CSCI3260
Principles of Computer Graphics

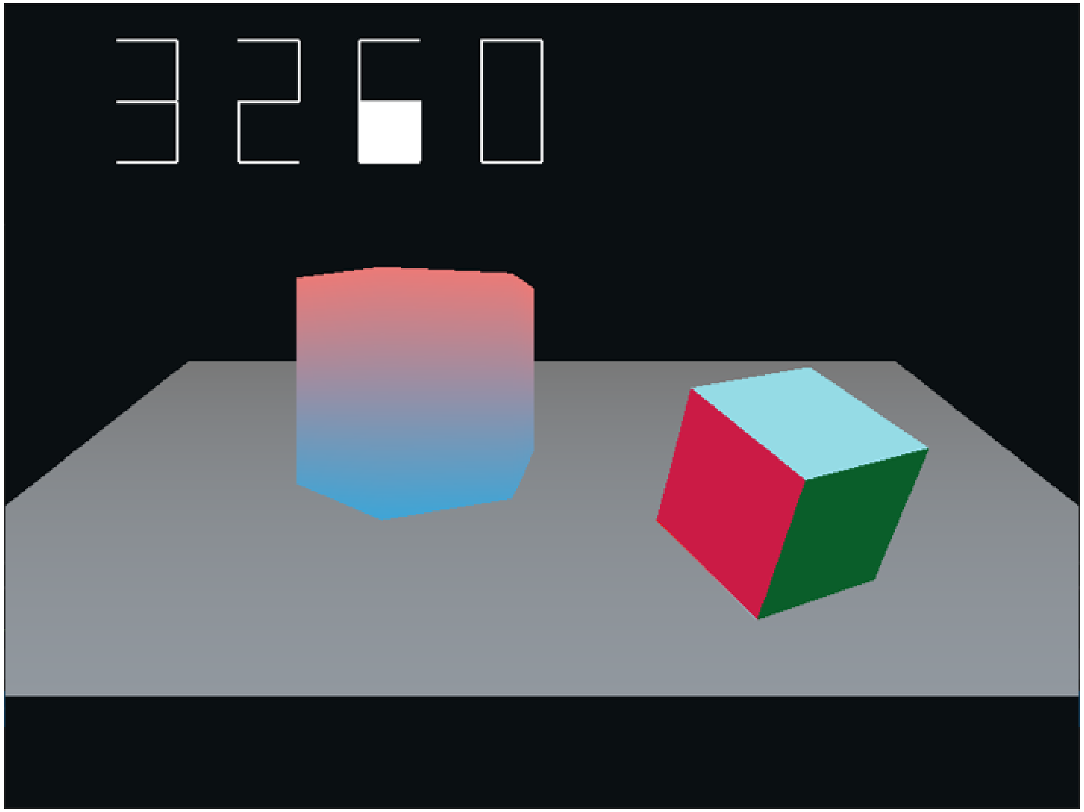
---------Tutorial 3
XU Jiaqi

OUTLINE

- Basic requirements in Assignment 1

- How to render a 3D object

Assignment 1:



+ user interaction

Assignment 1

Basic Requirements:

- OpenGL code should use the programmable pipeline with OpenGL 3.0+ instead of the fixed pipeline.
- Draw at least one 2D object and two 3D objects.
- Ensure at least one object is drawn with indexing;
- Create at least three keyboard and/or mouse events;
- Design object transformations, including rotation, translation and scaling;
- Use perspective projection to draw the scene and enable depth test to realize occlusion.

Basic Requirements:

- OpenGL code should use the programmable pipeline with OpenGL 3.0+ instead of the fixed pipeline

