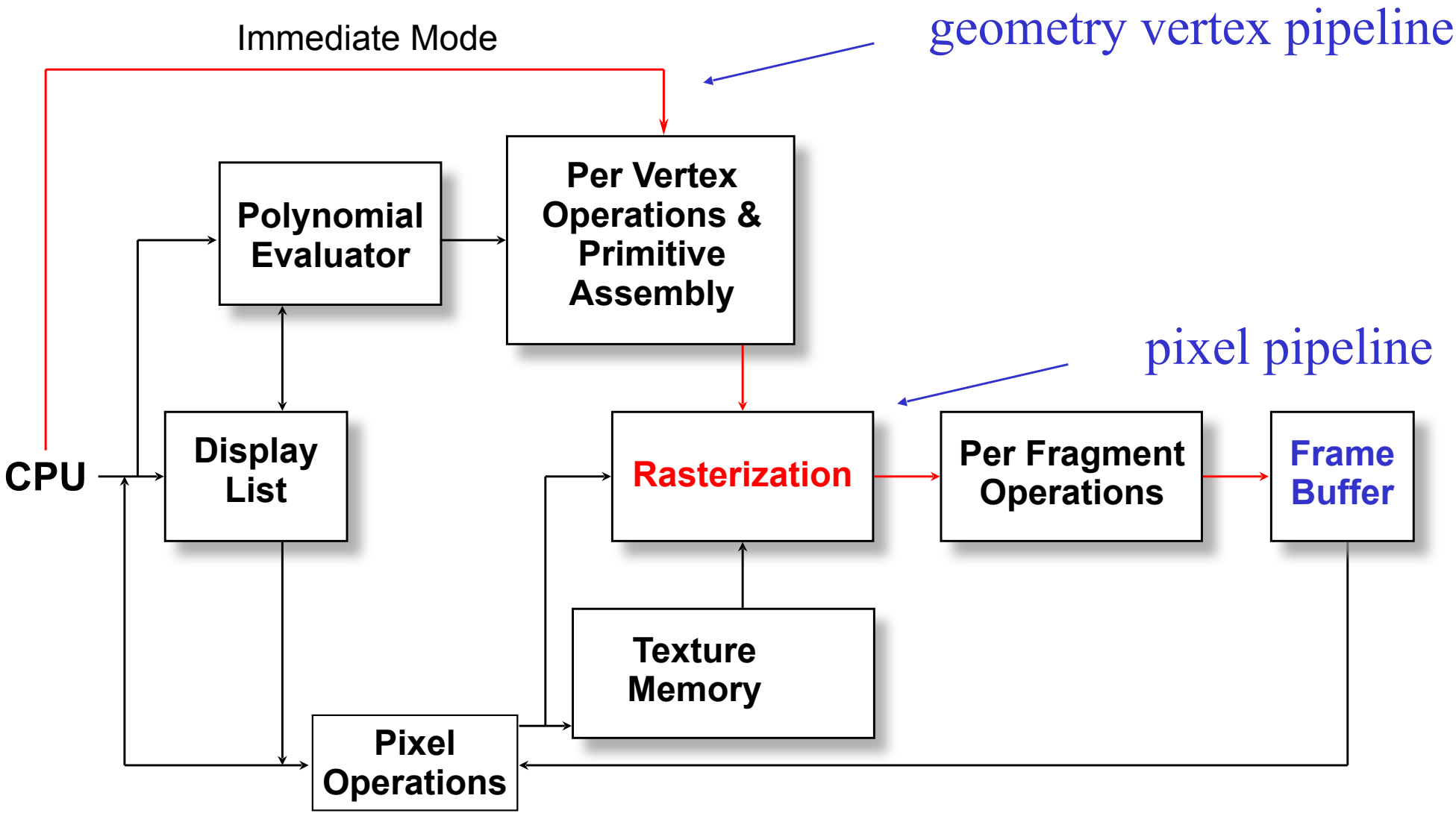




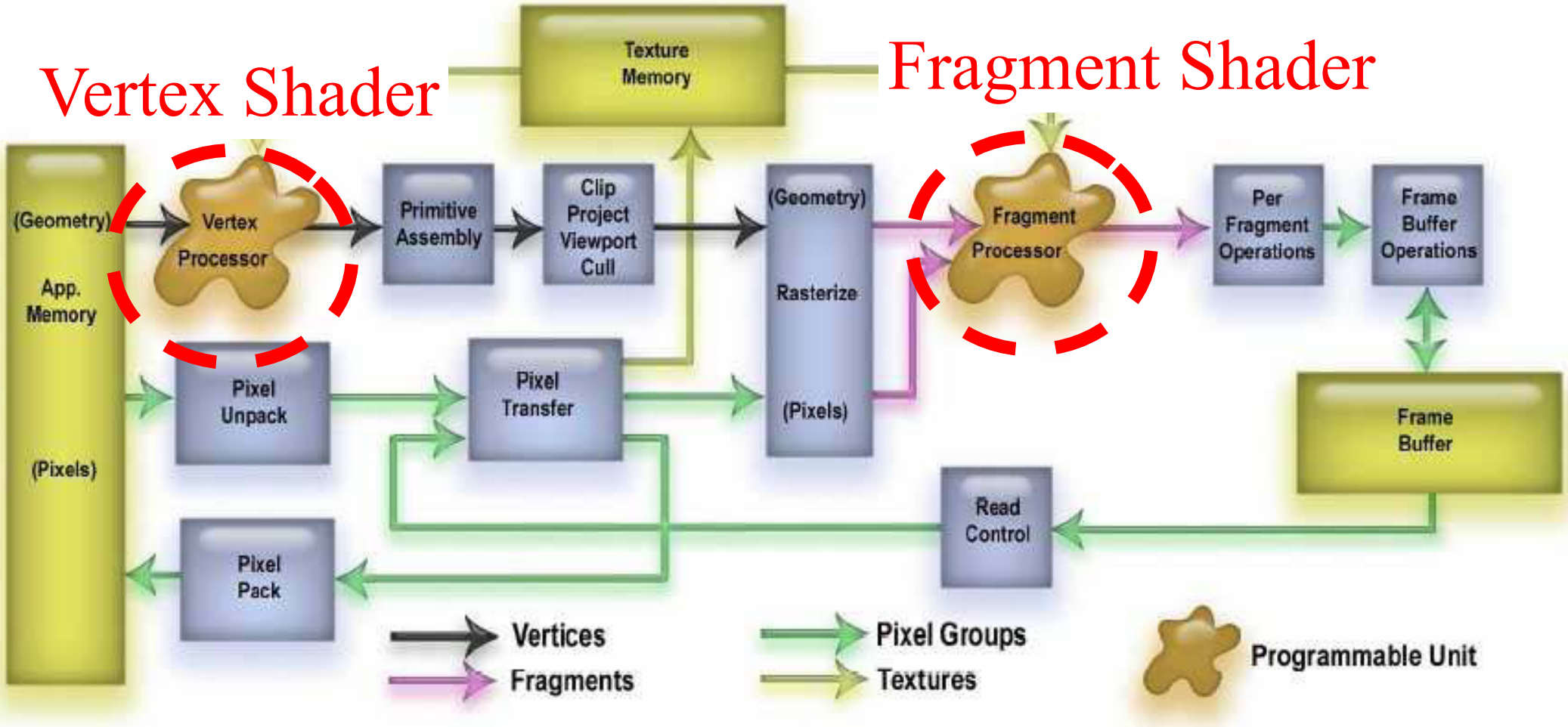
Objectives

- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a **State Machine**
- Functions
  - Types
  - Formats
- Simple program

OpenGL Architecture

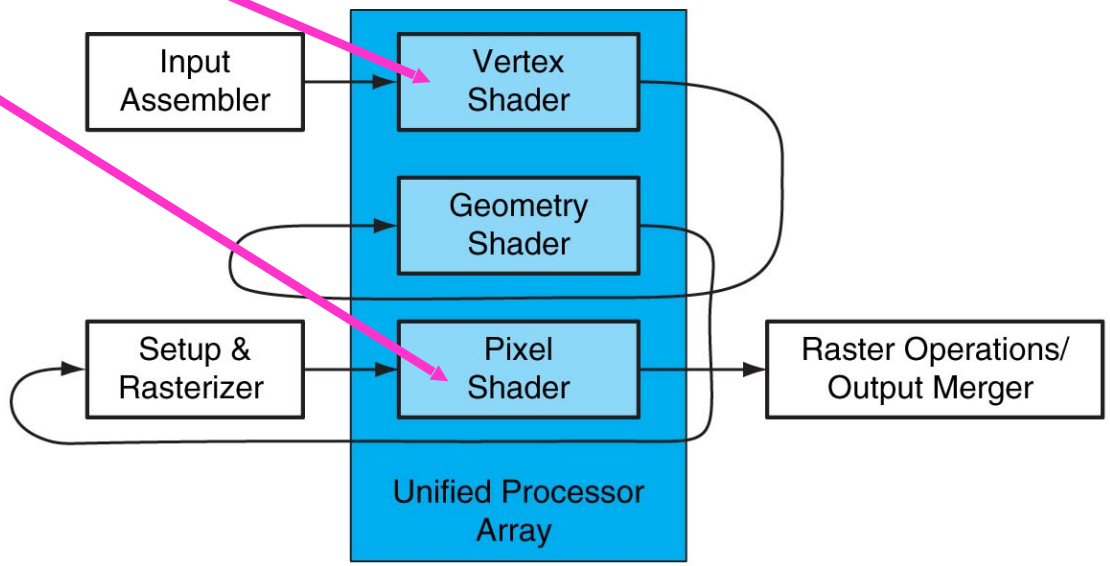


OpenGL Logical Diagram



Programmable Pipelines

- Text: Chapter 9
  - Shading Languages
  - GLSL (OpenGL Shading Language)
  - Vertex Shaders
  - Fragment (Pixel) Shaders

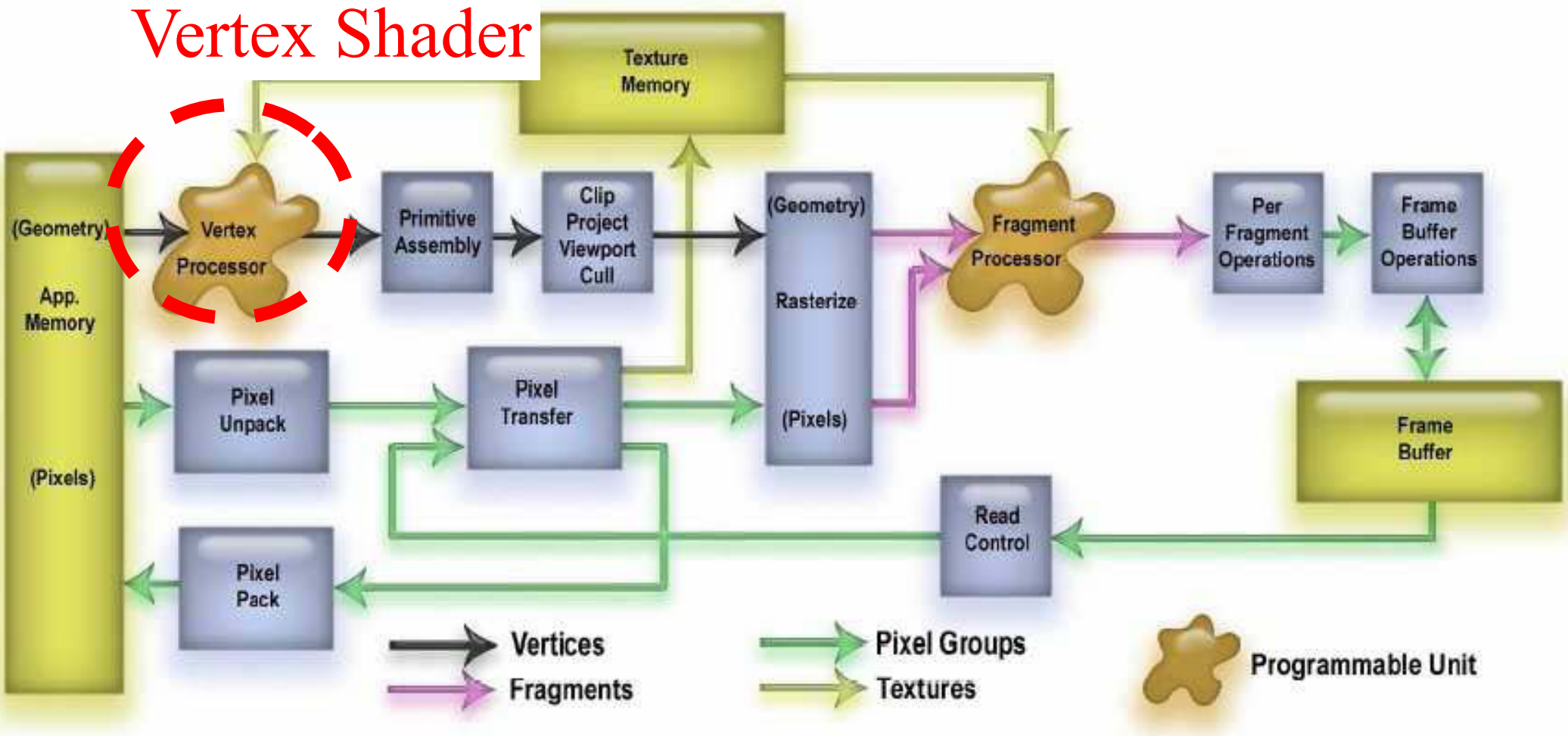


Programmable Components

- Shader: programmable processors.
  - Replacing fixed-function vertex and fragment processing.
- Shaders:
  - Vertex shaders
    - Dealing with per-vertex functions.
    - We can control the lighting and position of each vertex.
  - Fragment shaders
    - Dealing with per-pixel functions.
    - We can control the color of each pixel by user-defined programs.

The University of New Mexico

OpenGL Logical Diagram



Vertex Shaders

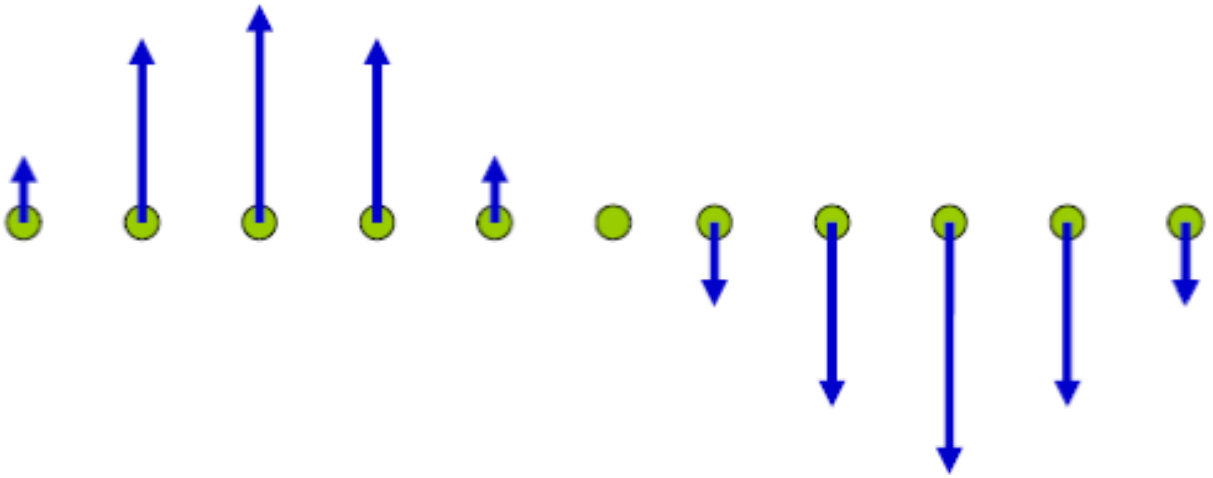
- Per-vertex calculations performed here
  - Without knowledge about other vertices (parallelism)
  - Your program take responsibility for:
    - Vertex transformation
    - Normal transformation
    - (Per-Vertex) Lighting
    - Color material application and color clamping
    - Texture coordinate generation

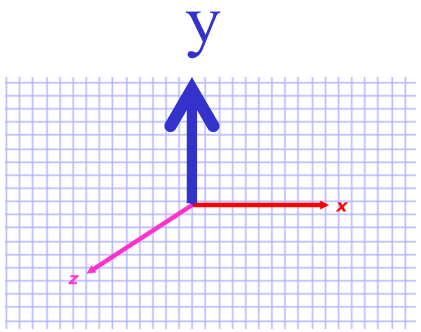
Vertex Shader Applications

- We can control movement with uniform variables and vertex attributes
  - Time
  - Velocity
  - Gravity
- Moving vertices
  - Morphing
  - Wave motion
  - .....
- Lighting
  - More realistic models
  - Cartoon shaders

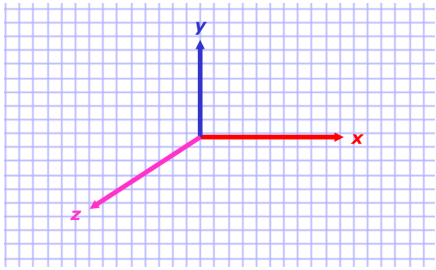
Vertex Shader Application: Wave Motion Vertex Shader

```
uniform float time;
uniform float xs, zs;
void main()
{
float s;
s = 1.0 + 0.1*sin(xs*time)*sin(zs*time);
gl_Vertex.y = s*gl_Vertex.y;

gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

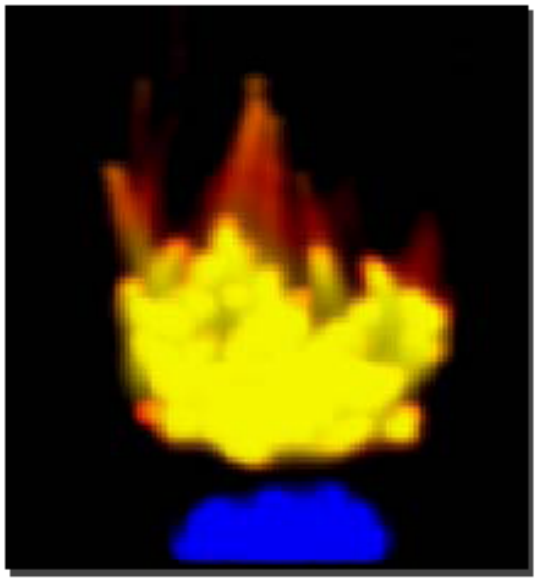


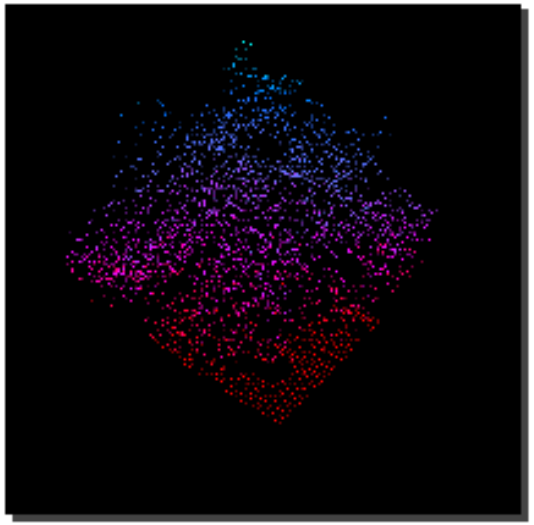


Vertex Shader Application: Particle System



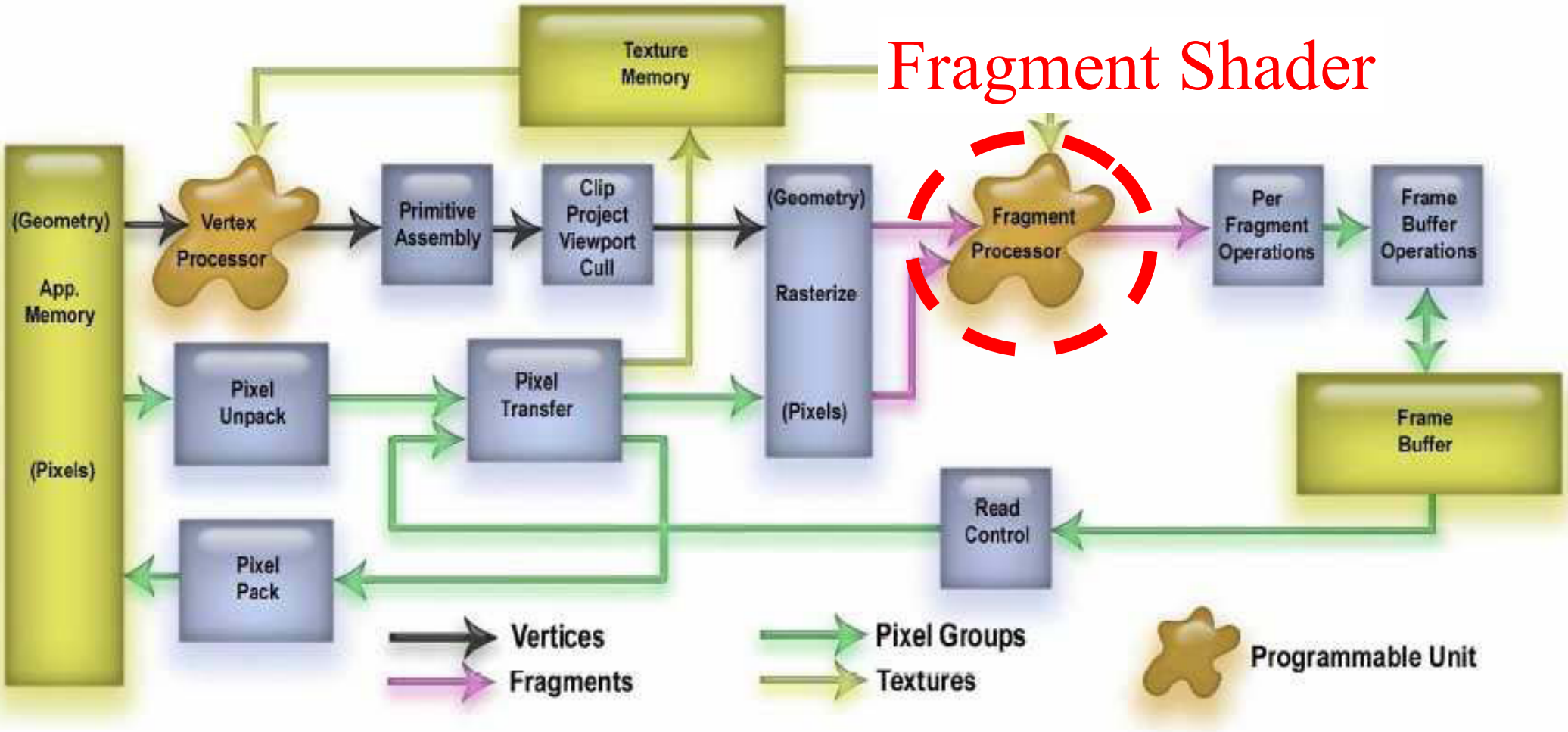