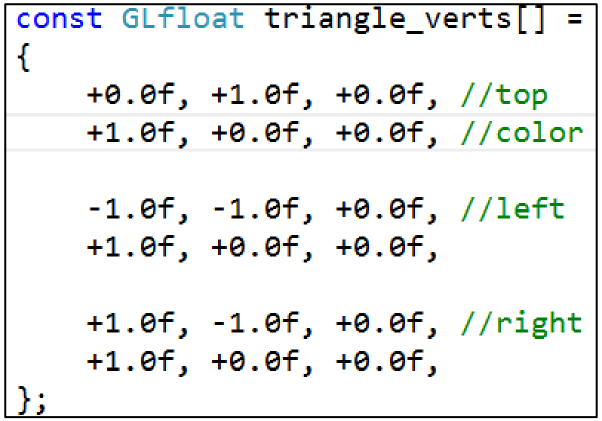
```
glBegin ( type ) ;
    glVertex3f ( … ) ;
    glVertex3f ( … ) ;
    glVertex3f ( … ) ;
    ……
glEnd() ;
```

```
glMatrixMode ( GL_MODELVIEW ) ;
glLoadIdentity () ;
glPushMatrix() ;
   glTranslatef ( ball_X , ball_Y , ball_Z ) ;
   glRotatef ( ball_ang , ball_dirX , ball_dirY , ball_dirZ ) ;
   glScalef ( ball_Sx , ball_Sy , ball_Sz ) ;
   Draw_ball() ;
glPopMatrix() ;
glPushMatrix() ;
   glTranslatef ( cube_X , cube_Y , cube_Z ) ;
   glRotatef ( cube_ang , cube_dirX , cube_dirY , cube_dirZ ) ;
   glScalef ( cube_Sx , cube_Sy , cube_Sz ) ;
   Draw_cube() ;
glPopMatrix() ;
```

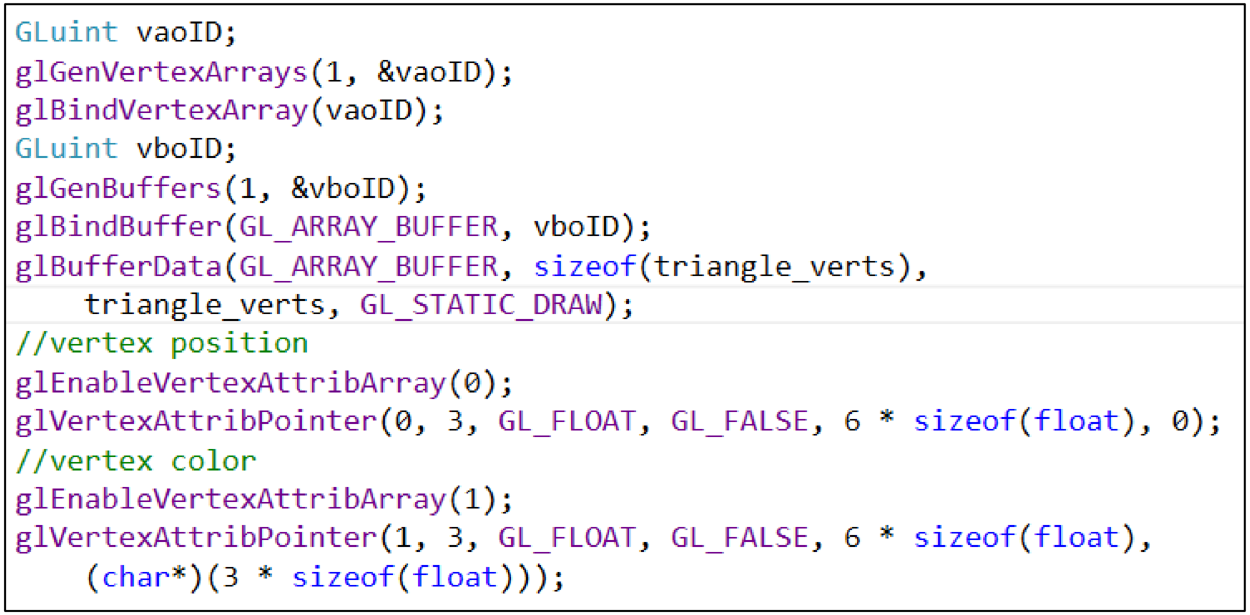
Assignment 1

Basic Requirements:

- OpenGL code should use the programmable pipeline with OpenGL 3.0+ instead of the fixed pipeline



```
const GLfloat triangle_verts[] =
{
    +0.0f, +1.0f, +0.0f, //top
    +1.0f, +0.0f, +0.0f, //color

    -1.0f, -1.0f, +0.0f, //left
    +1.0f, +0.0f, +0.0f,

    +1.0f, -1.0f, +0.0f, //right
    +1.0f, +0.0f, +0.0f,
};
```

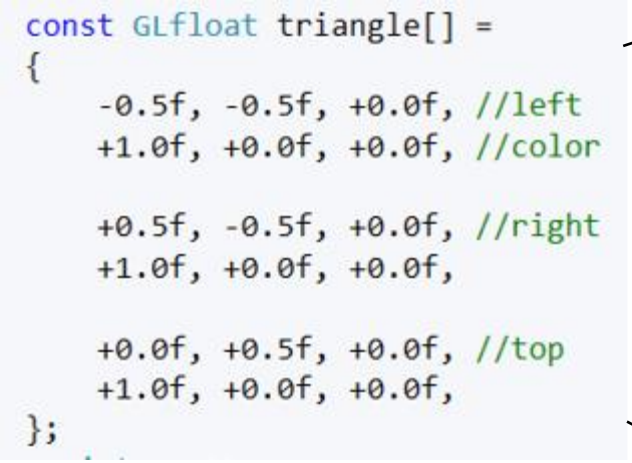


```
GLuint vaoID;
glGenVertexArrays(1, &vaoID);
glBindVertexArray(vaoID);
GLuint vboID;
glGenBuffers(1, &vboID);
glBindBuffer(GL_ARRAY_BUFFER, vboID);
glBufferData(GL_ARRAY_BUFFER, sizeof(triangle_verts),
    triangle_verts, GL_STATIC_DRAW);
//vertex position
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), 0);
//vertex color
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
    (char*)(3 * sizeof(float)));
```

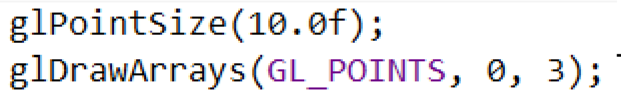
use VAOs and VBOs!

Basic Requirements:

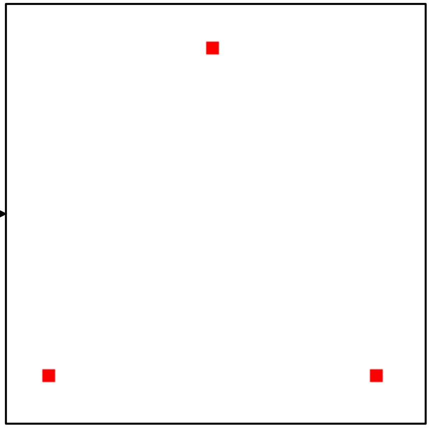
- Draw at least one 2D object and two 3D objects

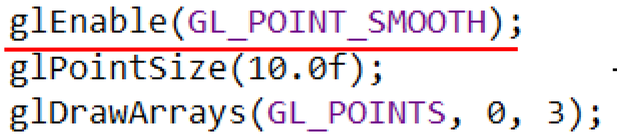


```
const GLfloat triangle[] =
{
    -0.5f, -0.5f, +0.0f, //left
    +1.0f, +0.0f, +0.0f, //color

    +0.5f, -0.5f, +0.0f, //right
    +1.0f, +0.0f, +0.0f,

    +0.0f, +0.5f, +0.0f, //top
    +1.0f, +0.0f, +0.0f,
};
```

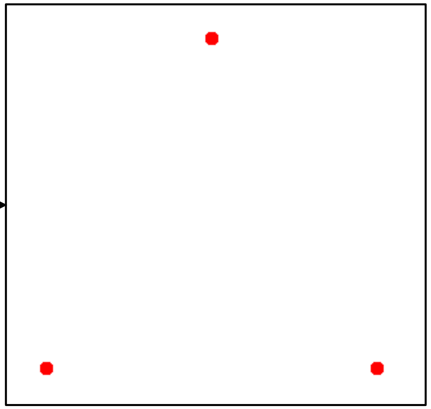


```
glPointSize(10.0f);
glDrawArrays(GL_POINTS, 0, 3);
```





```
glEnable(GL_POINT_SMOOTH);
glPointSize(10.0f);
glDrawArrays(GL_POINTS, 0, 3);
```



Assignment 1

Basic Requirements:

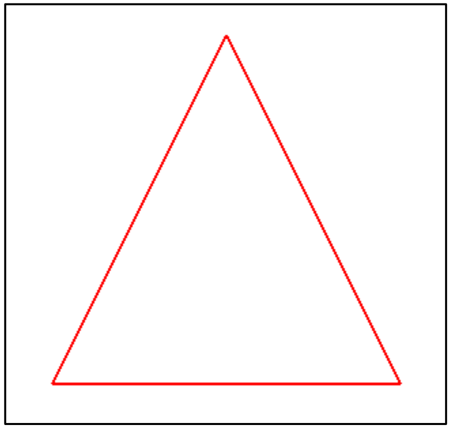
- Draw at least one 2D object and two 3D objects

```
         const GLfloat triangle[] =
         {
              -0.5f, -0.5f, +0.0f, //left
              +1.0f, +0.0f, +0.0f, //color
one line
              +0.5f, -0.5f, +0.0f, //right
              +1.0f, +0.0f, +0.0f,

              +0.5f, -0.5f, +0.0f, //right
              +1.0f, +0.0f, +0.0f,
one line
              +0.0f, +0.5f, +0.0f, //top
              +1.0f, +0.0f, +0.0f,

              +0.0f, +0.5f, +0.0f, //top
              +1.0f, +0.0f, +0.0f,
one line
              -0.5f, -0.5f, +0.0f, //left
              +1.0f, +0.0f, +0.0f, //color
         };
```

→

```
glLineWidth(1.5f);
glDrawArrays(GL_LINES, 0, 6);
```

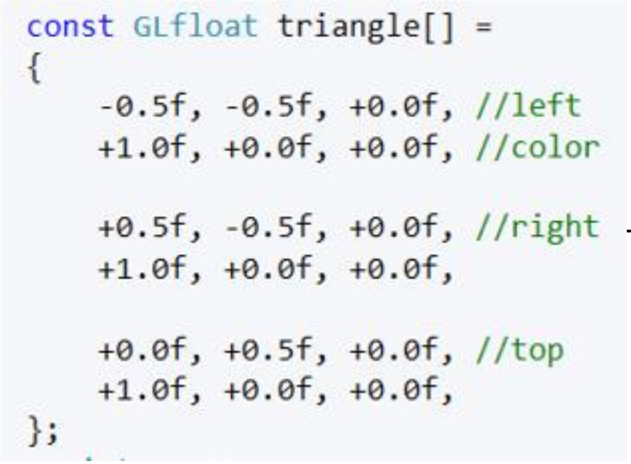
→



Assignment 1

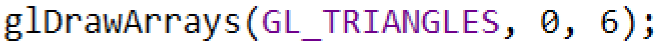
Basic Requirements:

- Draw at least <u>one</u> 2D object and <u>two</u> 3D objects



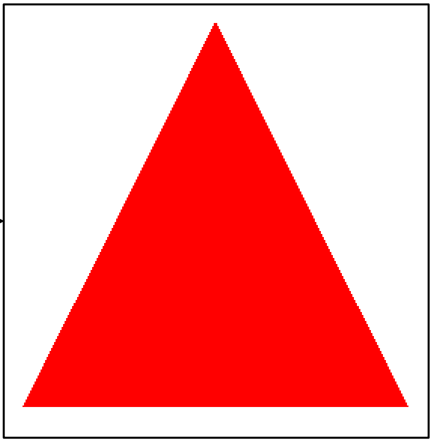
```
const GLfloat triangle[] =
{
    -0.5f, -0.5f, +0.0f, //left
    +1.0f, +0.0f, +0.0f, //color

    +0.5f, -0.5f, +0.0f, //right
    +1.0f, +0.0f, +0.0f,

    +0.0f, +0.5f, +0.0f, //top
    +1.0f, +0.0f, +0.0f,
};
```