```
uniform float g, m, t;
void main()
{
vec3 object_pos;
object_pos.x = gl_Vertex.x + vel.x*t;
object_pos.y = gl_Vertex.y + vel.y*t
+ g/(2.0*m)*t*t;
object_pos.z = gl_Vertex.z + vel.z*t;

gl_ModelViewProjectionMatrix*
vec4(object_pos,1);
}
```





The University of New Mexico

OpenGL Logical Diagram



Fragment Shaders

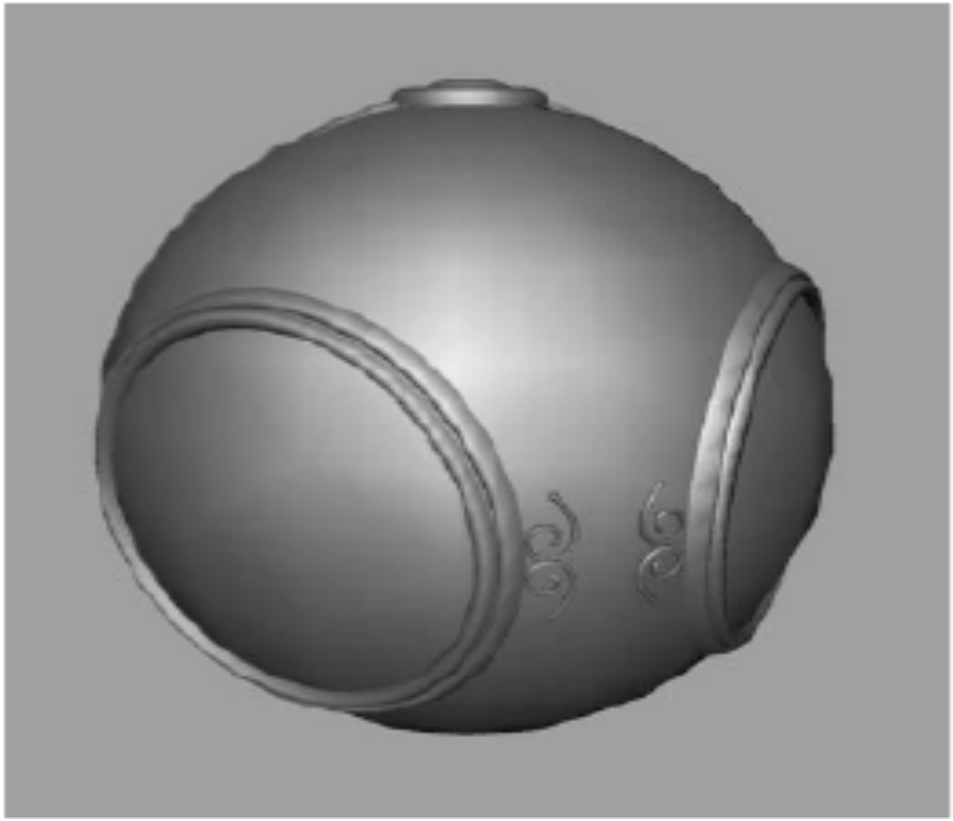
- What is a fragment?
  - Cg Tutorial says: "You can think of a fragment as a 'potential pixel'"
- Perform per-pixel calculations
  - Without knowledge about other fragments (parallelism)
- Your program's responsibilities:
  - Operations on interpolated values
  - Texture access and application
  - Other functions: fog, color lookup, etc.

Fragment Shader Applications

(Per-pixel) Phong shading **Hardware**

Per-vertex lighting

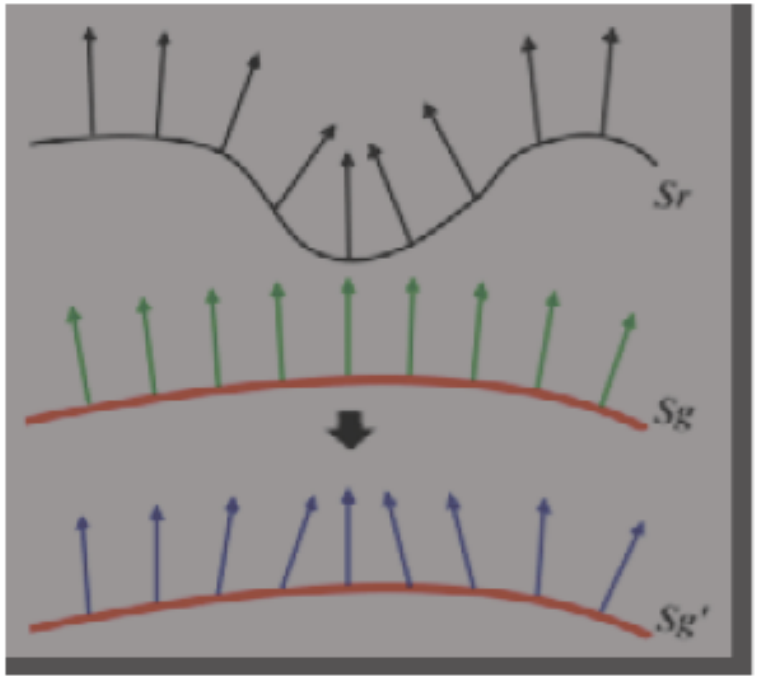
Fragment Shader Applications

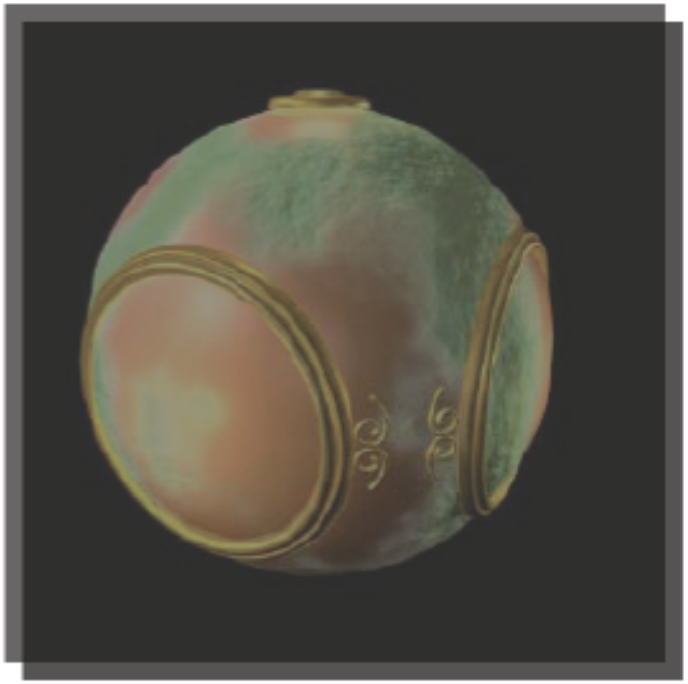


smooth shading

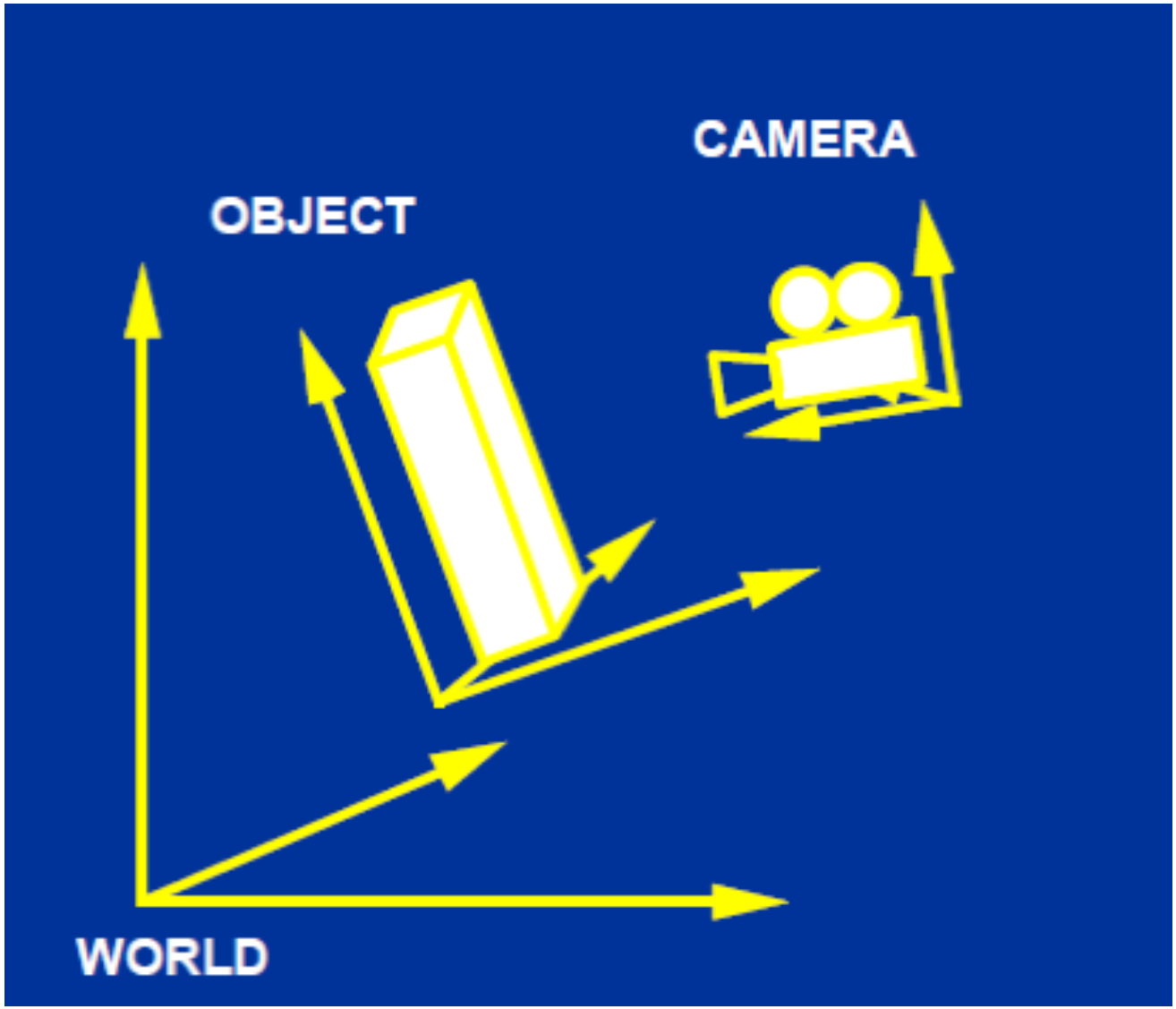
Fragment Shader Applications - Bump Mapping

- Perturb normal for each fragment
- Store perturbation as textures

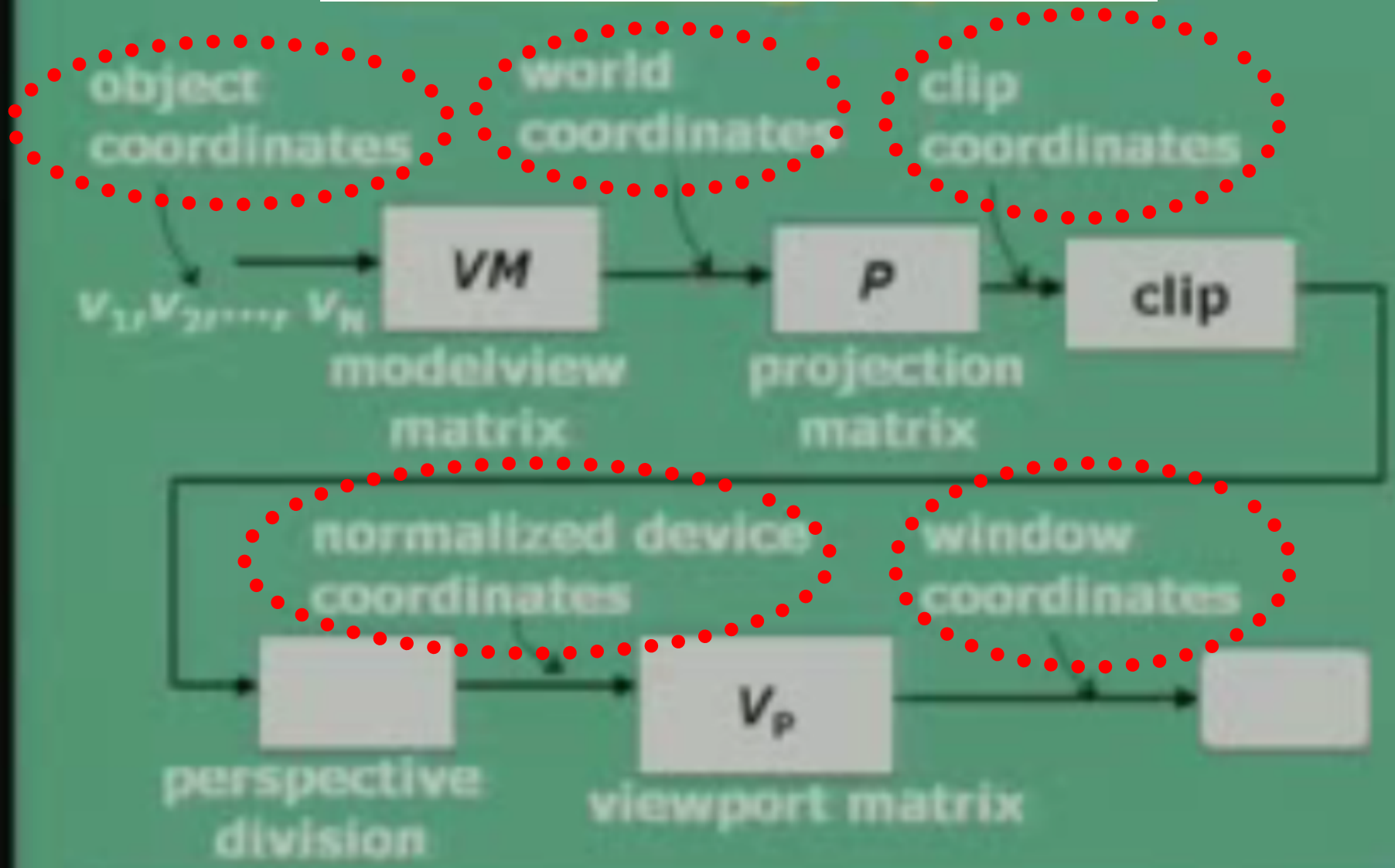




The Object, the World and Camera Frames



3D Viewing Pipeline



From F. S. Hill Jr., Computer Graphics using OpenGL

Viewing Pipeline

MCS - **object** coordinate system
WCS - **world** coordinate system
VCS - **viewing** coordinate system
CCS - **clipping** coordinate system
NDCS - **normalized** device coordinate system
DCS - **device** coordinate system

MC – Model Coordinates

WC – World Coordinates

MC

$(x_o, y_o, z_o, 1)^t$

Modeling Transformation

$M_{model} = T_m R_m S_m \ldots$

WC

$(x_m, y_m, z_m, 1)^t$

Viewing Transformation

$M_{view} = (T_v R_v)^{-1} = R_v^t T_v^{-1}$

VC

$(x_v, y_v, z_v, 1)^t$

DC – Device Coordinates

Projection Transformation

PC

$M_{norm,perspective}$

$M_{norm,oblique}$

$M_{norm,ortho}$

Normalization and clipping

$(x_h, y_h, z_h, h)^t$

Divide h

$(x_p, y_p, z_p, 1)^t$

NC

Viewport Transformation

$M_{vp} = T_{vp} S_{vp} \ldots$

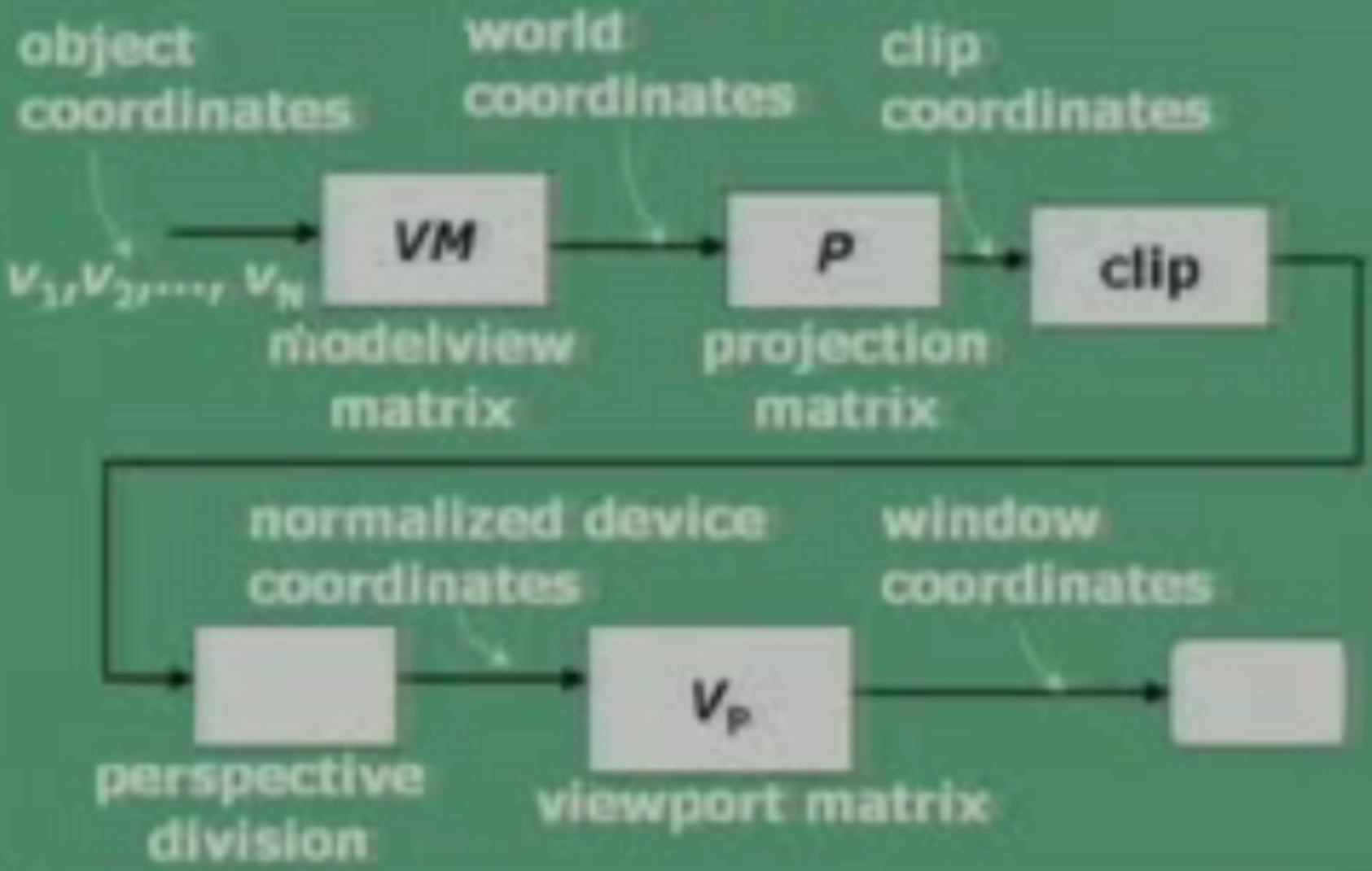
DC

$(x_d, y_d)^t$

OpenGL Primitives

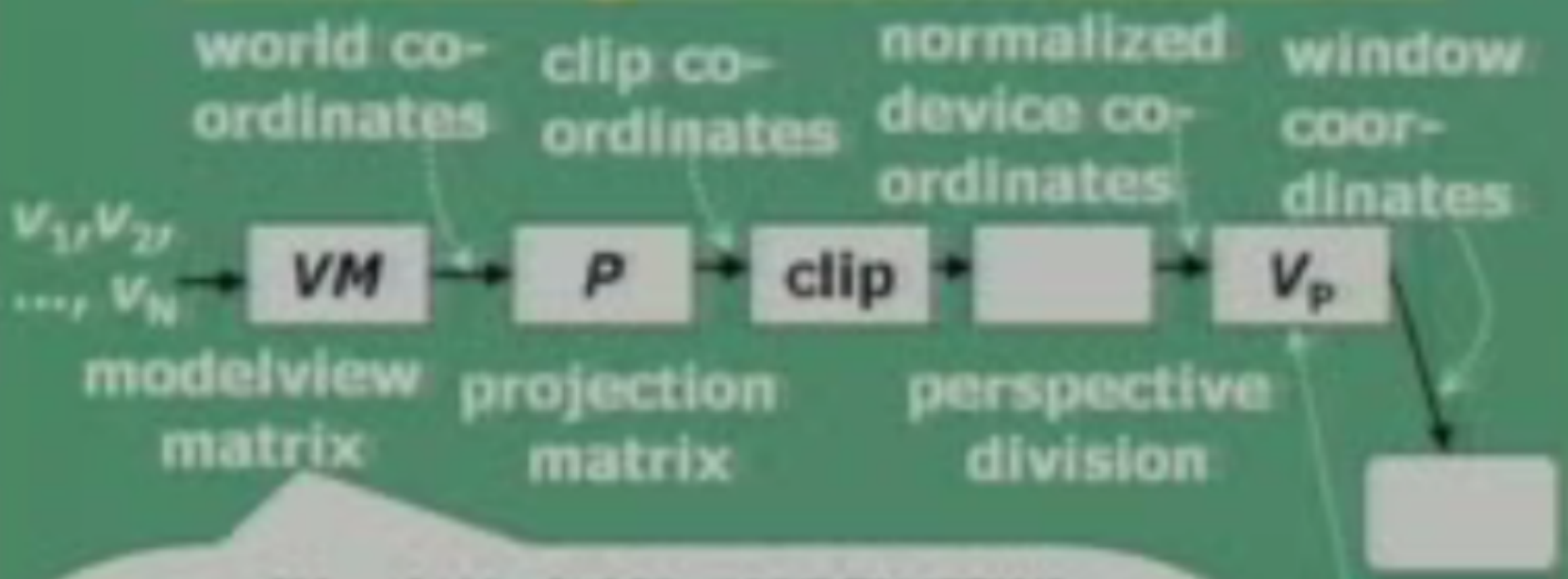
- Points 4D (x, y, z, w) coordinates
- Vectors 4D (x, y, z, w) coordinates
- Matrices 4 X 4 or 16 coordinates

3D Viewing Pipeline



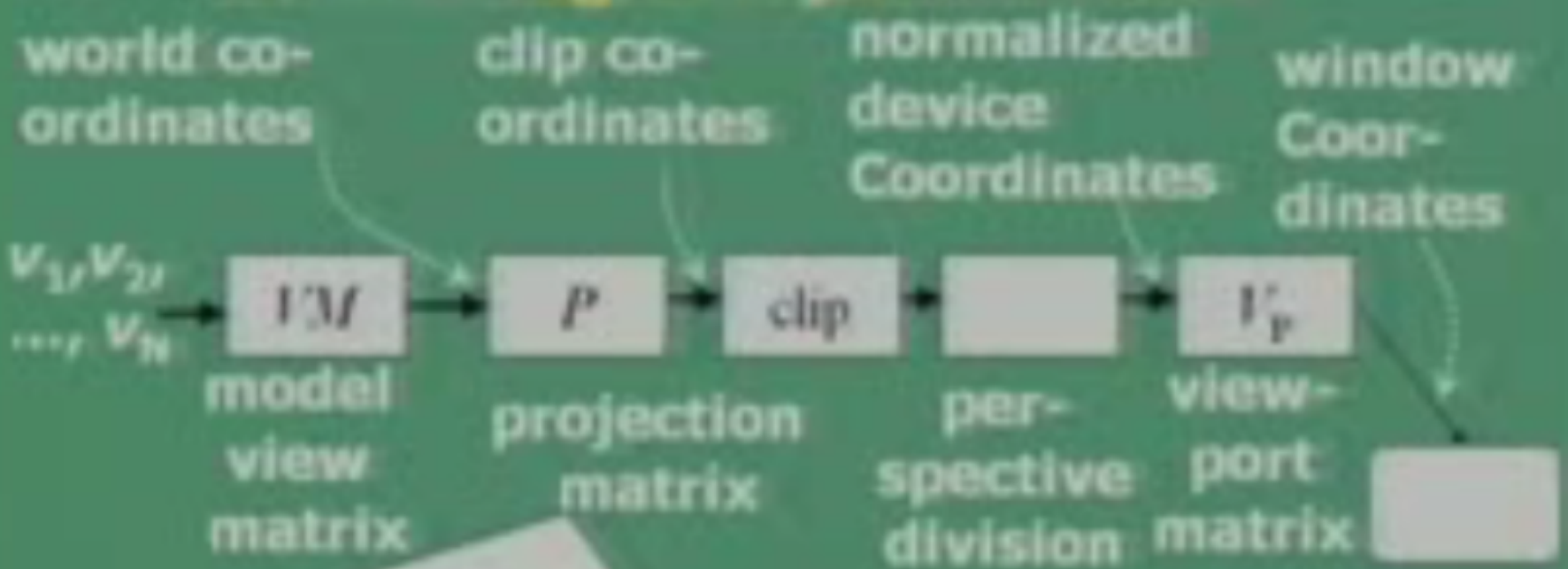
From F. S. Hill Jr., Computer Graphics using OpenGL

3D Viewing – ModelView Matrix