→



```
glDrawArrays(GL_TRIANGLES, 0, 6);
```

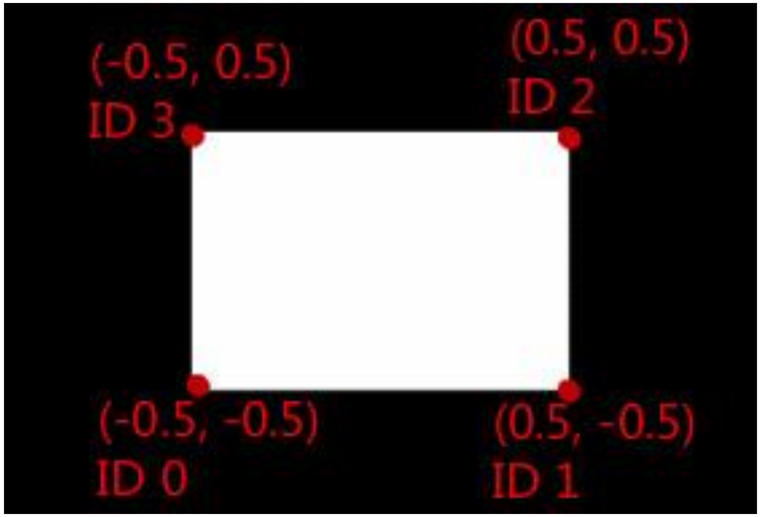
→

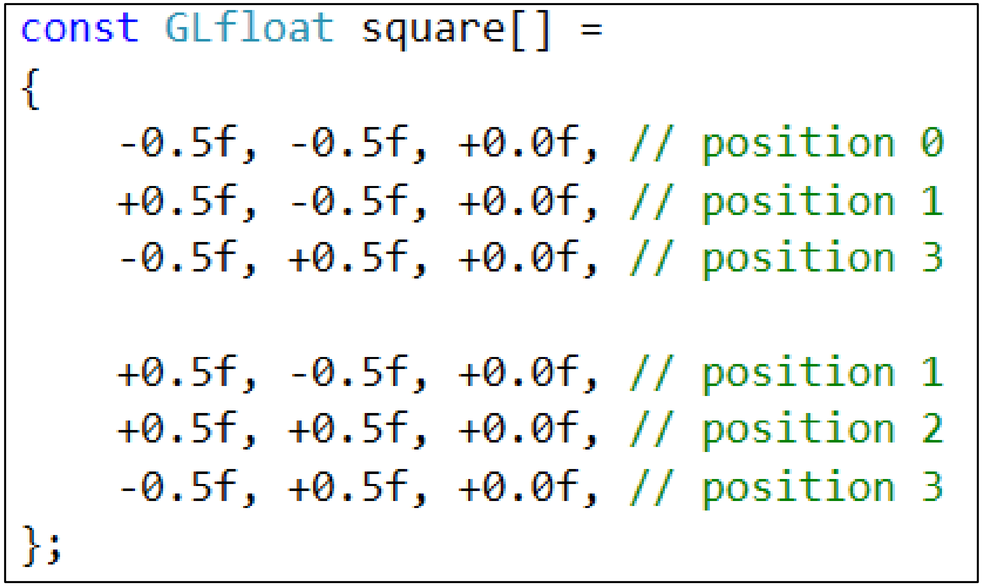


Assignment 1

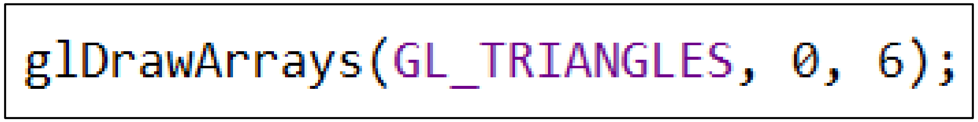
Basic Requirements:

- Ensure at least one object is drawn with indexing;





```
const GLfloat square[] =
{
    -0.5f, -0.5f, +0.0f, // position 0
    +0.5f, -0.5f, +0.0f, // position 1
    -0.5f, +0.5f, +0.0f, // position 3

    +0.5f, -0.5f, +0.0f, // position 1
    +0.5f, +0.5f, +0.0f, // position 2
    -0.5f, +0.5f, +0.0f, // position 3
};
```



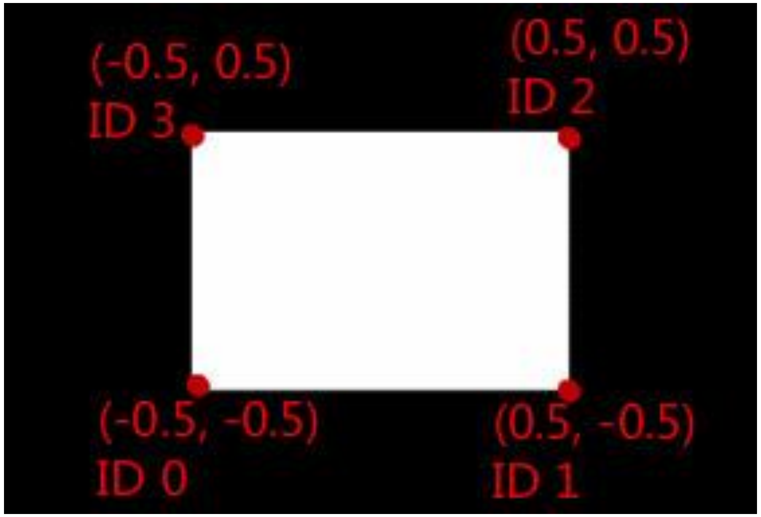
```
glDrawArrays(GL_TRIANGLES, 0, 6);
```