```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// viewing transform
gluLookAt( eyeX, eyeY, eyeZ,
lookAtX, lookAtY, lookAtZ, upX, upY, upZ);
// model transform
glTranslatef(delX, delY, delZ);
glRotatef(angle, i, j, k);
glScalef(multX,multY, multZ);
```

3D Viewing – Projection Matrix




```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
// perspective transform
gluPerspective( viewAngle, aspectRatio,nearZ,farZ );
// other commands for setting projection matrix
glFrustum(left, right, top, bottom);
glOrtho(left, right, top, bottom);
gluOrtho2D(left, right, top, bottom);
```

Objectives

- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a state machine
- Functions
  - Types
  - Formats
- Simple program

Graphics System as a Black Box

1. Primitive functions
2. Attribute functions
3. Viewing functions
4. Transformation functions
5. Input functions
6. Control functions
7. Query functions

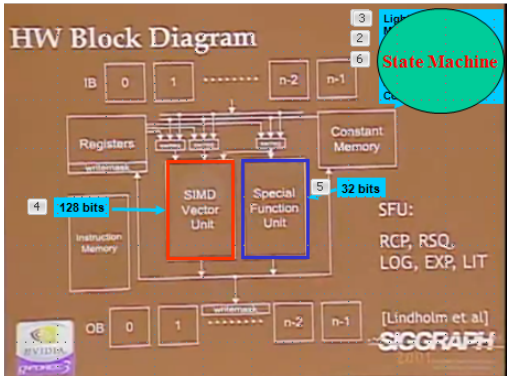
OpenGL Functions

- Primitives (supply their vertices!!!!)
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Modeling
  - Viewing
- Input (**GLUT**)
- Control (**GLUT**)
- Query

1. Primitive functions
2. Attribute functions
3. Viewing functions
4. Transformation functions
5. Input functions
6. Control functions
7. Query functions

```
195     width  = glutGet( GLUT_SCREEN_WIDTH );
196     height = glutGet( GLUT_SCREEN_HEIGHT );
```

OpenGL State (GLOBAL)



- OpenGL is a **State** machine
- OpenGL functions are of two types

Primitive generating

- Can cause output if primitive is visible
- How **vertices** are processed, and appearance of primitive are controlled by the **State**

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

State changing

- Transformation functions
- Attribute functions

```
7           glClear(GL_COLOR_BUFFER_BIT);
```

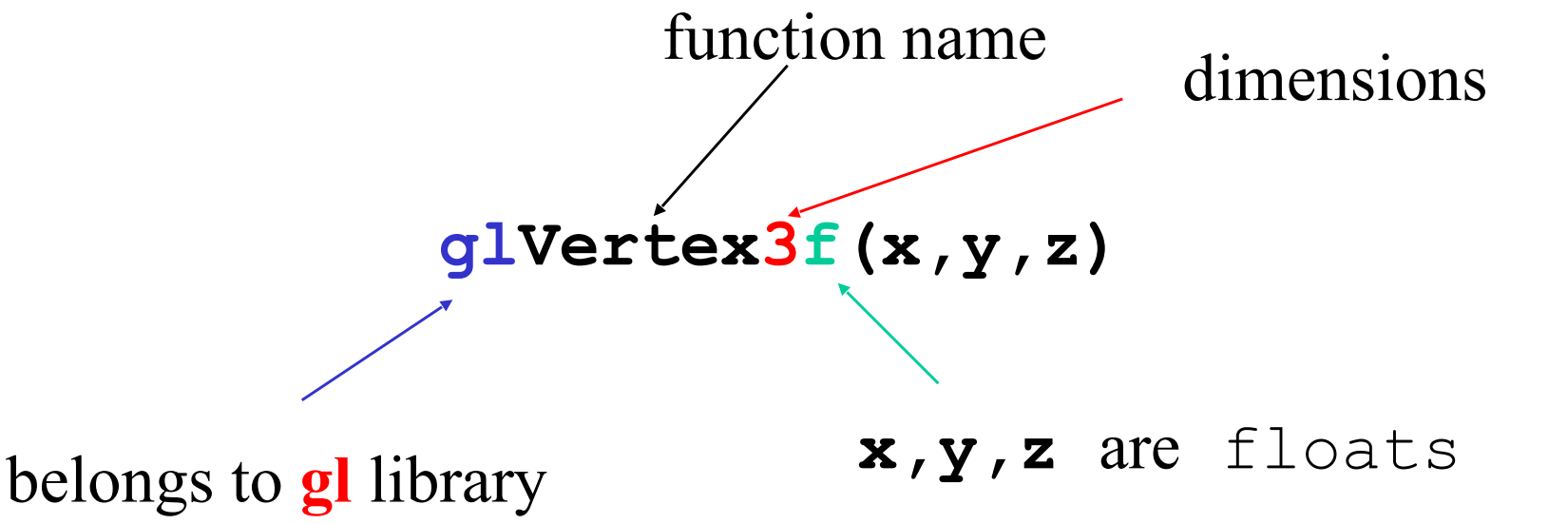
Lack of Object Orientation

- OpenGL is **not object oriented** so that there are multiple functions for a given logical function

```
glVertex3f
glVertex2i
glVertex3dv
```

- **Underlying storage mode is the same**
- Easy to create overloaded functions in **C++** but **issue is efficiency**

OpenGL function format




```
glVertex3fv(p)
```

p is a **pointer to an array**

OpenGL Command Formats

```
glVertex3fv( v )
```

Number of Components

```
2 - (x,y)
3 - (x,y,z)
4 - (x,y,z,w)
```

Data Type

```
b  - byte
ub - unsigned byte
s  - short
us - unsigned short
i  - int
ui - unsigned int
f  - float
d  - double
```

Vector

Omit "v" for Scalar form

```
glVertex2f( x, y )
```

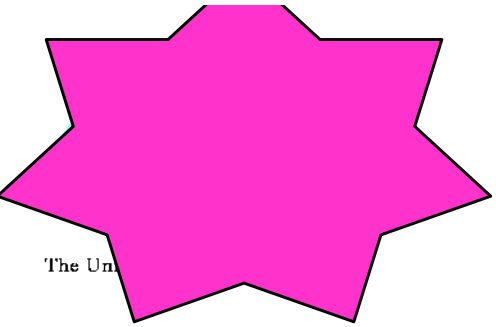
OpenGL #defines

- Most constants are defined in the include files **gl.h**, **glu.h** and **glut.h**

Note #include <GL/glut.h> should automatically include the others

```
3 #include <GL/glut.h>          /* glut.h includes gl.h and glu.h*/
```

- include files also define OpenGL data types: **GLfloat**, **GLdouble**,….



LECTURE 2 CLASS PARTICIPATION

VH, publish LECTURE 2 CLASS PARTICIPATION

Class Participation on Lecture 2

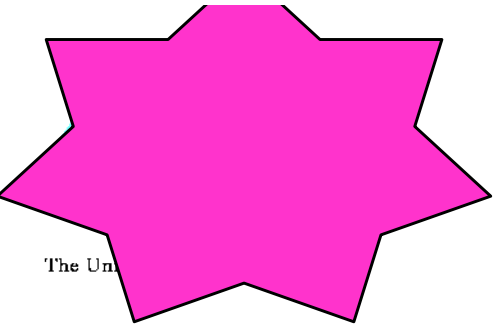
VH, publish Class Participation on Lecture 2

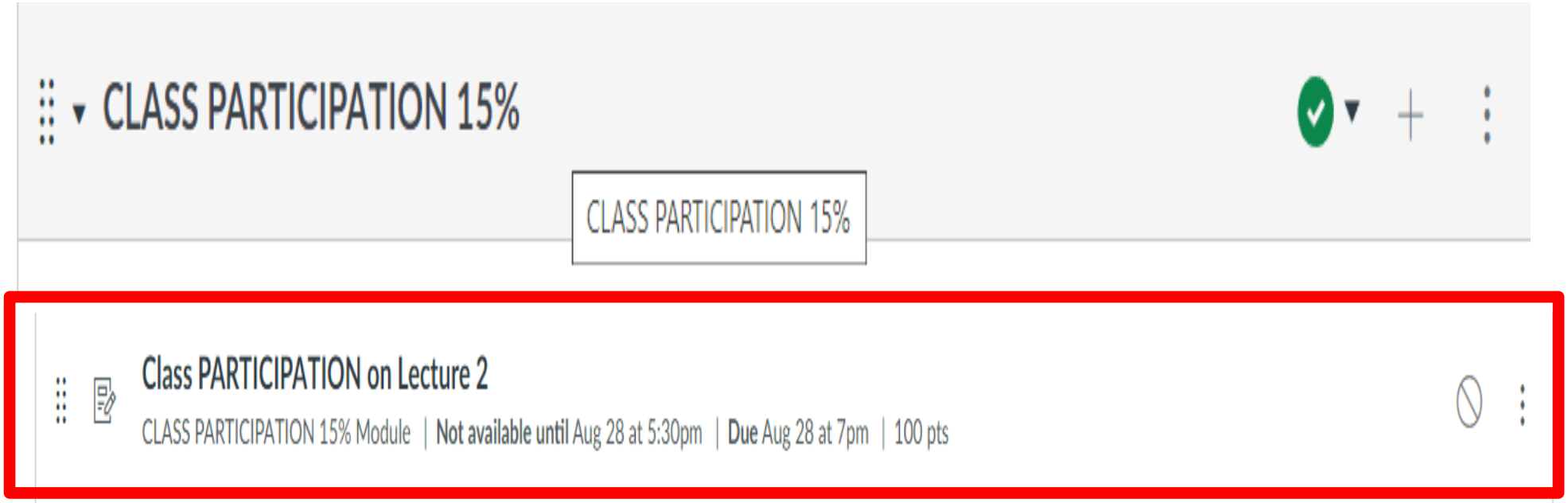
simplePolygon.c

VH, publish simplePolygon.c

house.c

VH, publish house.c





VH, publish Assignment Class PARTICIPATION on Lecture 2

Download Class PARTICIPATION on Lecture 2.doc

Class PARTICIPATION on Lecture 2

Publish | Edit

Download and complete this word document.

Class Participation on Lecture 2.doc

simplePolygon.c

house.c

You will be prompted when to Upload completed document to CANVAS as **score.doc** (example 100.doc).

Warning:

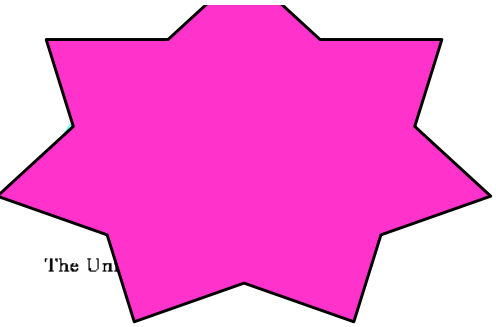
TA, at random, will inspect the Uploaded document.

If you score is not honestly entered you will get a zero.

Points 100

Submitting a file upload

Due	For	Available from	Until
Aug 28 at 7pm	Everyone	Aug 28 at 5:30pm	Aug 28 at 7pm



Name: _____

Total score: [ ]

Class PARTICIPATION on Lecture 2.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

I. Download the simplePolygon.c program from CANVAS, build a Lecture2 C++ Empty Project and add it.

a. **Build and run the project.**
(Take print screen under 1. a) (25 points)

Class Participation I. a.!

A Simple Program
simplePolygon.c

Th

Generate a square on a solid background

Lecture2 - Microsoft Visual Studio

File Edit View Project Build Debug Tools Visual Assert Test Window Help

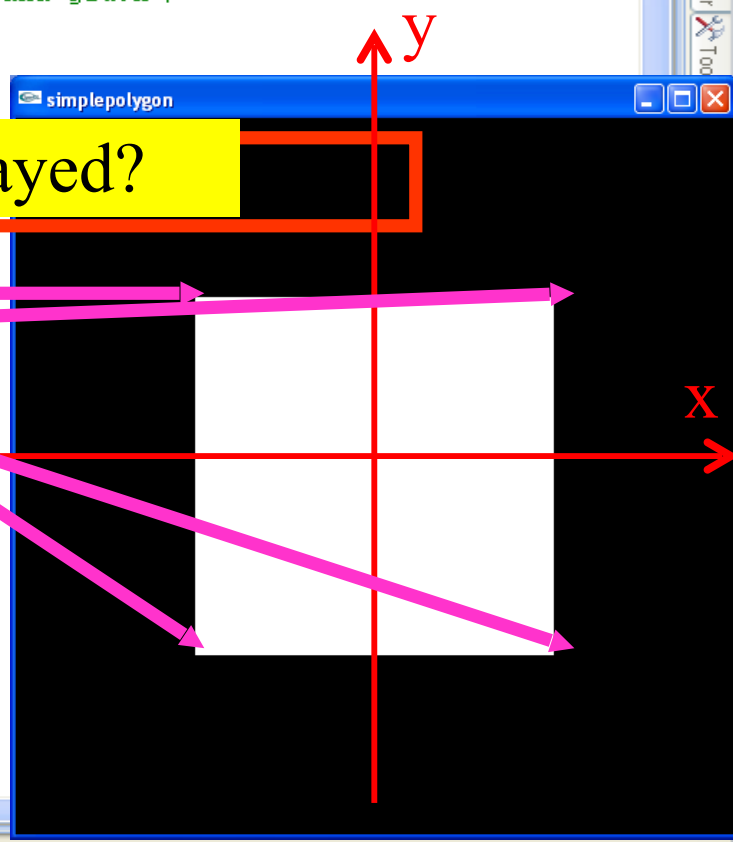
Debug Win32

Solution Explorer - Solution '...

Solution 'Lecture2' (1 project)
Lecture2
Header Files
Resource Files
Source Files
simplepolygon.c

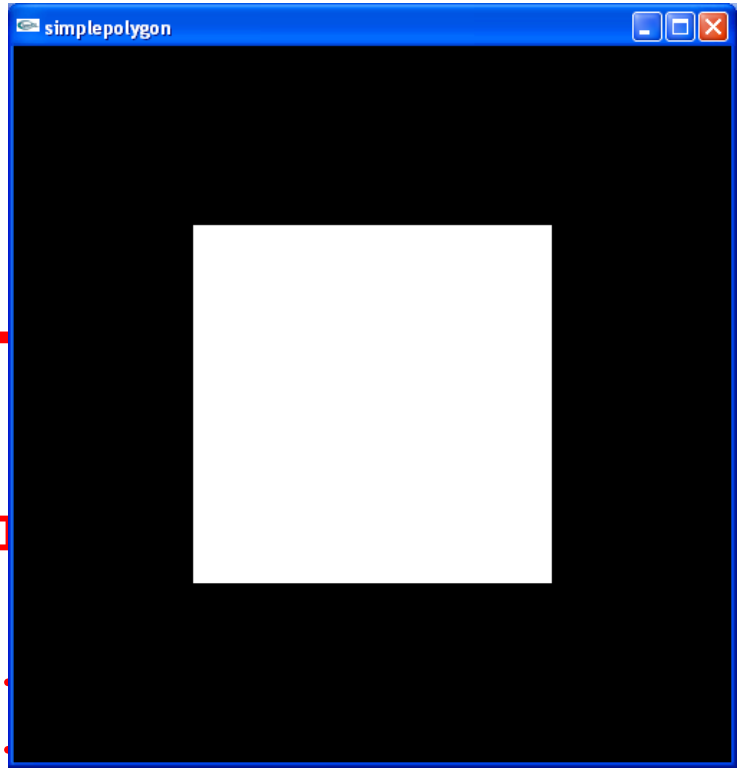
```
// Your Last, First Name
// Lecture 2 Fall 2023
/* simplepolygon.c  This program draws a white rectangle on a black background. */

#include <GL/glut.h>          /* glut.h includes gl.h and glu.h*/

void display(void)
{
      glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_POLYGON);
        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_RGB);
     glutInitWindowSize(500,500);
     glutInitWindowPosition(0,0);
     glutCreateWindow("simplepolygon");
     glutDisplayFunc(display);
     glutMainLoop();
}
```

Where is this polygon displayed?



Ready Ln 17 Col 14 Ch 11 INS

simplePolygon.c




```
#include <GL/glut.h>
  void mydisplay(){
glClear(GL_COLOR_BUFFER_BI
    glBegin(GL_POLYGON);
      glVertex2f(-0.5, -0.
      glVertex2f(-0.5,  0.
      glVertex2f( 0.5,  0.5);
      glVertex2f( 0.5, -0.5);
        glEnd();
      glFlush();
       }
int main(int argc, char** argv){
  glutCreateWindow("simplepolygon");
    glutDisplayFunc(mydisplay);
      glutMainLoop();
       }
```


```
glutDisplayFunc(mydisplay);
```

Event Loop

- Note that the program defines a ***display callback*** function named `mydisplay`

  Every **glut** program must have a **display callback**

  The **display callback** is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened

  The `main` function ends with the program entering an event loop

```
glutMainLoop();
```

Defaults

Light parameters
Material Properties
Texture
Normals
ModelView Matrix
Projection Matrix
Color

`simplepolygon.c` is too simple

Makes heavy use of **State** variable default values for

- **Viewing**
- **Colors**
- **Window parameters**

Next version will make the **defaults more explicit**

Programming with OpenGL
Part 2: Complete Programs
Chapter 2.5

Objectives

Light parameters
Material Properties
Texture
Normals
ModelView Matrix
Projection Matrix
Color

- Refine the first program
  - Alter the **default values**
  - Introduce a standard program structure
- Simple viewing
  - Two-dimensional viewing as a special case of three-dimensional viewing
- Fundamental OpenGL primitives
- Attributes

Program Structure

Light parameters
Material Properties
Texture
Normals
ModelView Matrix
Projection Matrix
Color

- **GLUT** OpenGL programs have a structure that consists of the following functions

`main()`:

- defines the **callback functions**
- opens one or more windows with the required properties
- enters event loop (last executable statement)

`init()`: sets the **State** variables

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

- Viewing
- Attributes

callbacks

- Display function

```
29      glutDisplayFunc(display);
```

- Input and window functions

simplePolygon.c revisited

In this version, we shall see the same output but we have defined all the relevant **State** values through function calls using the **default values**

In particular, we set

- **Colors**
- **Viewing conditions**
- **Window properties**

```
void init()
{
    glClearColor (0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

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

Lecture2 - Microsoft Visual Studio

File Edit View Project Build Debug Tools Visual Assert Test Window Help

Debug Win32

Solution Explorer - Solution '...

Solution 'Lecture2' (1 project)
- Lecture2
  - Header Files
  - Resource Files
  - Source Files
    - LessSimple.c

LessSimple.c*

(Global Scope) init()

```
void mydisplay(void)
{
      glClear(GL_COLOR_BUFFER_BIT);

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

      glFlush();
}

void init()
{
    glClearColor (0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("LessSimple");
    glutDisplayFunc(mydisplay);
```

Server Explorer Toolbox

Soluti... Class ... Prope...