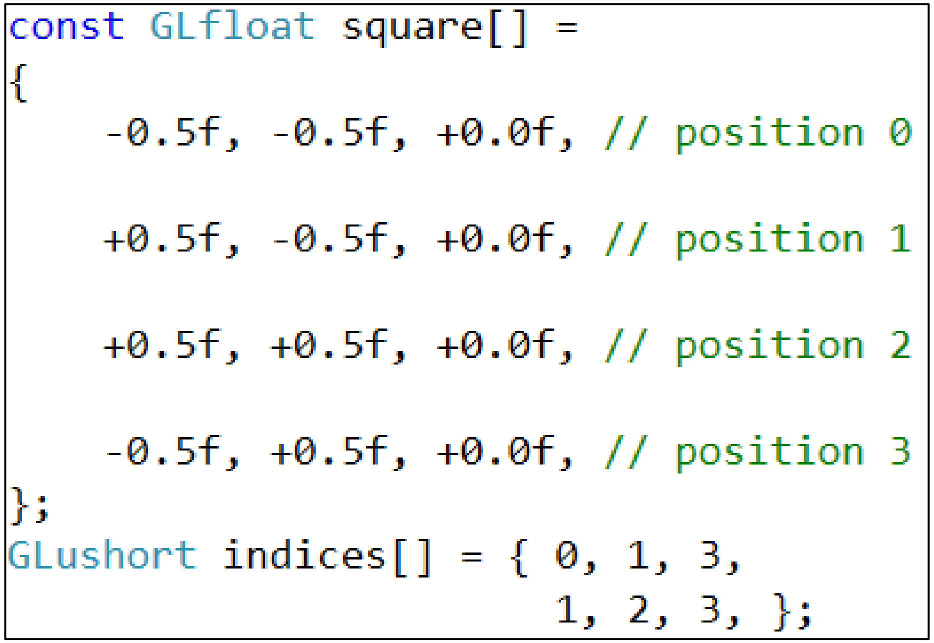
without indexing

Assignment 1

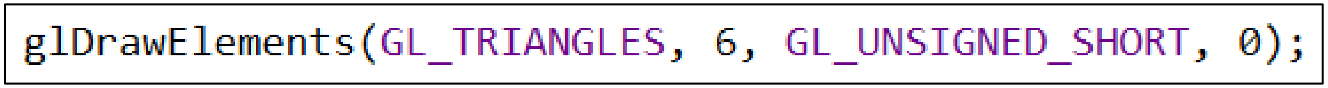
Basic Requirements:

- Ensure at least one object is drawn with indexing;





```
const GLfloat square[] =
{
    -0.5f, -0.5f, +0.0f, // position 0

    +0.5f, -0.5f, +0.0f, // position 1

    +0.5f, +0.5f, +0.0f, // position 2

    -0.5f, +0.5f, +0.0f, // position 3
};
GLushort indices[] = { 0, 1, 3,
                       1, 2, 3, };
```