Ready Ln 26 Col 2 Ch 2 INS

simplePolygon.c

The following code fragment demonstrates a very simple OpenGL program which opens a graphics window and draws a square. It also prints 'helllo world' in the console window. The code is illustrative of the use of the glut library in opening the graphics window and managing the display loop.

glutInit()

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

Following the initial print statement, the glutInit() call initializes the GLUT library and also processes any command line options related to glut. These command line options are window-system dependent.

```
void mydisplay(void)
{
        glClear(GL_COLOR_BUFFER_BIT);

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

        glFlush();
}
```

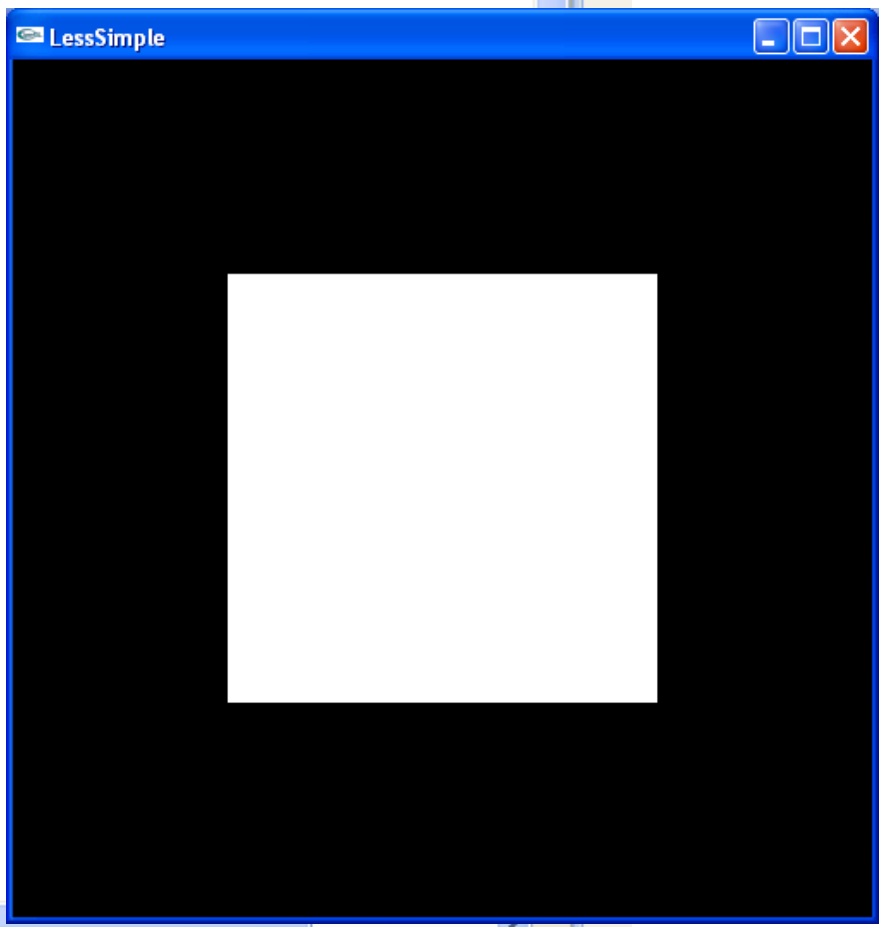
display()

The display() call-back function clears the screen, sets the current colour to red and draws a square polygon. The last call, glFlush(), forces previously issued OpenGL commands to begin execution.

glut Program Structure

LessSimple.c

```
5  void mydisplay(void)
6  {
7       glClear(GL_COLOR_BUFFER_BIT);
8      glBegin(GL_POLYGON);
9          glVertex2f(-0.5,  -0.5);
10         glVertex2f(-0.5,   0.5);
11         glVertex2f( 0.5,   0.5);
12         glVertex2f( 0.5,  -0.5);
13      glEnd();
14      glFlush();
15 }
16
17 void init()
18 {
19     glClearColor (0.0, 0.0, 0.0, 1.0);
20     glColor3f(1.0, 1.0, 1.0);
21     glMatrixMode (GL_PROJECTION);
22     glLoadIdentity ();
23     glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
24 }
25
26 int main(int argc, char** argv)
27 {
28     glutInit(&argc,argv);
29     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
30     glutInitWindowSize(500,500);
31     glutInitWindowPosition(0,0);
32     glutCreateWindow("LessSimple");
33     glutDisplayFunc(mydisplay);
34     init();
35     glutMainLoop();
36 }
```



Main

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

int main(int argc, char** argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("LessSimple");
  glutDisplayFunc(mydisplay);

  init();

  glutMainLoop();
}
```

includes `gl.h`

define window properties

display callback

set OpenGL state

enter event loop

glutInitDisplaymode()


```
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
```

```
29    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
```

glutInitDisplaymode()

Before opening a graphics window, we need to decide on the `depth' of the buffers associated with the window. The following table shows the types of parameters that can be stored on a per-pixel basis:

The various GLUT_* options are invoked together by ORing them together, as illustrated in the example code, which creates a graphics window which has only a single copy of all buffers (GLUT_SINGLE), does not have an alpha buffer (GLUT_RGB), and has a depth buffer (GLUT_DEPTH).

RGB	Red, green and blue, Typically 8 bits per pixel	GLUT_RGB
A	Alpha or accumulation buffer, Used for composting images	GLUT_RGBA
Z	Depth value, used for Z-buffer visibility tests	GLUT_DEPTH
Double buffer	Extra copy of all buffers, Used for smoothing animation	GLUT_DOUBLE
Stencil buffer	Several extra bits, Useful in composting images	GLUT_STENCIL

glutInitWindowPosition(), glutInitWindowSize(), glutCreateWindow()

glutInitWindowPosition(), glutInitWindowSize(), glutCreateWindow()

These calls assign an initial position, size, and name to the window and create the window itself.

```
30      glutInitWindowSize(500,500);
31      glutInitWindowPosition(0,0);
32      glutCreateWindow("LessSimple");
```

glClearColor(), glMatrixMode(), glLoadIdentity(), glOrtho()

glClearColor() sets the colour to be used when clearing the window. The remaining calls are used to define the type of camera projection. In this case, an orthographic projection is specified using a call to glOrtho(x1,x2,y1,y2,z1,z2). This defines the field of view of the camera, in

```
19      glClearColor (0.0, 0.0, 0.0, 1.0);
20      glColor3f(1.0, 1.0, 1.0);
21      glMatrixMode (GL_PROJECTION);
22      glLoadIdentity ();
23      glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

glutDisplayFunc(display), glutMainLoop()

This provides the name of the function you would like to have called whenever glut thinks the window needs to be redrawn. Thus, when the window is first created and whenever the window is uncovered or moved, the user-defined display() function will be called.

glutDisplayFunc() registers the call-back function, while glutMainLoop() hands execution control over to the glut library.

```
33      glutDisplayFunc(mydisplay);
34      init();
35      glutMainLoop();
```

GLUT functions

- **glutInit** allows application to get command line arguments and initializes system
- **glutInitDisplayMode** requests properties for the window (the *rendering context*)
  - RGB color
  - Single buffering
  - Properties logically ORed together
- **glutWindowSize** in pixels
- **glutWindowPosition** from top-left corner of display
- **glutCreateWindow** create window with title "LessSimple"
- **glutDisplayFunc** display callback
- **glutMainLoop** enter infinite event loop

```
26  int main(int argc, char** argv)
27  {
28      glutInit(&argc,argv);
29      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
30      glutInitWindowSize(500,500);
31      glutInitWindowPosition(0,0);
32      glutCreateWindow("LessSimple");
33      glutDisplayFunc(mydisplay);
34      init();
35      glutMainLoop();
36  }
```

init

```
void init()
{
 glClearColor (0.0, 0.0, 0.0, 1.0);

 glColor3f(1.0, 1.0, 1.0);

 glMatrixMode (GL_PROJECTION);
 glLoadIdentity ();
 glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

black clear color

opaque window

fill/draw with white

viewing volume

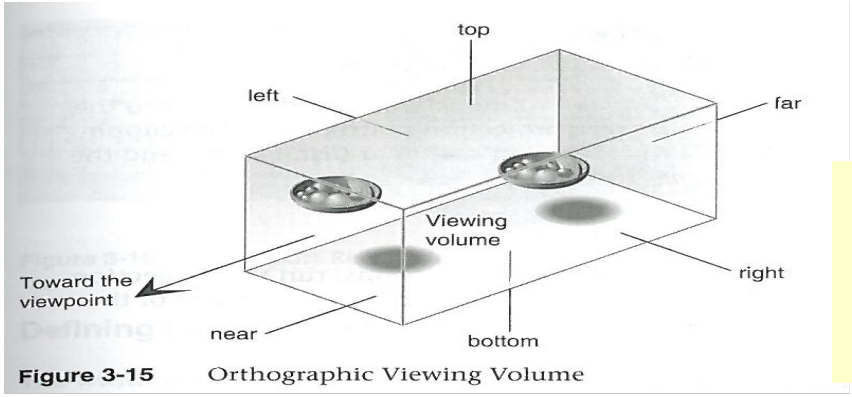


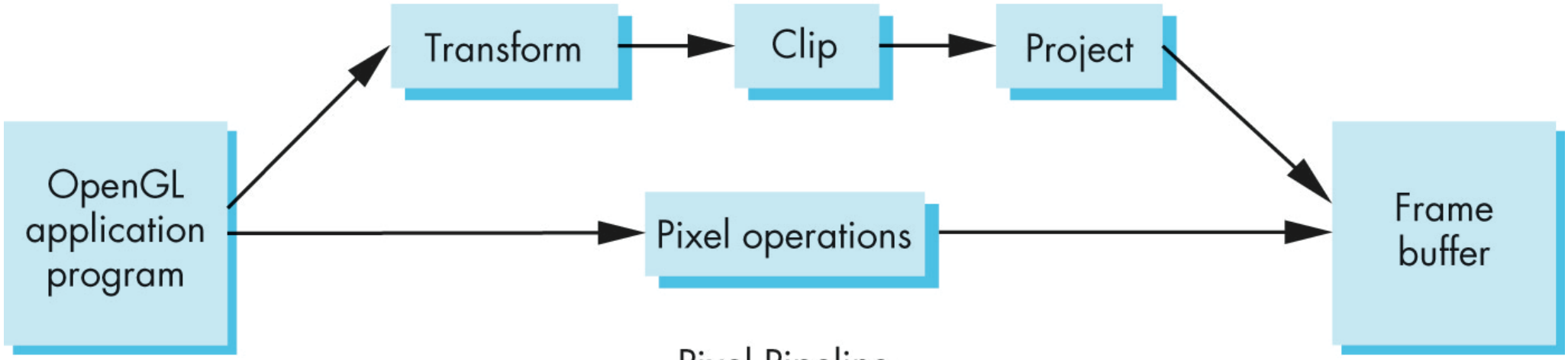
Figure 3-15 Orthographic Viewing Volume

void ***glOrtho****(GLdouble* ***left****, GLdouble* ***right****, GLdouble* ***bottom****, GLdouble* ***top****, GLdouble* ***near****, GLdouble* ***far****);*

Simplified OpenGL pipeline

Modeling Pipeline

Geometric Pipeline



Pixel Pipeline

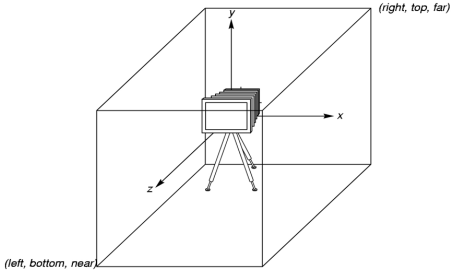
Rendering Pipeline

Coordinate Systems

The units in **`glVertex`** are determined by the application and are called *object* or *problem coordinates*

The **viewing** specifications are also in object coordinates and it is the **size of the viewing volume** that determines what will appear in the image

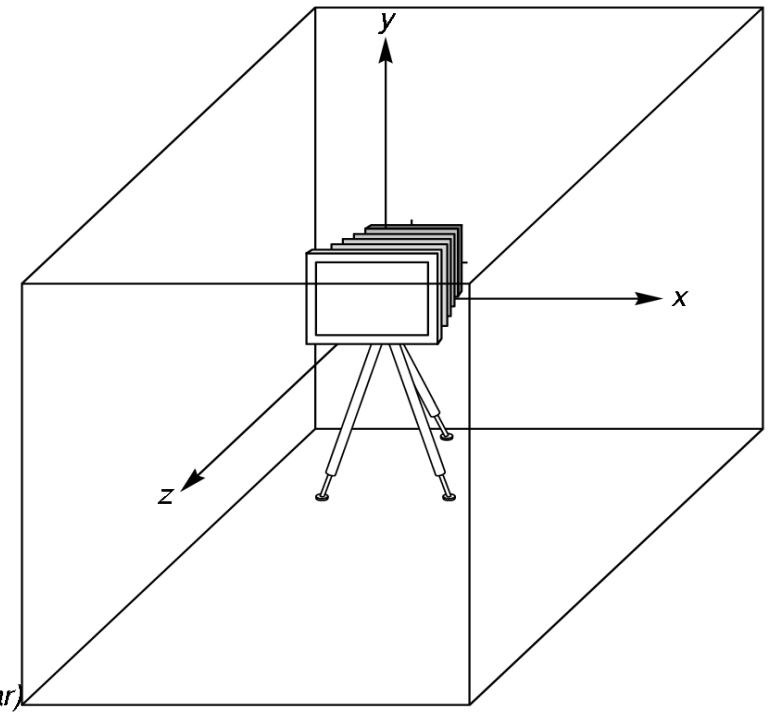
Internally, OpenGL will convert to ***camera (eye)*** *coordinates* and later to *screen coordinates*



OpenGL Camera

- OpenGL places a camera at the origin in object space pointing in the negative z direction

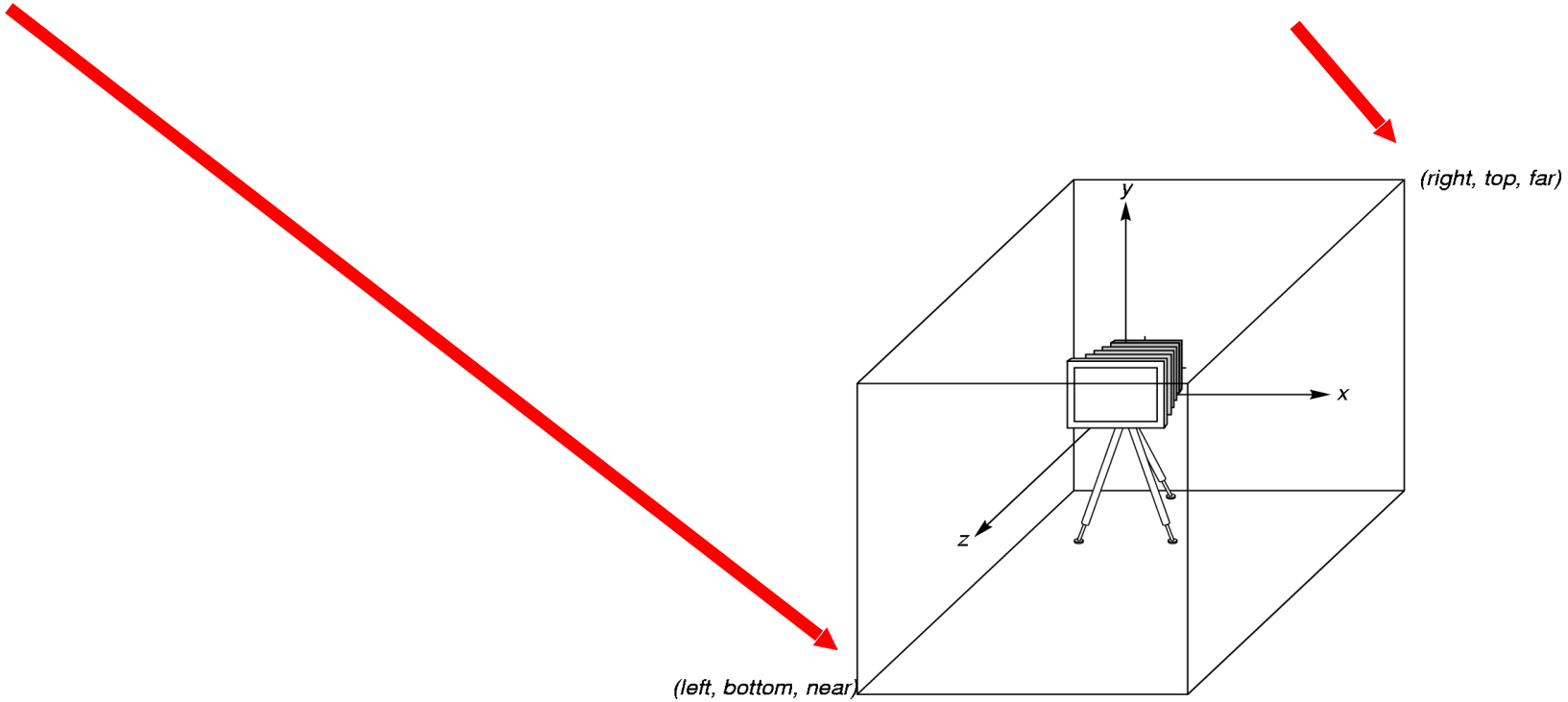
- **The default viewing volume is a box centered at the origin with a side of length 2**



Default OpenGL Camera

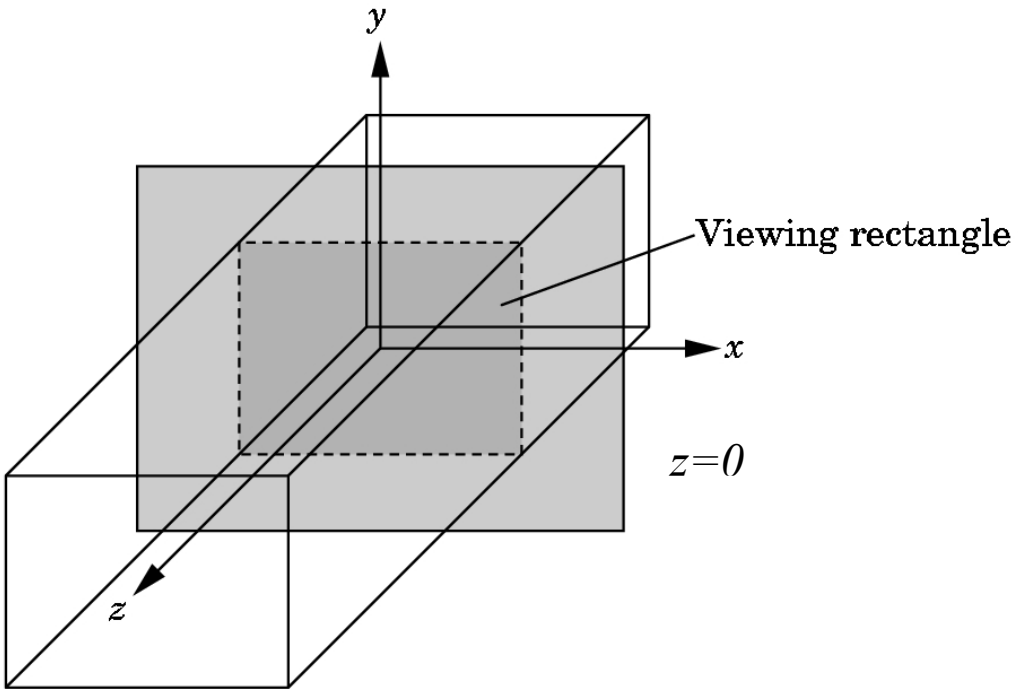
If we do not specify a viewing volume, OpenGL uses its default, a 2 X 2 X 2 cube, with the origin in the center.

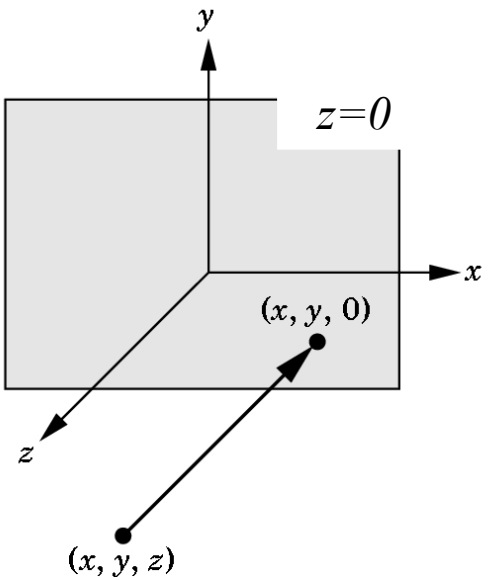
In terms of our two-dimensional plane, the bottom-left corner is at (- 1.0, - 1.0), and the upper-right corner is at (1.0, 1.0).



Orthographic Viewing

In the default orthographic view, Points are **projected forward along the z axis onto the plane $z=0$**





DOP – Direction Of Projection

Transformations and Viewing

- In OpenGL, projection is carried out by a projection matrix (transformation)
- There is only one set of transformation functions so we must set the matrix mode first

```
glMatrixMode (GL_PROJECTION)
```

- Transformation functions are incremental so we start with an identity matrix and alter it with a projection matrix that gives the **view volume**

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

Two- and three-dimensional viewing

In `glOrtho(left, right, bottom, top, near, far)` the **near** and **far distances are measured from the camera**

Two-dimensional **vertex** commands place all **vertices** in the plane z=0

If the application is in two dimensions, we can use the function

`gluOrtho2D(left, right, bottom, top)`

In two dimensions, the view or clipping volume becomes a ***clipping window***

mydisplay

```
void mydisplay()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_POLYGON);
      glVertex2f(-0.5, -0.5);
      glVertex2f(-0.5, 0.5);
      glVertex2f(0.5, 0.5);
      glVertex2f(0.5, -0.5);
 glEnd();
 glFlush();
}
```

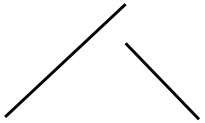
OpenGL Data Types

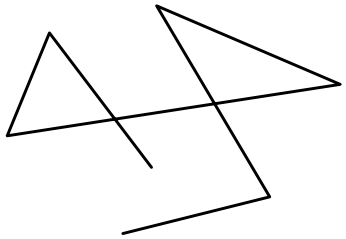
void glVertex2{sifd}{v} (TYPE *x*, TYPE *y*

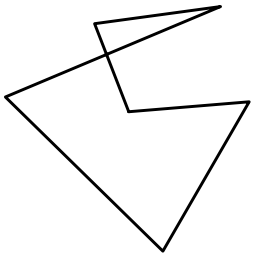
character	data type	C-language type	OpenGL type definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int	GLuint, GLenum, GLbitfield
		void	GLvoid

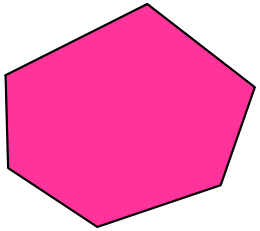
OpenGL Primitives

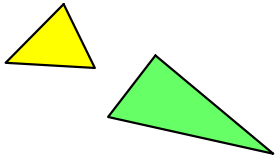

GL_POINTS

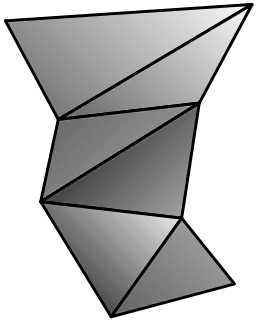

GL_LINES

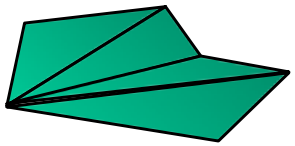

GL_LINE_STRIP

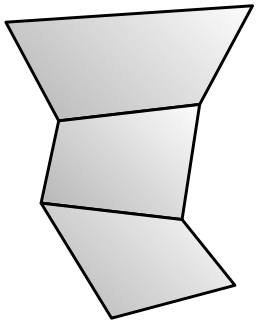

GL_LINE_LOOP


GL_POLYGON

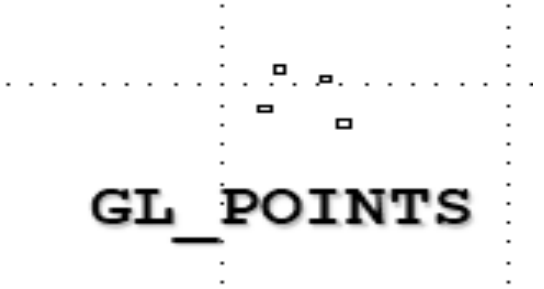

GL_TRIANGLES


GL_TRIANGLE_STRIP


GL_TRIANGLE_FAN


GL_QUAD_STRIP

Drawing



- Points

```
glBegin (GL_POINTS);
   glVertex3f (x1, y1, z1);
   glVertex3f (x2, y2, z2);
   ......
glEnd ();
```

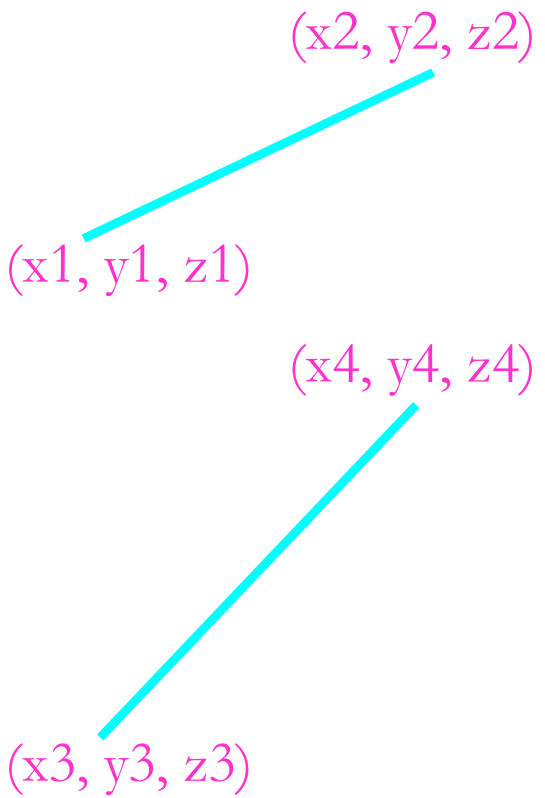
```
glPointSize (float size);
```

Drawing

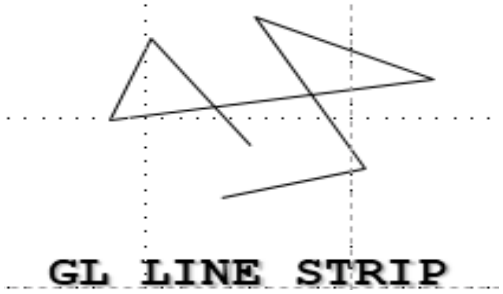
- Lines

```
glBegin (GL_LINES);
   glVertex3f (x1, y1, z1);
   glVertex3f (x2, y2, z2);

   glVertex3f (x3, y3, z3);
   glVertex3f (x4, y4, z4);
glEnd ();
```

```
glLineWidth (float width);
```

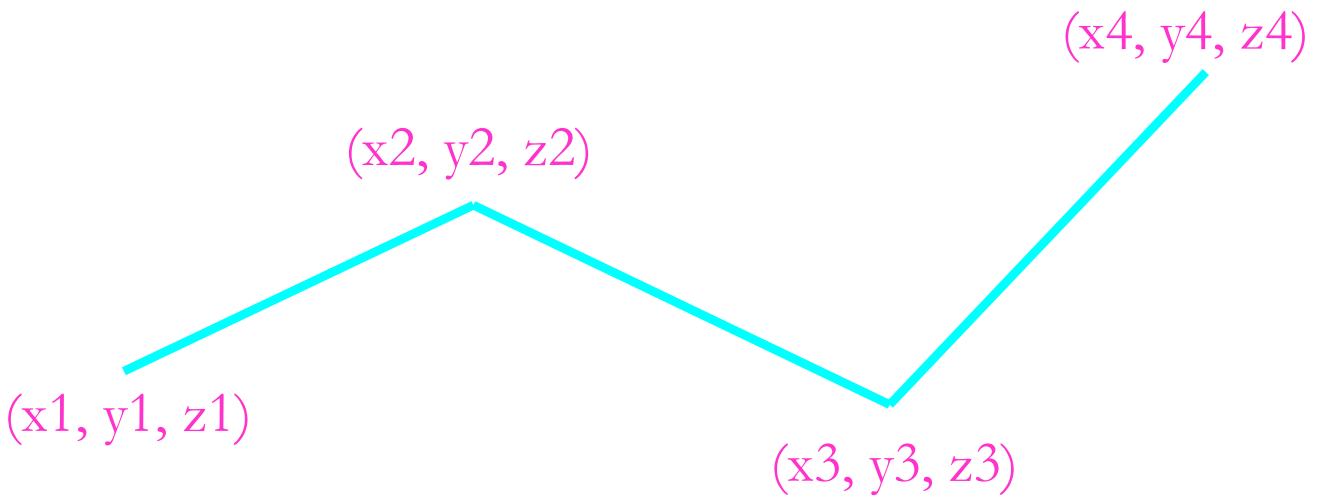


Drawing

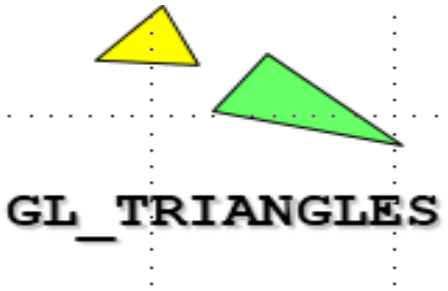


- Line Strips

```
glBegin (GL_LINE_STRIP);
   glVertex3f (x1, y1, z1);
   glVertex3f (x2, y2, z2);
   glVertex3f (x3, y3, z3);
   glVertex3f (x4, y4, z4);
glEnd ();
```

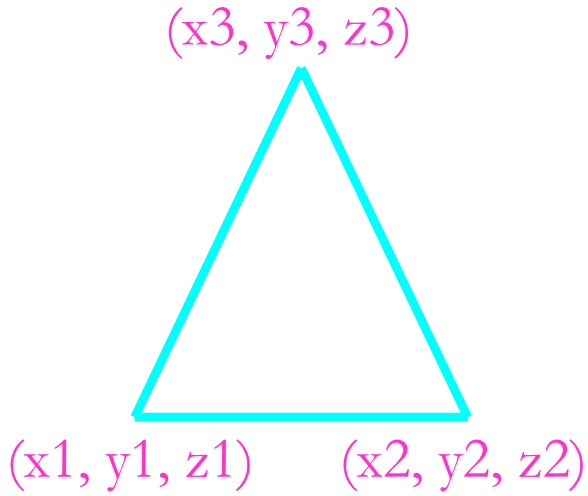


Drawing

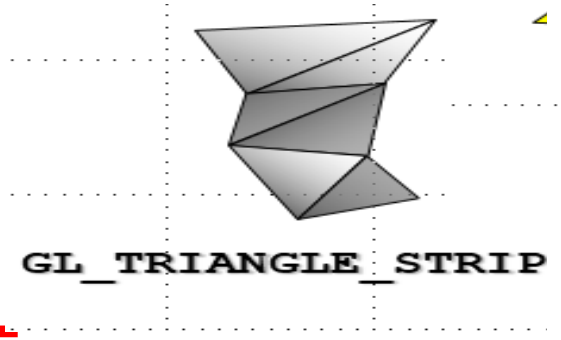


- Triangles

```
glBegin (GL_TRIANGLES);
   glVertex3f (x1, y1, z1);
   glVertex3f (x2, y2, z2);
   glVertex3f (x3, y3, z3);
glEnd ();
```

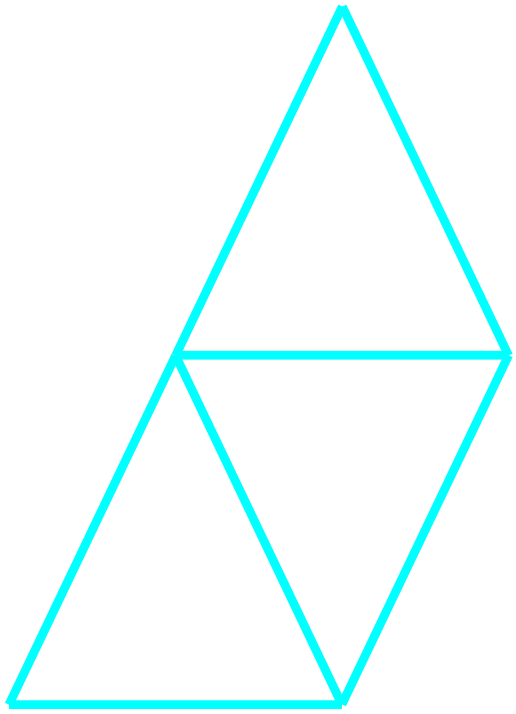


Drawing



- Triangle Strips

```
glBegin (GL_TRIANGLE_STRIP);
   glVertex3f (x1, y1, z1);
   glVertex3f (x2, y2, z2);
   glVertex3f (x3, y3, z3);

   glVertex3f (x4, y4, z4);

   glVertex3f (x5, y5, z5);
glEnd ();
```



The University of New Mexico

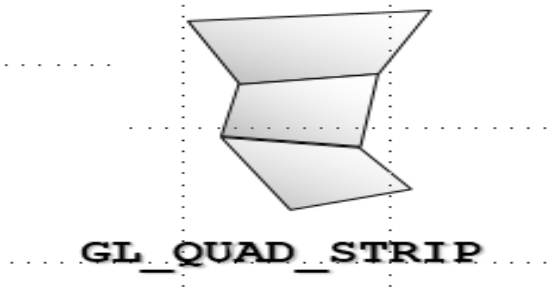
Drawing

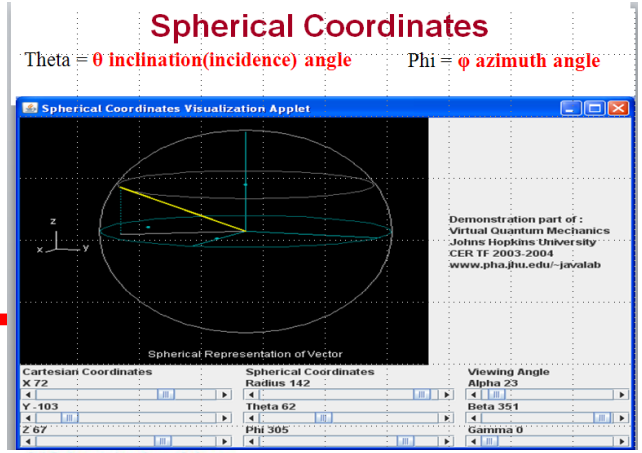
- Quads

```
glBegin (GL_QUADS);
        ......
glEnd ();
```

- Quad Strips

```
glBegin (GL_QUAD_STRIP);
        ......
glEnd ();
```





2.4.3 Approximating a Sphere

Fans and strips allow us to approximate many curved surfaces simply. For example, one way to construct an approximation to a sphere is to use a set of polygons defined by lines of longitude and latitude as shown in Figure 2.15. We can do so very efficiently using either quad strips or triangle strips. Consider a unit sphere. We can describe it by the following three equations:

$$x(\theta, \phi) = \sin\theta \cos\phi,$$
$$y(\theta, \phi) = \cos\theta \cos\phi,$$
$$z(\theta, \phi) = \sin\phi.$$

If we fix θ and draw curves as we change ϕ, we get circles of constant longitude. Likewise, if we fix ϕ and vary θ, we obtain circles of constant latitude. By generating points at fixed increments of θ and ϕ, we can define quadrilaterals as shown in Figure 2.15. Remembering that we must convert degrees to radians for the standard trigonometric functions, the code for the quadrilaterals corresponding to increments of 20 degrees in θ and to 20 degrees in ϕ is

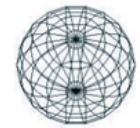


FIGURE 2.15 Sphere approximation with quadrilaterals.

GL_QUAD_STRIP

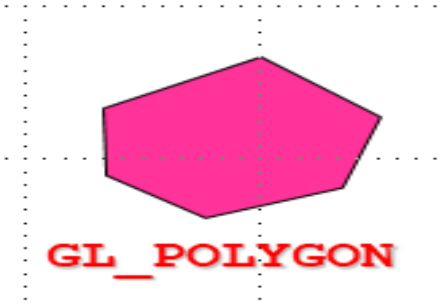
```
for(phi=-80.0; phi<=80.0; phi+=20.0)
{
    phir=c*phi;
    phir20=c*(phi+20);
    glBegin(GL_QUAD_STRIP);
    for(theta=-180.0; theta<=180.0;theta+=20.0)
    {
        thetar=c*theta;
        x=sin(thetar)*cos(phir);
        y=cos(thetar)*cos(phir);
        z=sin(phir);
        glVertex3d(x,y,z);
        x=sin(thetar)*cos(phir20);
        y=cos(thetar)*cos(phir20);
        z=sin(phir20);
        glVertex3d(x,y,z);
    }
    glEnd();
}
```

However, we have a problem at the poles, where we can no longer use strips because all lines of longitude converge there. We can, however, use two triangle fans, one at each pole as follows:

GL_TRIANGLE_FAN

```
glBegin(GL_TRIANGLE_FAN);
   glVertex3d(0.0, 0.0 , 1.0);
   c=M_PI/180.0;
   c80=c*80.0;
   z=sin(c80);
   for(thet=-180.0; theta<=180.0;theta+=20.0)
   {
       thetar=c*theta;
       x=sin(thetar)*cos(c80);
       y=cos(thetar)*cos(c80);
       glVertex3d(x,y,z);
   }
glEnd();

glBegin(GL_TRIANGLE_FAN);
   glVertex3d(0.0, 0.0, -1.0);
   z=-sin(c80);
   for(theta=-180.0;theta<=180.0;theta+=20.0)
   {
       thetar=c*theta;
       x=sin(thetar)*cos(c80);
       y=cos(thetar)*cos(c80);
       glVertex3d(x,y,z);
   }
glEnd();
```

Drawing



Polygon Modes

```
glPolygonMode (GL_BACK_AND_FRONT, GL_LINE);
glPolygonMode (GL_BACK_AND_FRONT, GL_FILL);
```

Hidden Surface Removal

```
glEnable  (GL_DEPTH_TEST);
glDisable (GL_DEPTH_TEST);
```

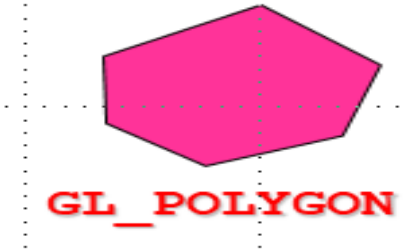
`GL_LINE`

Boundary edges of the polygon are drawn as line segments. Line attributes such as `GL_LINE_WIDTH` and `GL_LINE_SMOOTH` control the rasterization of the lines. Polygon rasterization attributes other than `GL_POLYGON_MODE` have no effect.

`GL_FILL`

The interior of the polygon is filled. Polygon attributes such as `GL_POLYGON_SMOOTH` control the rasterization of the polygon.

Polygon Issues



- OpenGL **will only display polygons correctly** that are

  Simple: edges cannot cross

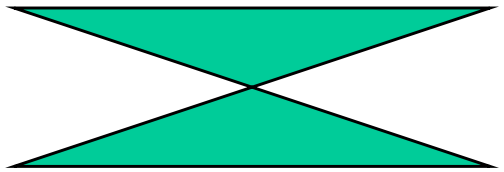
  Convex: All points on line segment between two points in a polygon are also in the polygon

  Flat: all vertices are in the same plane

- User program can check if above true

  OpenGL will produce output if these conditions are violated but it may not be what is desired

- **Triangles satisfy all conditions**



nonsimple polygon

nonconvex polygon

Attributes

- Attributes are part of the **OpenGL State** and determine the appearance of **objects**

  Color (points, lines, polygons)

  Size and width (points, lines)

  Stipple pattern (lines, polygons)

  Polygon mode

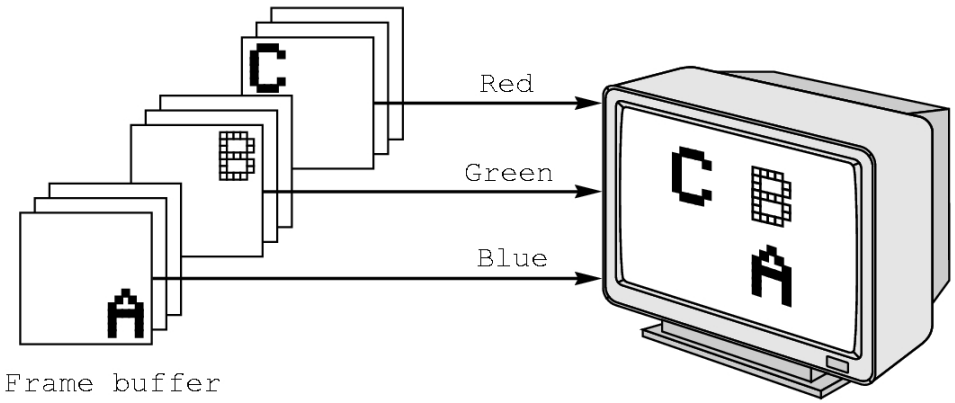
  - **Display as filled: solid color or stipple pattern**
  - **Display edges**
  - **Display vertices**

RGB color

Each color component is stored separately in the frame buffer

Usually 8 bits per component in buffer

Note in **`glColor3f`** the color values range from 0.0 (none) to 1.0 (all), whereas in **`glColor3ub`** the values range from 0 to 255

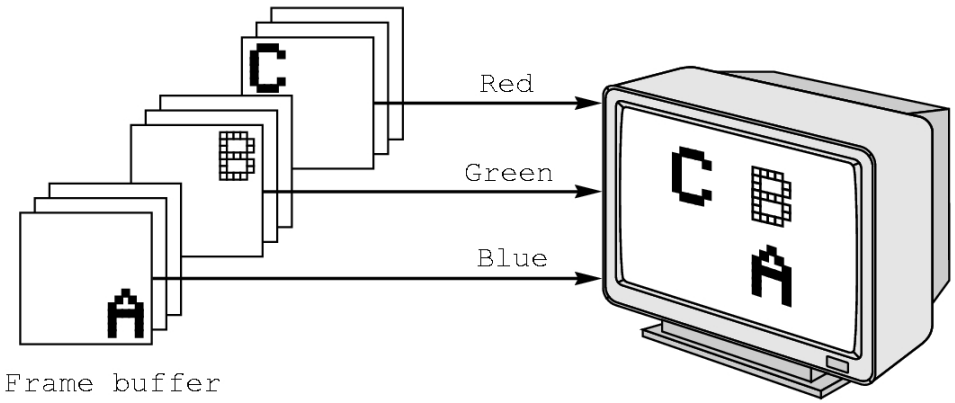


RGB color

Note in **glColor3f** the color values range from 0.0 (none) to 1.0 (all), whereas in **glColor3ub** the values range from 0 to 255

glColor**3f**(1.0, 0.0, 0.0);

glClearColor(1.0, 1.0, 1.0, 1.0); white



Color and State

Light parameters
Material Properties
Texture
Normals
ModelView Matrix
Projection Matrix
Color

- **The color as set by `glColor` becomes part of the State and will be used until changed**

  Colors and other attributes are not part of the **object** but **are assigned when the object is rendered**

- We can create conceptual ***vertex*** *colors* by code such as

```
glColor
glVertex

glColor
glVertex
```

Smooth Color

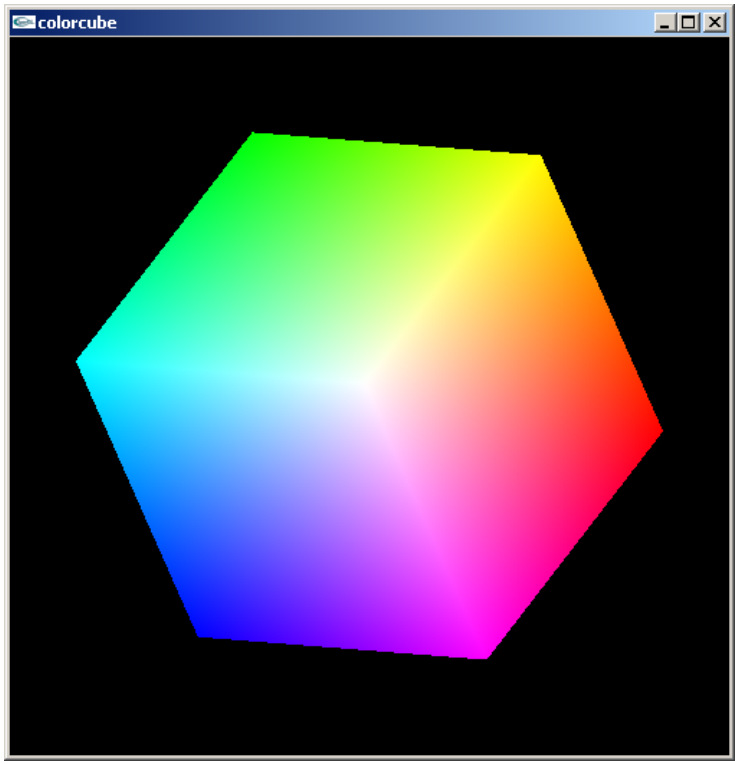
Default is *smooth* shading

OpenGL interpolates vertex colors across visible polygons

Alternative is *flat shading*

Color of first vertex determines fill color