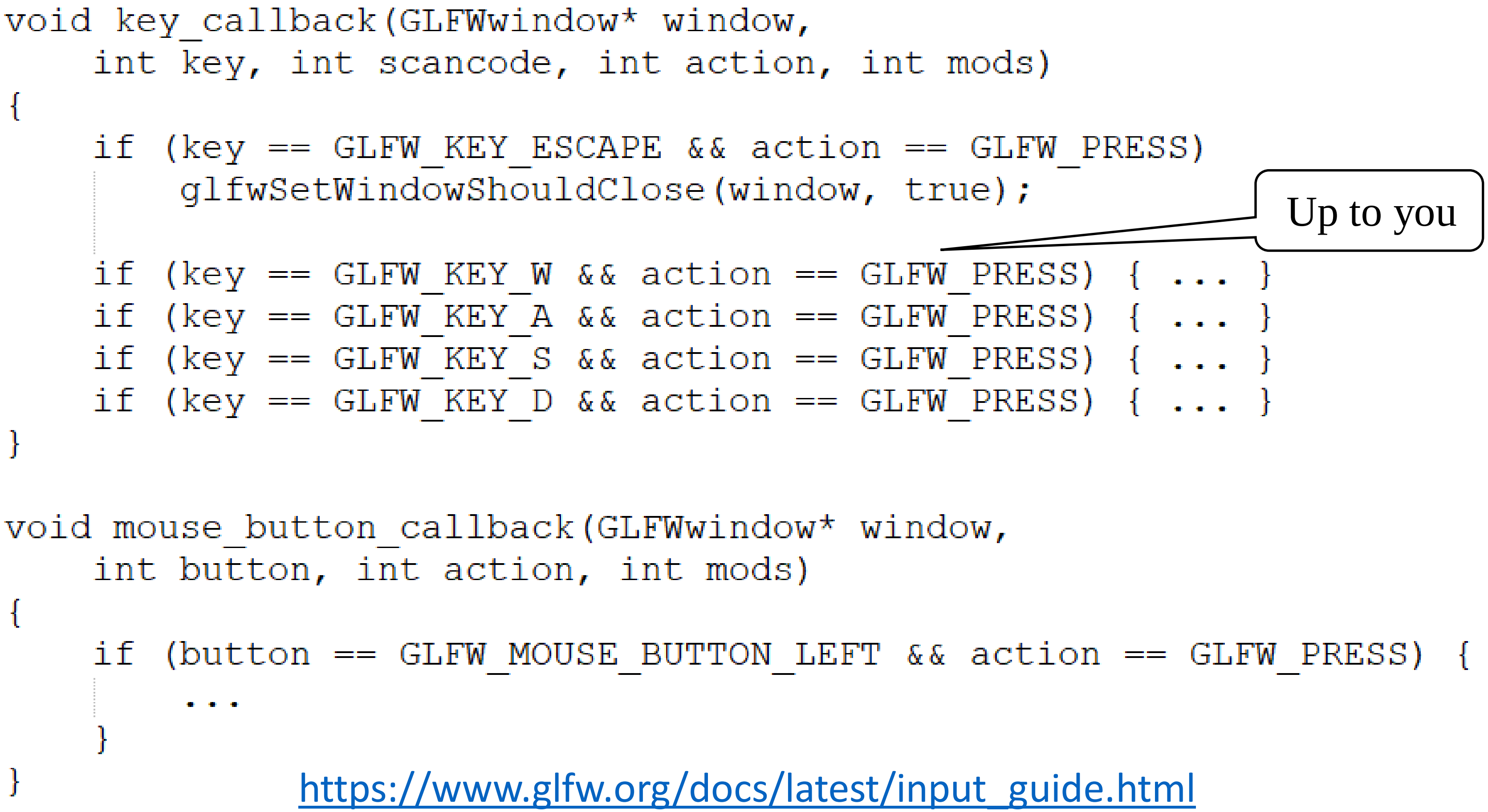


```
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_SHORT, 0);
```

with indexing

Basic Requirements:

- Create at least three kinds of keyboard and/or mouse events;



```
void key_callback(GLFWwindow* window,
    int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if (key == GLFW_KEY_W && action == GLFW_PRESS) { ... }
    if (key == GLFW_KEY_A && action == GLFW_PRESS) { ... }
    if (key == GLFW_KEY_S && action == GLFW_PRESS) { ... }
    if (key == GLFW_KEY_D && action == GLFW_PRESS) { ... }
}

void mouse_button_callback(GLFWwindow* window,
    int button, int action, int mods)
{
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS) {
        ...
    }
}
```

Up to you

https://www.glfw.org/docs/latest/input_guide.html

Basic Requirements:

- Design object transformations, including rotation, translation and scaling;

Transformation commands:

(1) Translate: glm::translate(mat4(1.0f), vec3(dx, dy, dz));

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(2) Scale: glm::scale(mat4(1.0f), vec3(x, y, z));

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Assignment 1

Basic Requirements:

- Design diverse objects transformations, such as rotation, translation and scaling;