```
glShadeModel
   (GL_SMOOTH)
 or
   (GL_FLAT   )
```

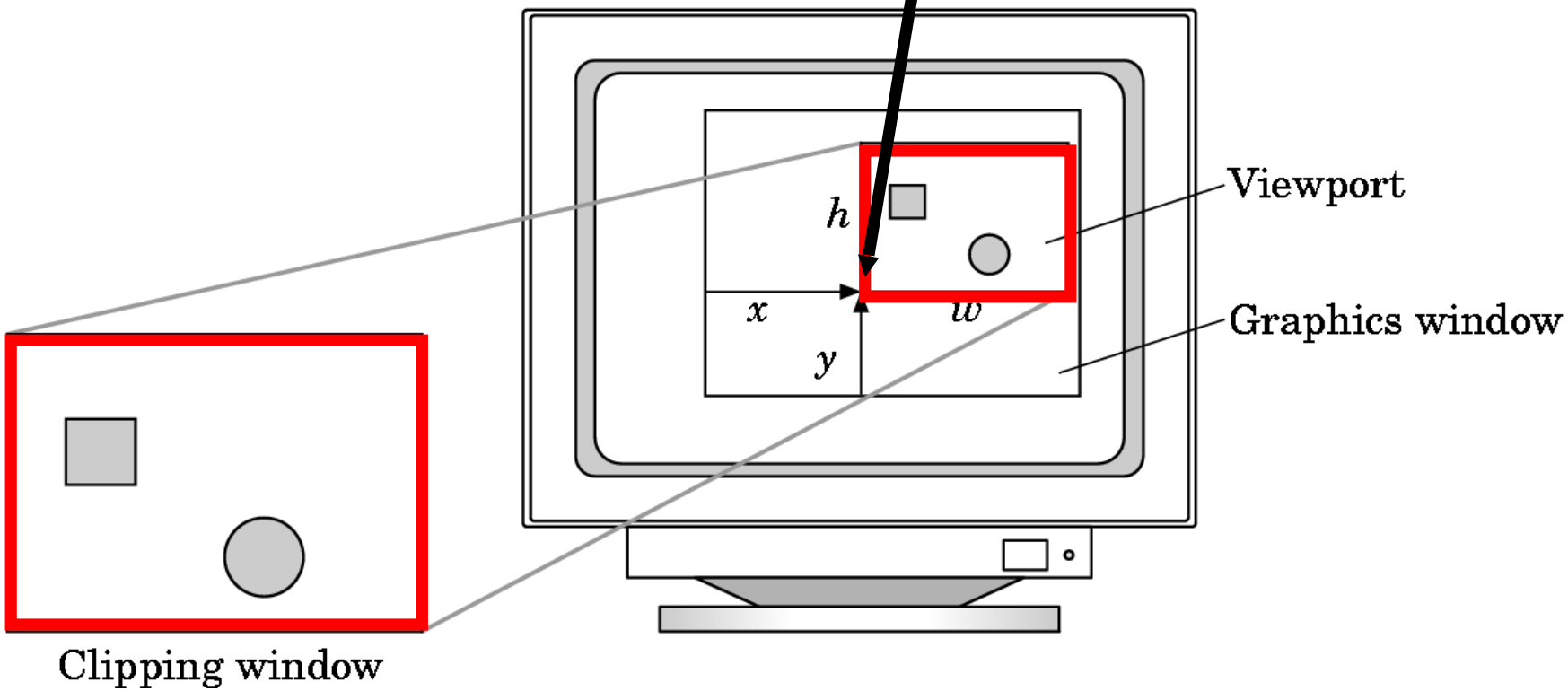


Viewports

- Do not have use the entire window for the image:

```
glViewport(x,y,w,h)
```

- Values in pixels (screen coordinates)



Objectives

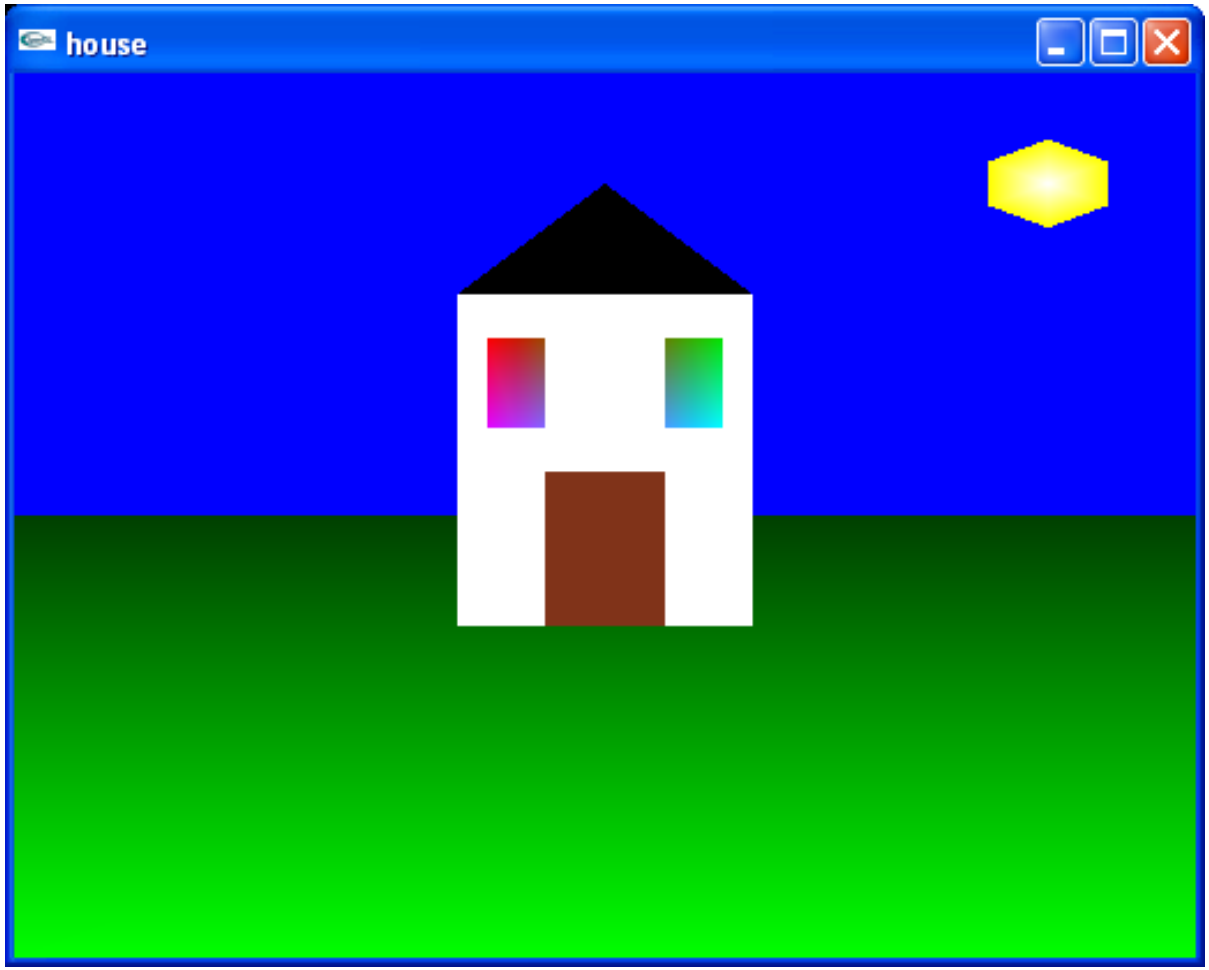
- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a state machine
- Functions
  - Types
  - Formats
- Simple program

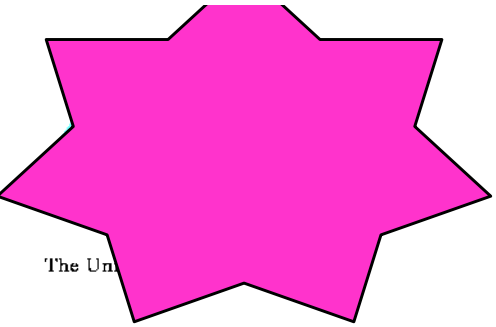
Programming with OpenGL Example

Objectives

- Develop a **more sophisticated example**

  house.c





The Un

Name: _____

Total score: [ ]

Class PARTICIPATION on Lecture 2.doc **ANSWER SHEET**

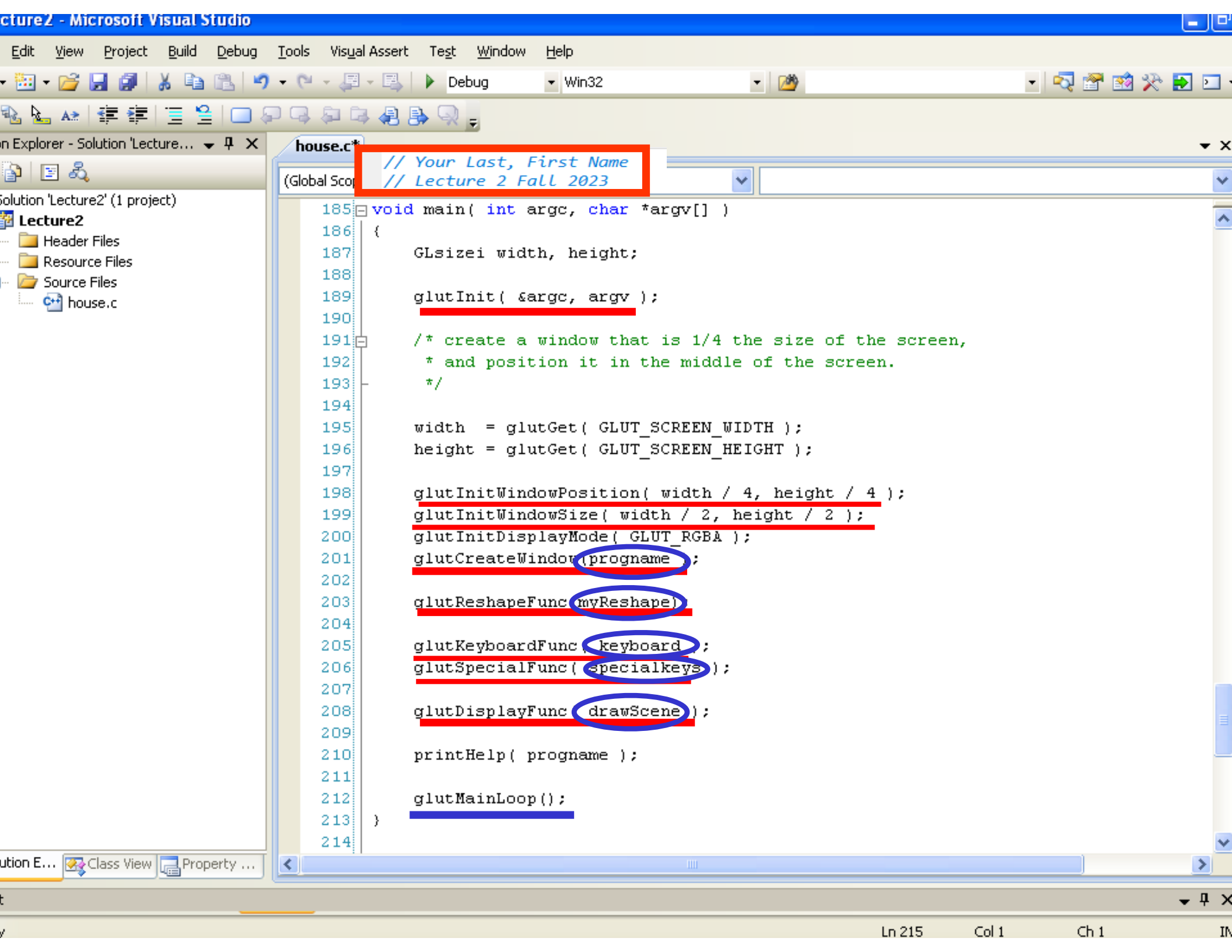
(Out of 100 points. Please record your own total score!)

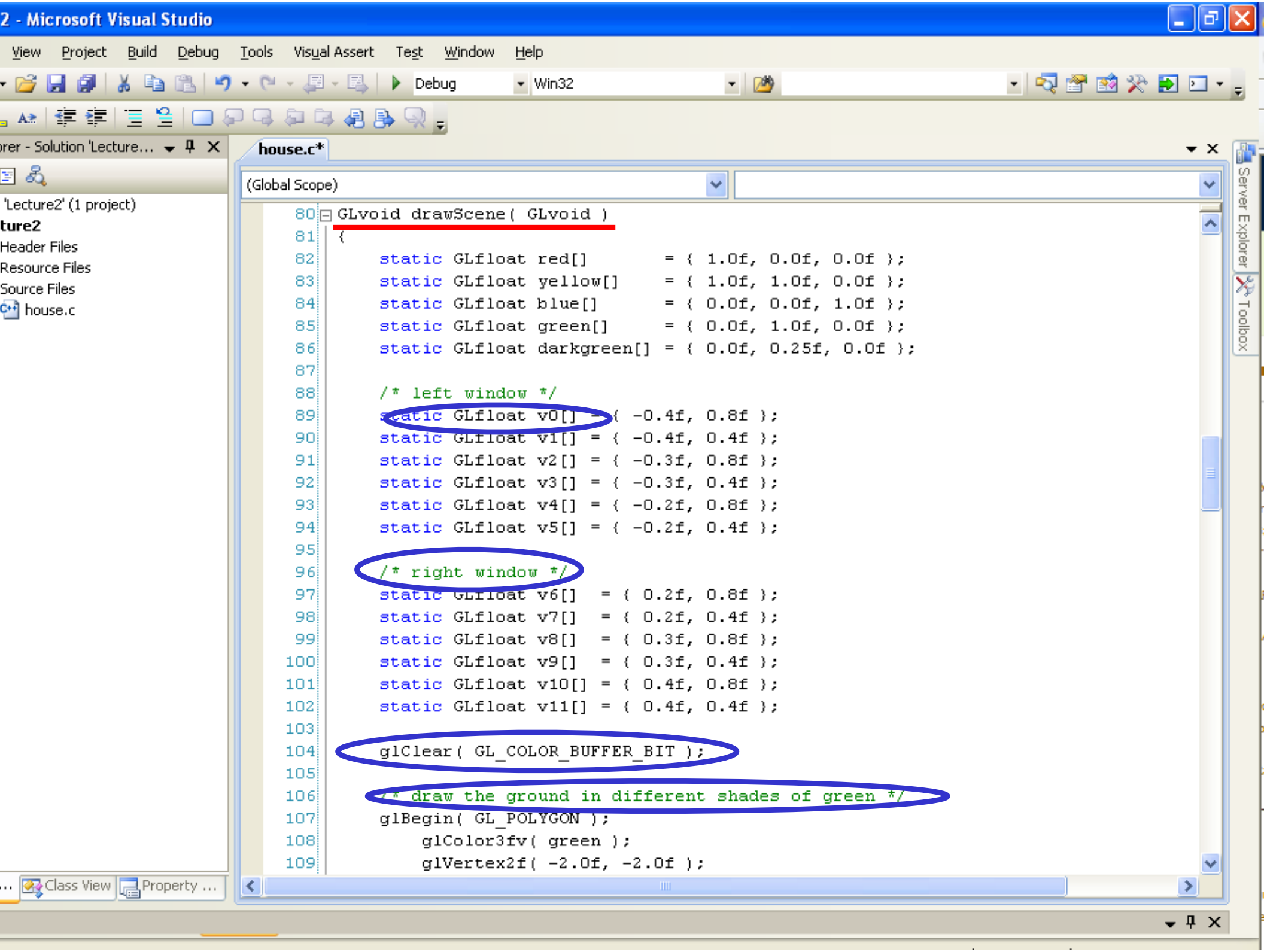
(Attach as score.doc!)

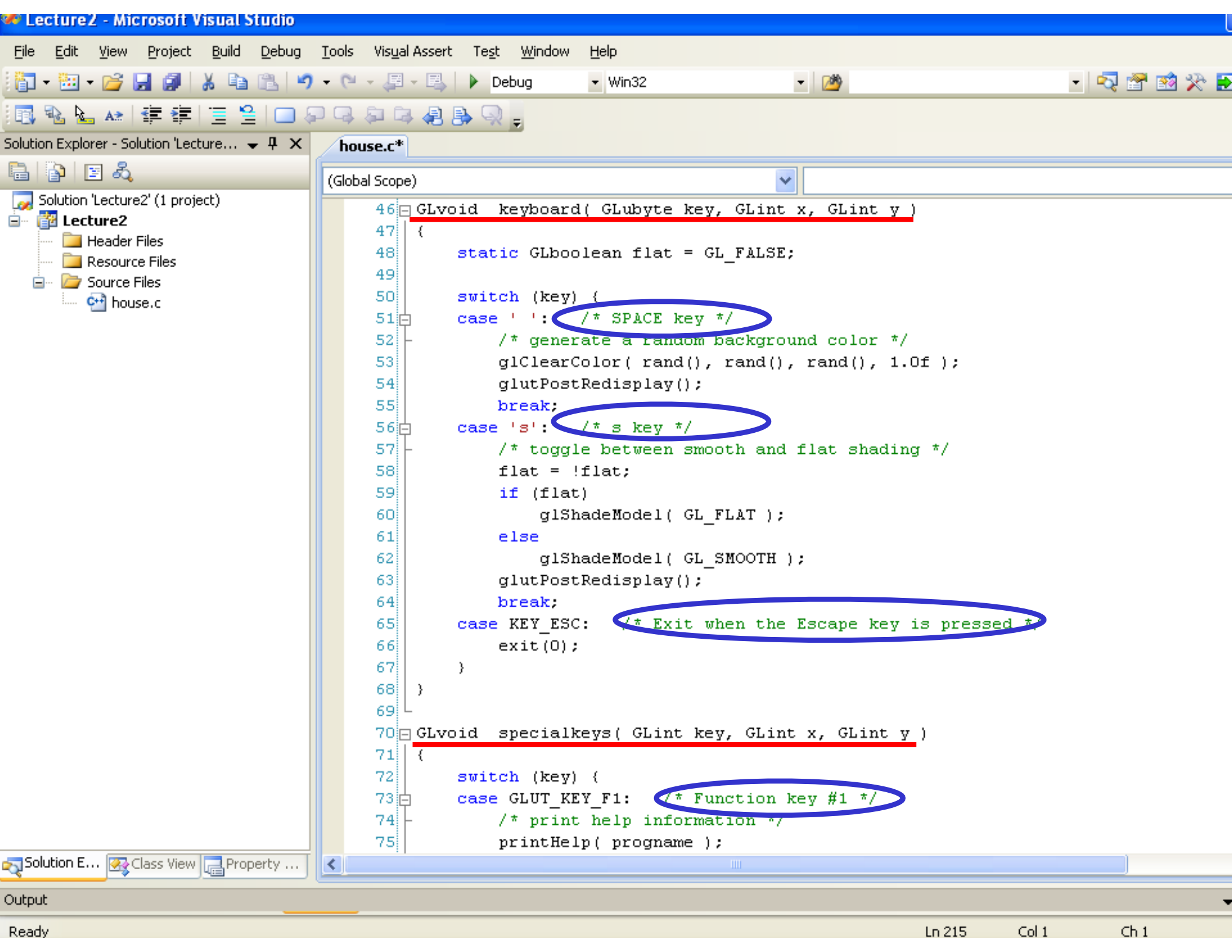
II. Download the house.c from CANVAS, build a Lecture2 C++ Empty Project and add it.

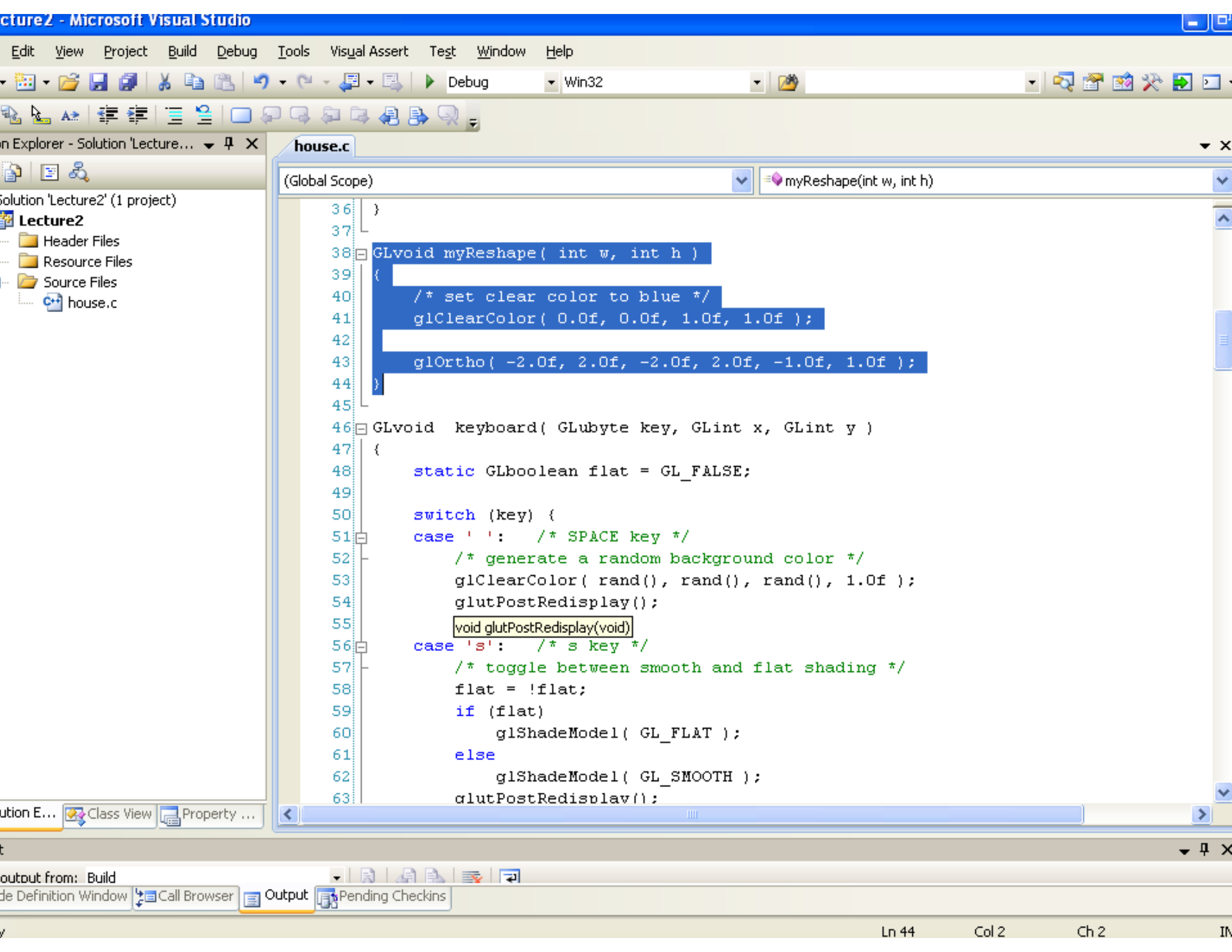
b. Build and run the project.
(Take print screen under 1. b) (25 points)

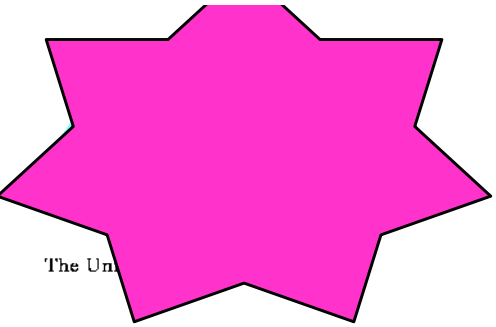
Class Participation II. b.!


cture2 - Microsoft Visual Studio
Edit View Project Build Debug Tools Visual Assert Test Window Help
Debug
Win32
n Explorer - Solution 'Lecture...
Solution 'Lecture2' (1 project)
Lecture2
Header Files
Resource Files
Source Files
house.c
house.c*
// Your Last, First Name
// Lecture 2 Fall 2023
(Global Scop
185 void main(int argc, char *argv[])
186 {
187 GLsizei width, height;
188
189 glutInit(&argc, argv);
190
191 /* create a window that is 1/4 the size of the screen,
192 * and position it in the middle of the screen.
193 */
194
195 width = glutGet(GLUT_SCREEN_WIDTH);
196 height = glutGet(GLUT_SCREEN_HEIGHT);
197
198 glutInitWindowPosition(width / 4, height / 4);
199 glutInitWindowSize(width / 2, height / 2);
200 glutInitDisplayMode(GLUT_RGBA);
201 glutCreateWindow(progname);
202
203 glutReshapeFunc(myReshape);
204
205 glutKeyboardFunc(keyboard);
206 glutSpecialFunc(specialkeys);
207
208 glutDisplayFunc(drawScene);
209
210 printHelp(progname);
211
212 glutMainLoop();
213 }
214
Class View
Property ...
Ln 215
Col 1
Ch 1


2 - Microsoft Visual Studio
View Project Build Debug Tools Visual Assert Test Window Help
Debug
Win32
house.c*
(Global Scope)
'Lecture2' (1 project)
ture2
Header Files
Resource Files
Source Files
house.c
Class View
Property ...
Server Explorer
Toolbox
80 GLvoid drawScene(GLvoid)
81 {
82 static GLfloat red[] = { 1.0f, 0.0f, 0.0f };
83 static GLfloat yellow[] = { 1.0f, 1.0f, 0.0f };
84 static GLfloat blue[] = { 0.0f, 0.0f, 1.0f };
85 static GLfloat green[] = { 0.0f, 1.0f, 0.0f };
86 static GLfloat darkgreen[] = { 0.0f, 0.25f, 0.0f };
87
88 /* left window */
89 static GLfloat v0[] = { -0.4f, 0.8f };
90 static GLfloat v1[] = { -0.4f, 0.4f };
91 static GLfloat v2[] = { -0.3f, 0.8f };
92 static GLfloat v3[] = { -0.3f, 0.4f };
93 static GLfloat v4[] = { -0.2f, 0.8f };
94 static GLfloat v5[] = { -0.2f, 0.4f };
95
96 /* right window */
97 static GLfloat v6[] = { 0.2f, 0.8f };
98 static GLfloat v7[] = { 0.2f, 0.4f };
99 static GLfloat v8[] = { 0.3f, 0.8f };
100 static GLfloat v9[] = { 0.3f, 0.4f };
101 static GLfloat v10[] = { 0.4f, 0.8f };
102 static GLfloat v11[] = { 0.4f, 0.4f };
103
104 glClear(GL_COLOR_BUFFER_BIT);
105
106 /* draw the ground in different shades of green */
107 glBegin(GL_POLYGON);
108 glColor3fv(green);
109 glVertex2f(-2.0f, -2.0f);


Lecture2 - Microsoft Visual Studio
File Edit View Project Build Debug Tools Visual Assert Test Window Help
Debug
Win32
Solution Explorer - Solution 'Lecture...
Solution 'Lecture2' (1 project)
Lecture2
Header Files
Resource Files
Source Files
house.c
house.c*
(Global Scope)
46 GLvoid keyboard(GLubyte key, GLint x, GLint y)
47 {
48 static GLboolean flat = GL_FALSE;
49
50 switch (key) {
51 case ' ': /* SPACE key */
52 /* generate a random background color */
53 glClearColor(rand(), rand(), rand(), 1.0f);
54 glutPostRedisplay();
55 break;
56 case 's': /* s key */
57 /* toggle between smooth and flat shading */
58 flat = !flat;
59 if (flat)
60 glShadeModel(GL_FLAT);
61 else
62 glShadeModel(GL_SMOOTH);
63 glutPostRedisplay();
64 break;
65 case KEY_ESC: /* Exit when the Escape key is pressed */
66 exit(0);
67 }
68 }
69
70 GLvoid specialkeys(GLint key, GLint x, GLint y)
71 {
72 switch (key) {
73 case GLUT_KEY_F1: /* Function key #1 */
74 /* print help information */
75 printHelp(progname);
Solution E...
Class View
Property ...
Output
Ready
Ln 215
Col 1
Ch 1


cture2 - Microsoft Visual Studio
Edit View Project Build Debug Tools Visual Assert Test Window Help
Debug
Win32
n Explorer - Solution 'Lecture...
Solution 'Lecture2' (1 project)
Lecture2
Header Files
Resource Files
Source Files
house.c
house.c
(Global Scope)
myReshape(int w, int h)
36 }
37
38 GLvoid myReshape(int w, int h)
39 {
40 /* set clear color to blue */
41 glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
42
43 glOrtho(-2.0f, 2.0f, -2.0f, 2.0f, -1.0f, 1.0f);
44 }
45
46 GLvoid keyboard(GLubyte key, GLint x, GLint y)
47 {
48 static GLboolean flat = GL_FALSE;
49
50 switch (key) {
51 case ' ': /* SPACE key */
52 /* generate a random background color */
53 glClearColor(rand(), rand(), rand(), 1.0f);
54 glutPostRedisplay();
55 void glutPostRedisplay(void)
56 case 's': /* s key */
57 /* toggle between smooth and flat shading */
58 flat = !flat;
59 if (flat)
60 glShadeModel(GL_FLAT);
61 else
62 glShadeModel(GL_SMOOTH);
63 glutPostRedisplay();
ution E... Class View Property ...
Output from: Build
de Definition Window Call Browser Output Pending Checkins
Ln 44 Col 2 Ch 2



The Un

Name: _____

Total score: [ ]

Class PARTICIPATION on Lecture 2.doc **ANSWER SHEET**

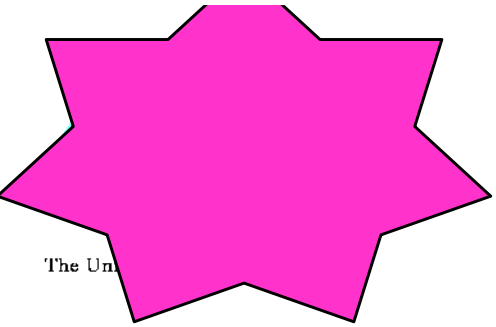
(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

II. Download the house.c from CANVAS, build a Lecture2 C++ Empty Project and add it.

c. How can you control the shading options?
(Take print screen under 1. c) (25 points)

Class Participation II. c.!



The Un

Name: _____

Total score: [ ]

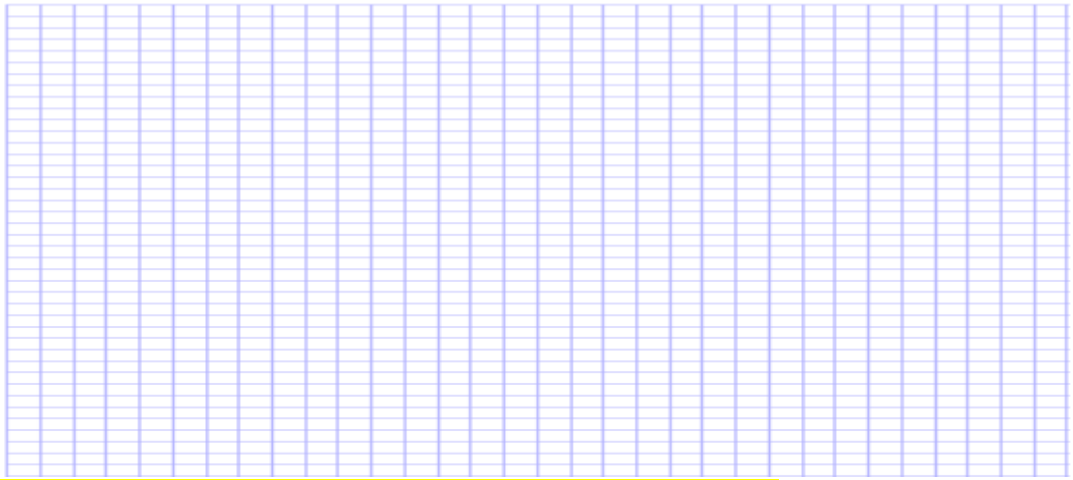
Class PARTICIPATION on Lecture 2.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

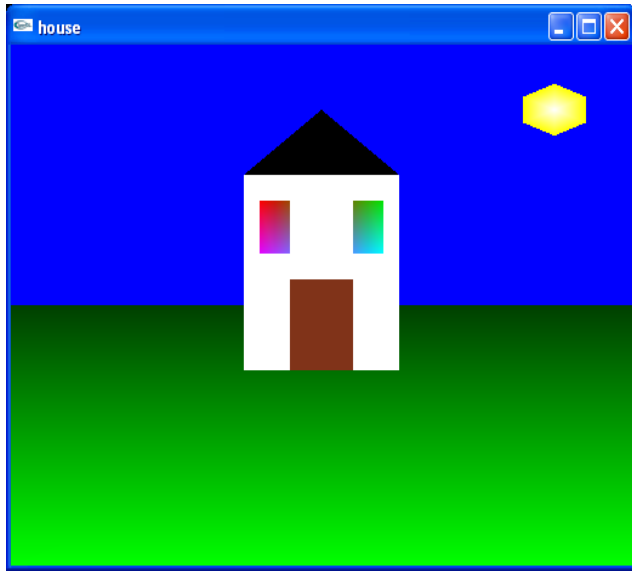
(Attach as score.doc!)

II. Download the house.c from CANVAS, build a Lecture2 C++ Empty Project and add it.

d. Can you draw the coordinates for the house frame, windows and roof? (25 points)



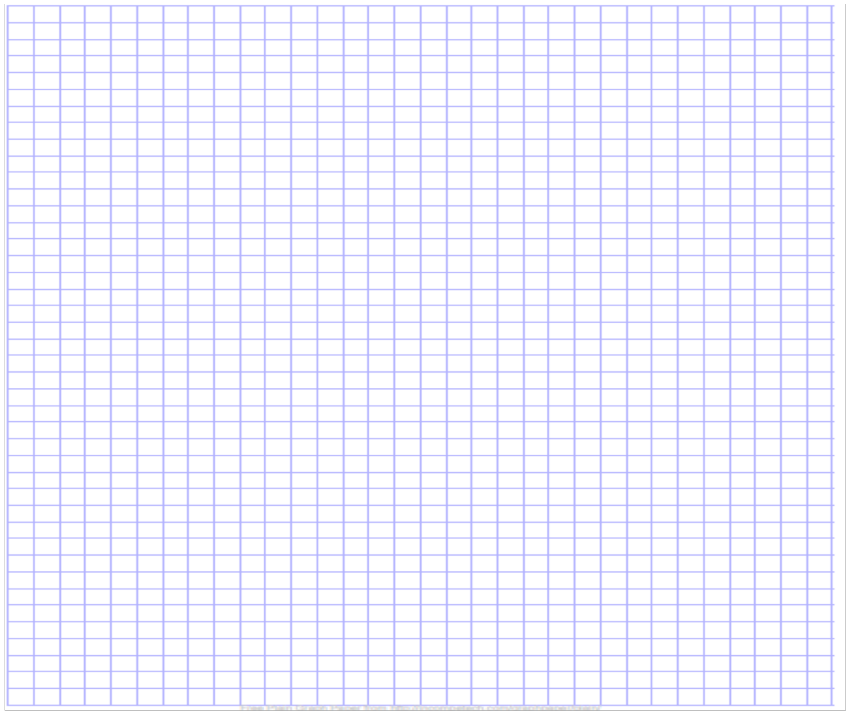
Class Participation II. d.!

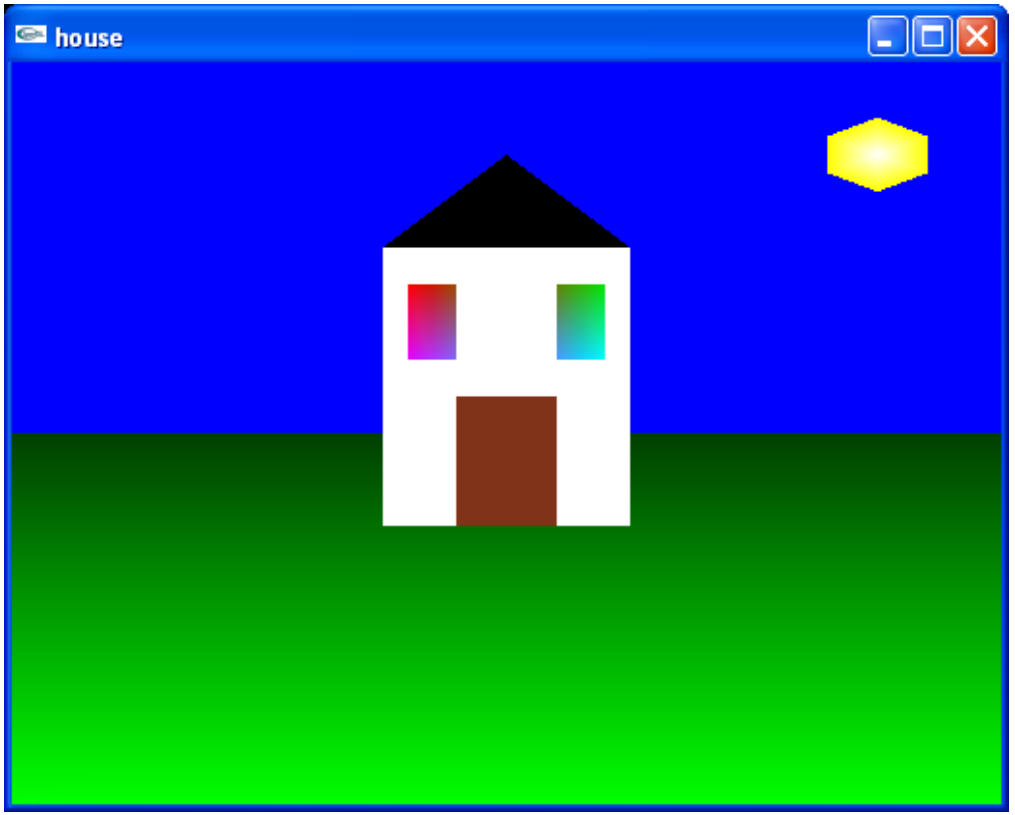


house.c

The Un

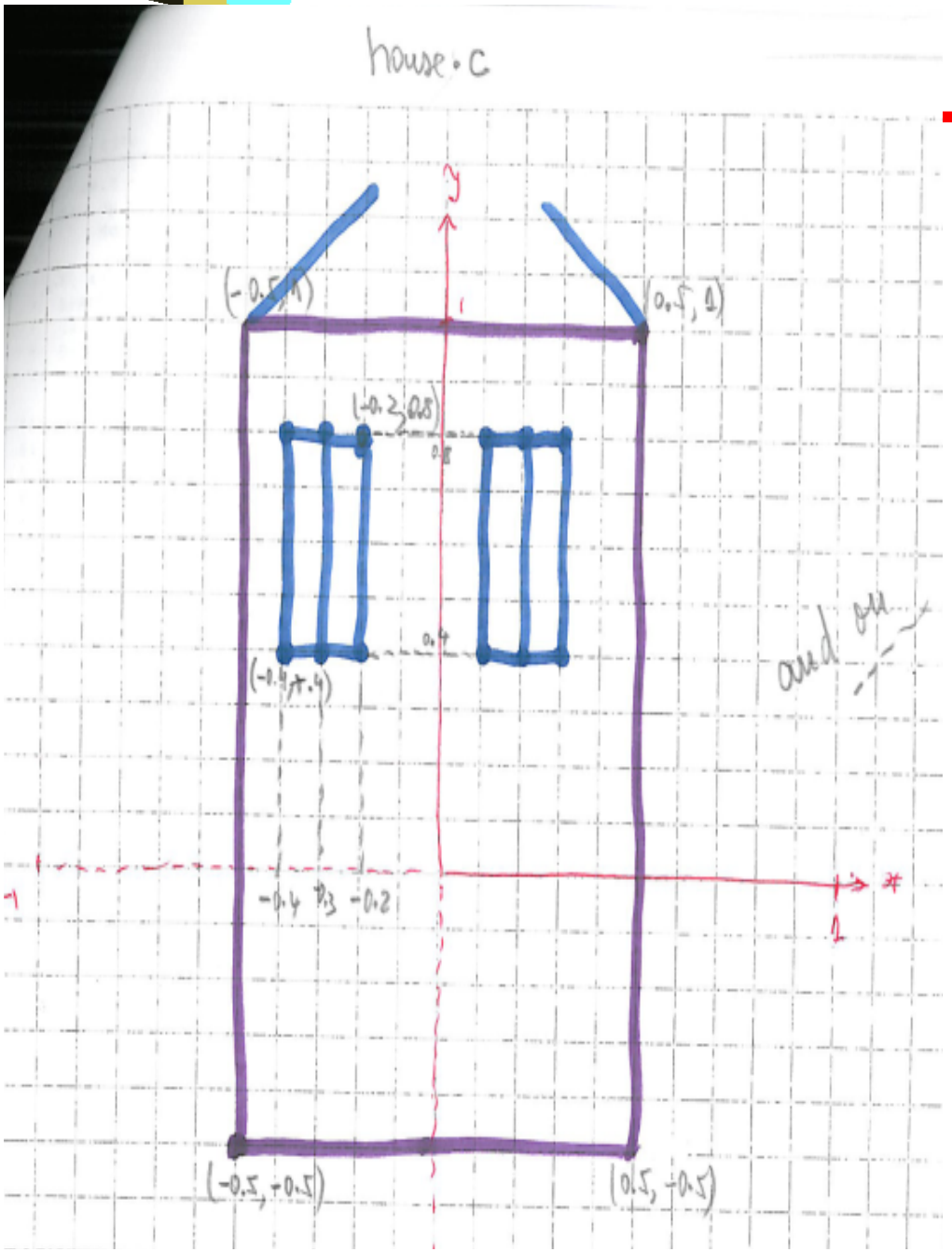
d. Can you draw the coordinates for the house frame, windows and roof? (25 points)

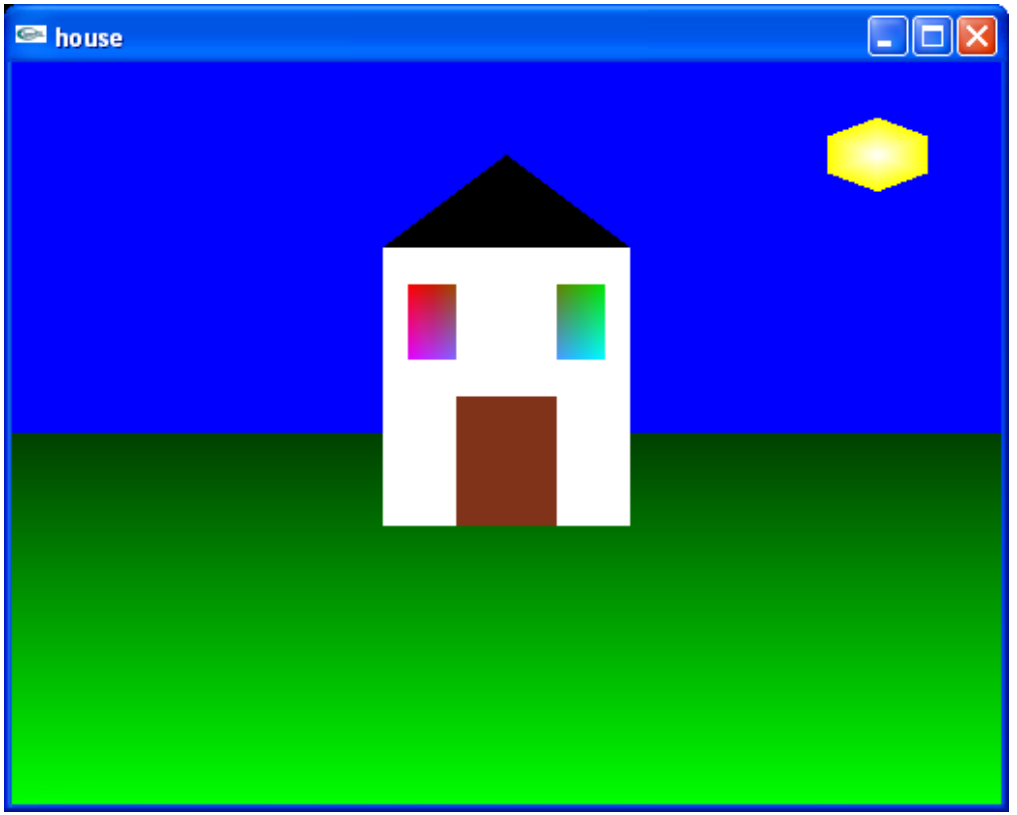




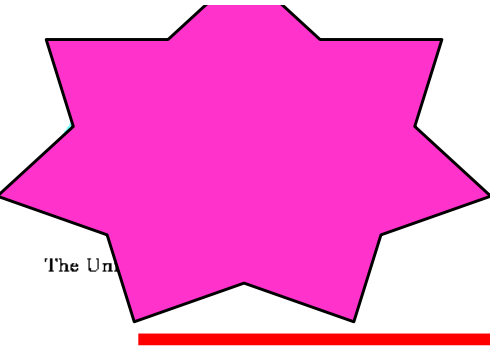
house.c
Lecture 2 Class Participation II. d

house.c





house.c
Lecture 2 Class Participation 2



You will be prompted when to Upload completed document to CANVAS as **score.doc** (example 100.doc).

Warning:

TA, at random, will inspect the Uploaded document.

If you score is not honestly entered you will get a zero.

Please rename document to score.doc (example 100.doc)
Warning: if you score is not honestly honest you will get a zero.

Name: _____

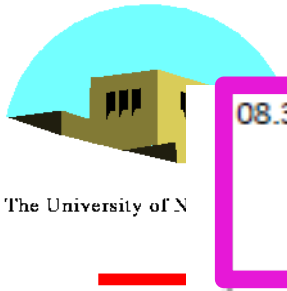
Total score: ☐

Class PARTICIPATION on Lecture 2.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

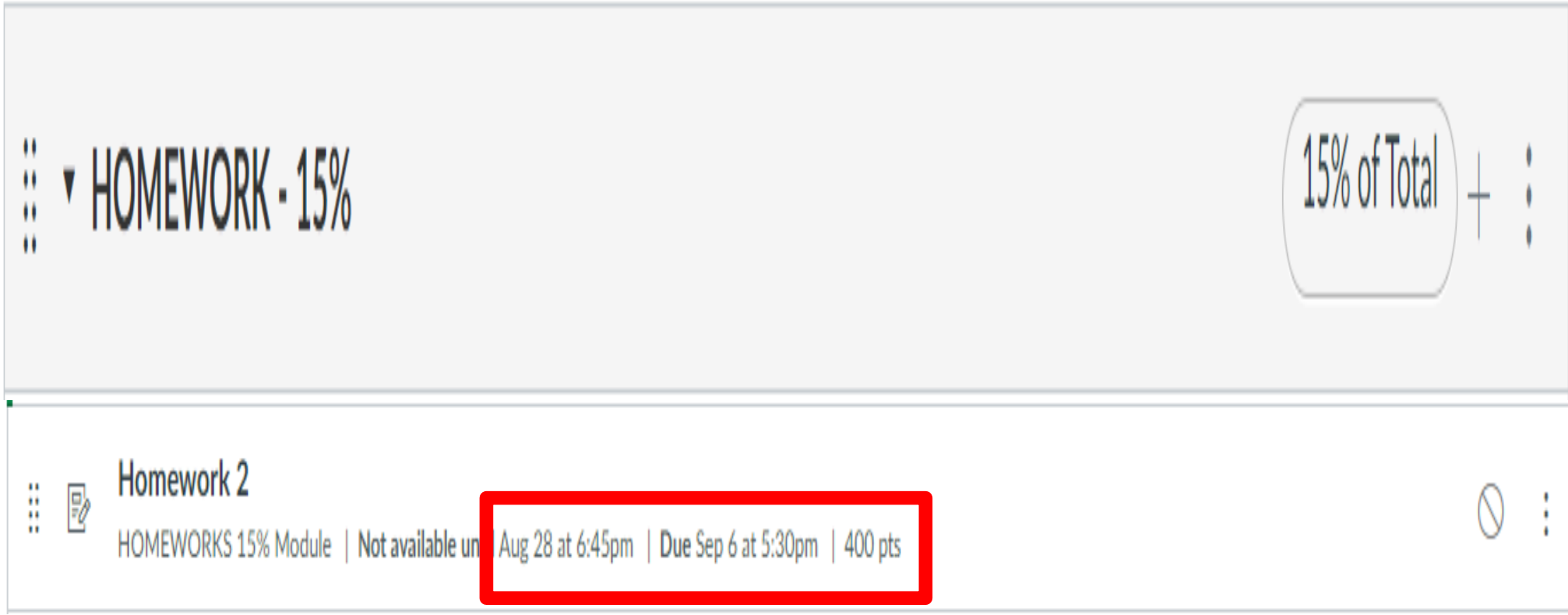
THE END



08.30.2023 (W 5:30 to 7) (4)		Math Review 1
09.06.2023 (W 5:30 to 7) (5)	Homework 2	Lecture 3
09.11.2023 (M 5:30 to 7) (6)		Math Review 2
09.13.2023 (W 5:30 to 7) (7)	Homework 3	Lecture 4
09.18.2023 (M 5:30 to 7) (8)		PROJECT 1
09.20.2023 (W 5:30 to 7) (9)		EXAM 1 REVIEW
09.25.2023 (M 5:30 to 7) (10)		EXAM 1

At 6:30 PM NEXT.

The University of New Mexico



VH Publish.

AccessUH | Homework 2

canvas.uh.edu/courses/4717/assignments/105857

COSC4370 23469 - Interactive Computer ... › Assignments › Homework 2

Student

Account
Dashboard
Courses
Calendar
Inbox
History
Studio
Commons
Help

H 2023 Fall 1

Home
Announcements
Assignments
Discussions
Grades
People
Pages
Files
Syllabus
Outcomes
Rubrics
Quizzes
Modules
Collaborations
New Analytics
Studio
LockDown Browser
Course Reserves

Homework 2

Publish | Edit

Homework 2.docx

The HOMEWORK is due by 5:30 PM of the due date class.

The HOMEWORK will have a theoretical and a programming part.

You also are to submit the zip of the programming part - HOMEWORK2 (Visual Studio 2019 PROJECT (top FOLDER)) to CANVAS.

Rename Homework 2.doc t score.OPENGL.docx

You can do the programming part using WEBGL. You are to submit the zip of the programming part - HOMEWORK2 (WEBGL PROJECT (top FOLDER)) to CANVAS.

Rename Homework 2.doc t score.WEBGL.docx

(MUST BE A .DOCX DOCUMENT)

Points 400

Submitting a file upload

Due	For	Available from	Until
Sep 6 at 5:30pm	Everyone	Aug 28 at 6:45pm	Sep 6 at 5:30pm

Name: _____

Homework 2

(400 points)

Hours spent:

The homework is to be turned in before 5:30 PM of the due date class.

Also, an implementation in Visual Studio 2019 or WEBGL is required, thus you are to submit the ZIPPED project (the top Homework1 folder) to CANVAS.

I UNDERSTAND THAT TURNING ANOTHER's WORK IN is CHEATING.
I UNDERSTAND THAT ANY KIND OF DISSEMINATION of this WORK is CHEATING.

I CERTIFY THAT THE HOMEWORKs SOLUTIONs ARE MY OWN WORK!

SIGNATURE: V X ?

HOMEWORK CHECKLIST (YOU MUST GRADE YOURSELF!):

1. A. 300 points – please count! 300 points
2. B. 100 points – please count! 100 points

3. Homework2.zip NOT submitted to CANVAS? -100 points

PLEASE ENTER YOUR GRADE IN THIS BOX:

Please rename Homework 2.doc to either score.OPENGL.doc or score.WEBGL.doc

(Example 100.OPENGLdoc)

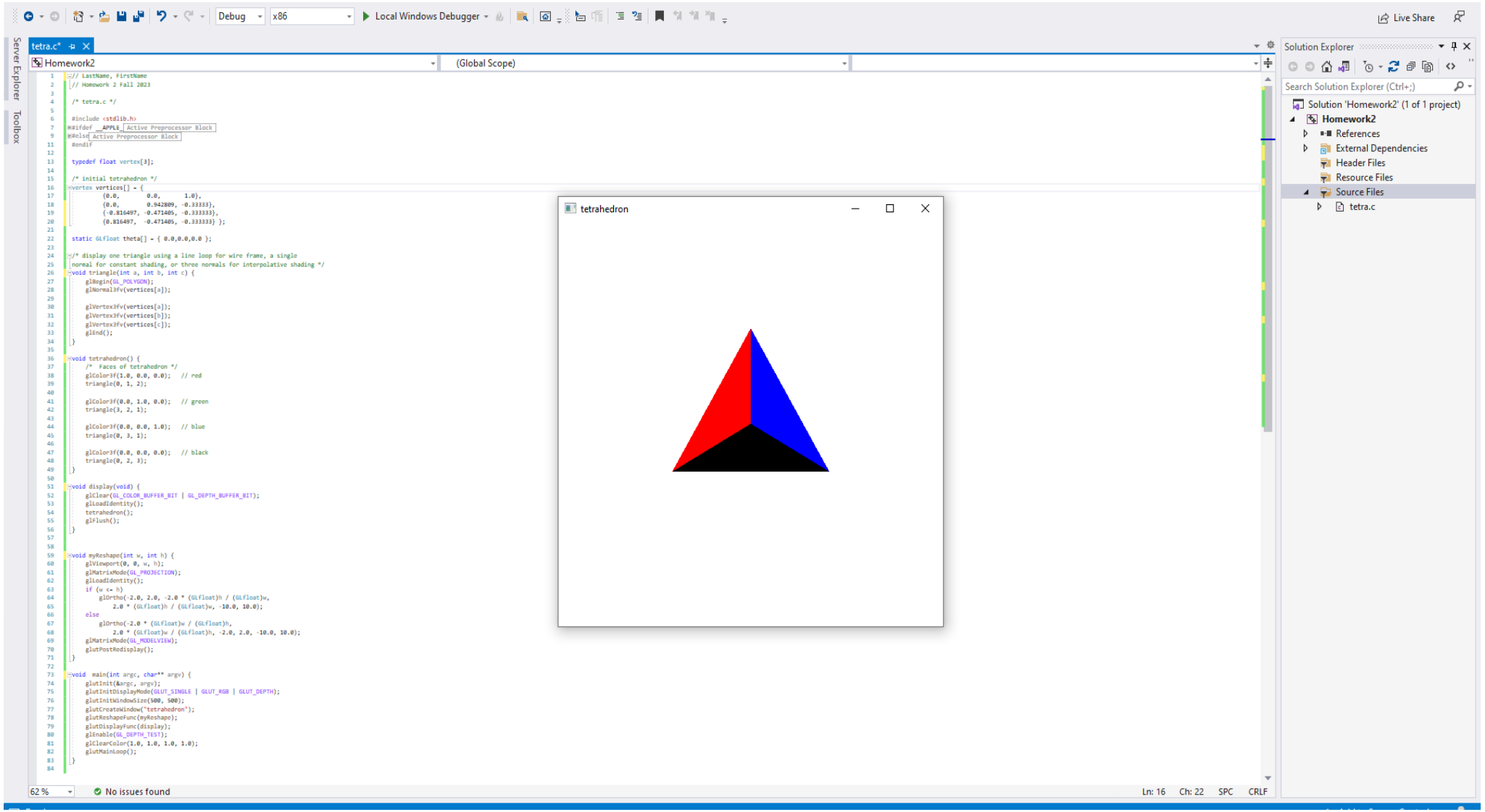
Warning: if your score is not honest you will get a zero.

B. (100 pts) Visual Studio 2019 C++ Project

B1. (10 points) Create Visual Studio 2019 C++, Empty Project, Homework2:

tetra.c

```
// Your Last, First Name
// Homework 2 Fall 2023

/* tetra.c */

#include <stdlib.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
```



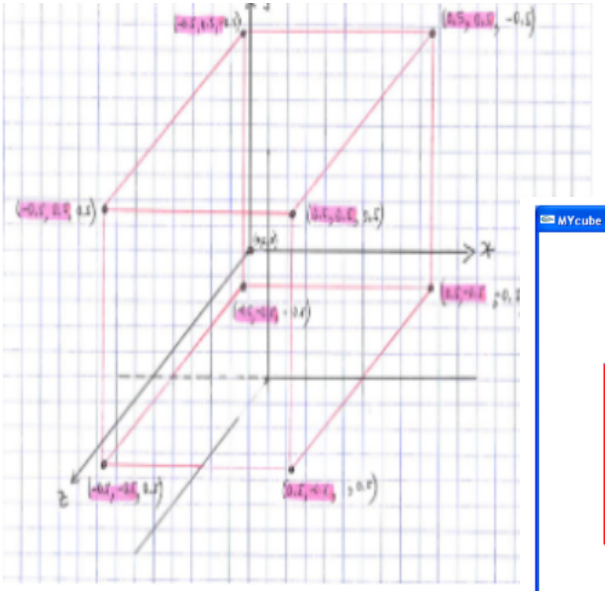
B. (100 pts) Visual Studio 2019 C++ Project

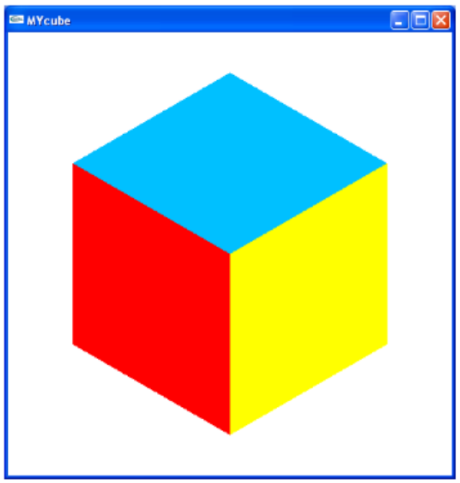
MYcube.c

// **Your Last, First Name**
// **Homework 2 Fall 2023**

//MYCube.c

B2. The default viewing volume in openGL is a cube of side length 2 centered at (0, 0, 0). Thus, objects have to be inside this volume in order to be displayed:





After each modification, build the project to isolate faults.

1. Modify CreateWindow from "tetrahedron" to "MYcube".
2. Modify tetrahedron **function call** to MYcube.
3. Build MYcube just like the one above except that it is bounded by (-0.5, 0.5) in x, y, and z. (thus the corner **(-1,-1,-1)** becomes **(-0.5, -0.5, -0.5)**). vertices vertex[] will have 8 vertices. Please use the vertices as specified below:

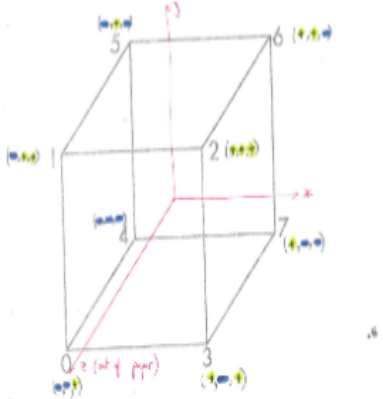
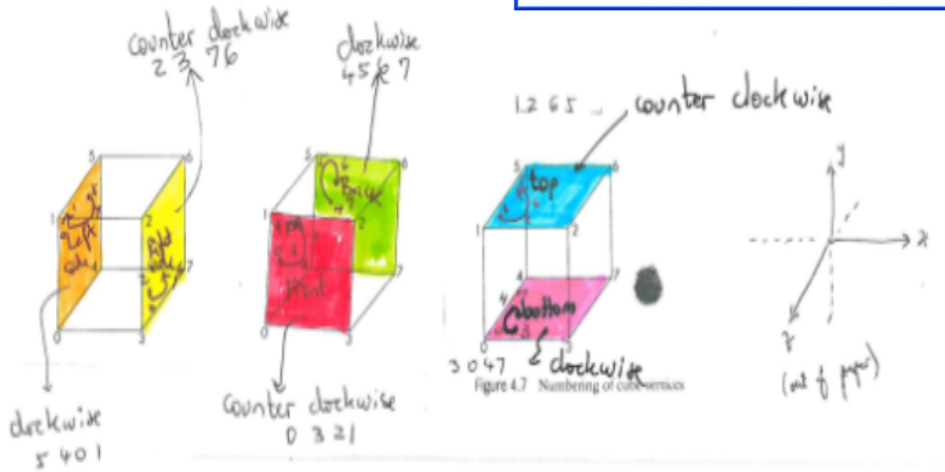
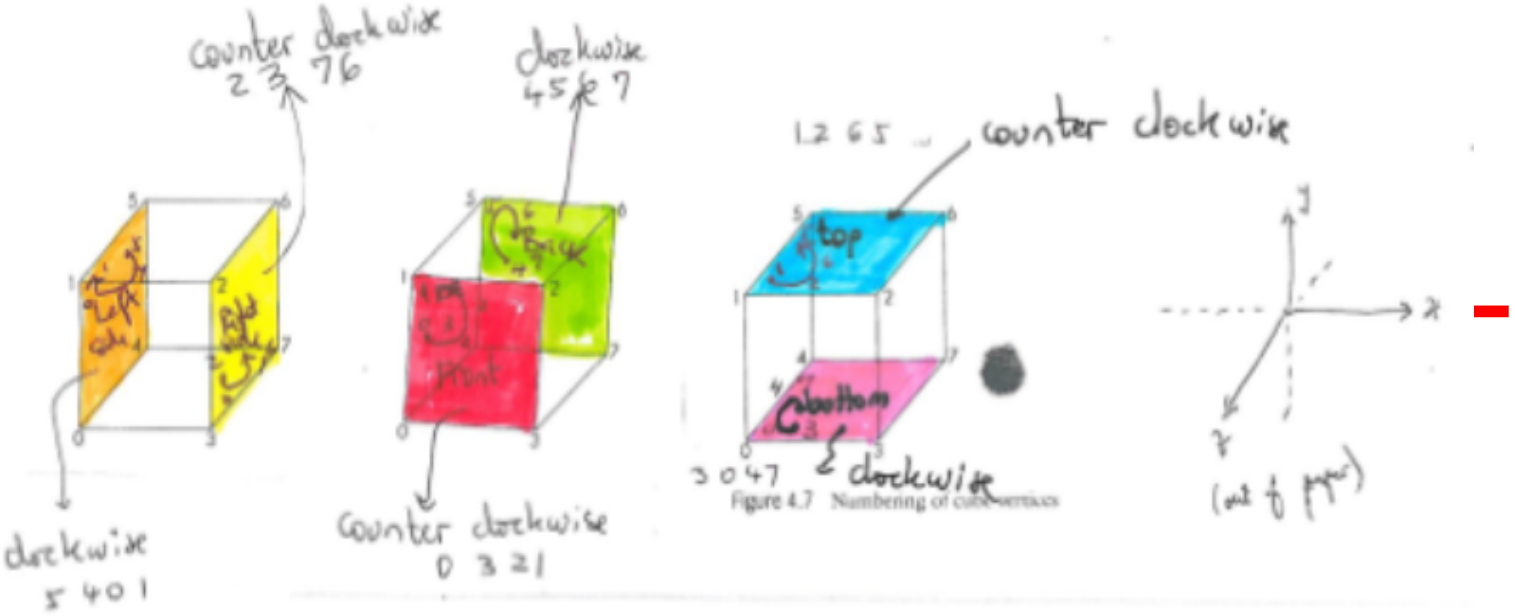


Figure 4.7 Numbering of cube vertices

4. Each face of the cube will be created by calling a polygon function instead of triangle (glBegin is GL_POLYGON) and passing to it the vertices that make up that polygon in the following order. Please place a comment of which face you are drawing, and use the following colors:

front (`glColor3f(1.0,0.0,0.0);` `//red`),
back (`glColor3f(0.0,1.0,0.0);` `//green`),
top (`glColor3f(0.0,0.75388,1.0);` `//blue`),
bottom (`glColor3f(1.0,0.2,0.7);` `//deep pink`),
right side (`glColor3f(1.0,1.0,0.0);` `// yellow`),
left side (`glColor3f(1.0,0.75,0.0);` `// orange`).



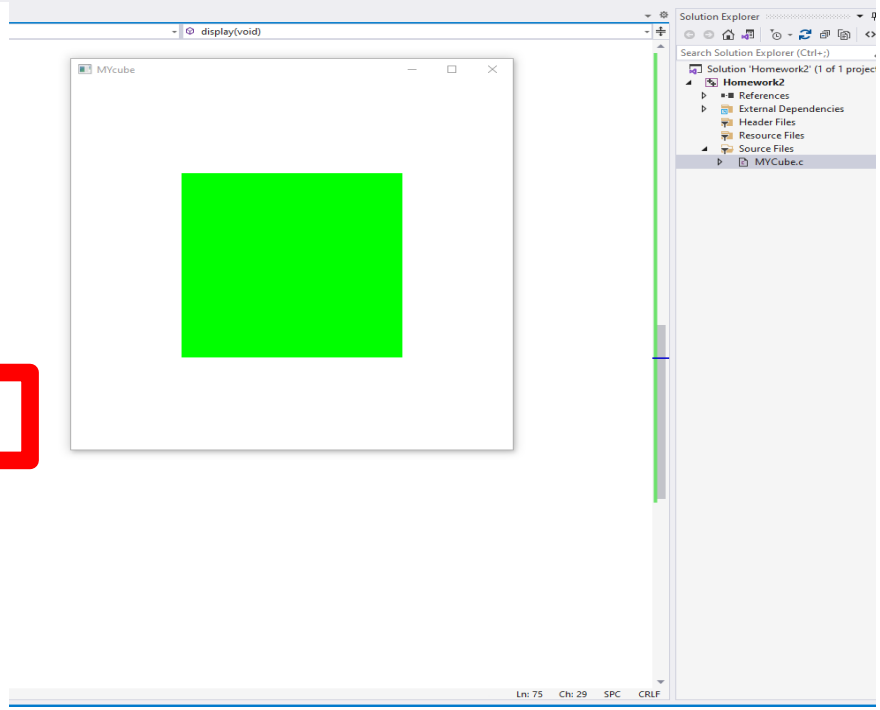


MYcube.c

// **Your Last, First Name**
// **Homework 2 Fall 2023**

```
//MYCube.c
```

```
71  void display(void)
72  {
73      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
74      glLoadIdentity();
75      gluLookAt(0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
76      MYcube();
77      glFlush();
78  }
```



Camera at z = -1 looking at the center of the cube 0, 0 ,0

At 6:45 PM.

End Class 3

VH, Download Attendance Report
Rename it:
8.28.2023 Attendance Report FINAL

VH, upload Lecture 2 to CANVAS.