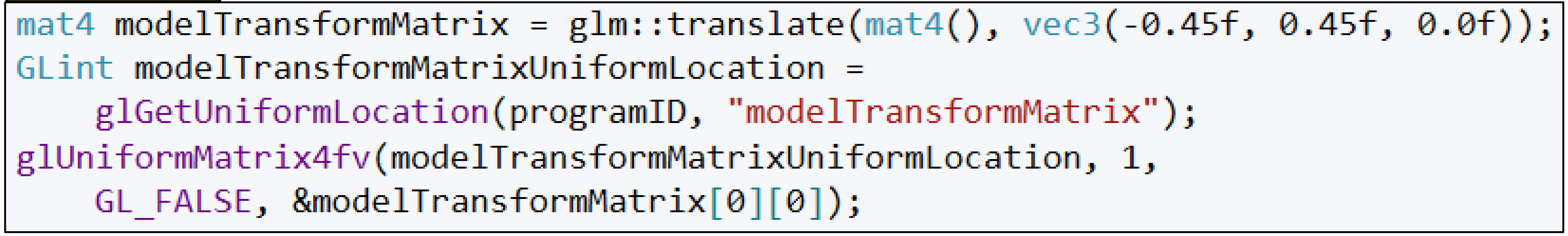
(3) Rotate around X-axis: glm::rotate(mat4(1.0f), θ, vec3(1, 0, 0)); $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

(4) Rotate around Y-axis: glm::rotate(mat4(1.0f), θ, vec3(0, 1, 0)); $\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

(5) Rotate around Z-axis: glm::rotate(mat4(1.0f), θ, vec3(0, 0, 1)); $\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

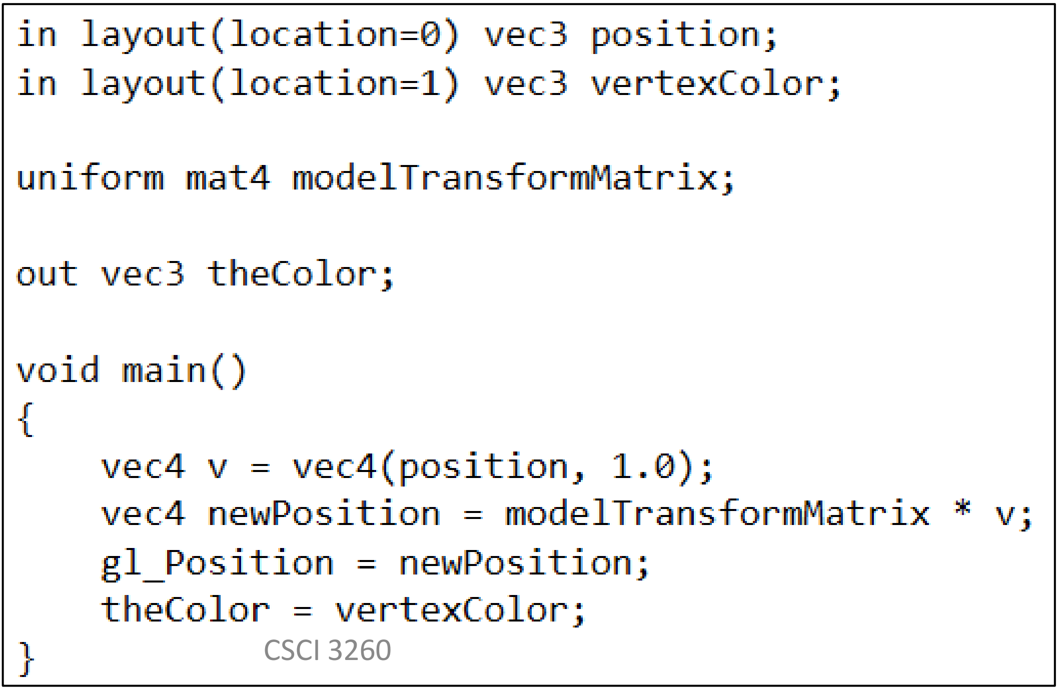
Code example: translate your object

main.cpp



```
mat4 modelTransformMatrix = glm::translate(mat4(), vec3(-0.45f, 0.45f, 0.0f));
GLint modelTransformMatrixUniformLocation =
    glGetUniformLocation(programID, "modelTransformMatrix");
glUniformMatrix4fv(modelTransformMatrixUniformLocation, 1,
    GL_FALSE, &modelTransformMatrix[0][0]);
```

VertexShaderCode.glsl

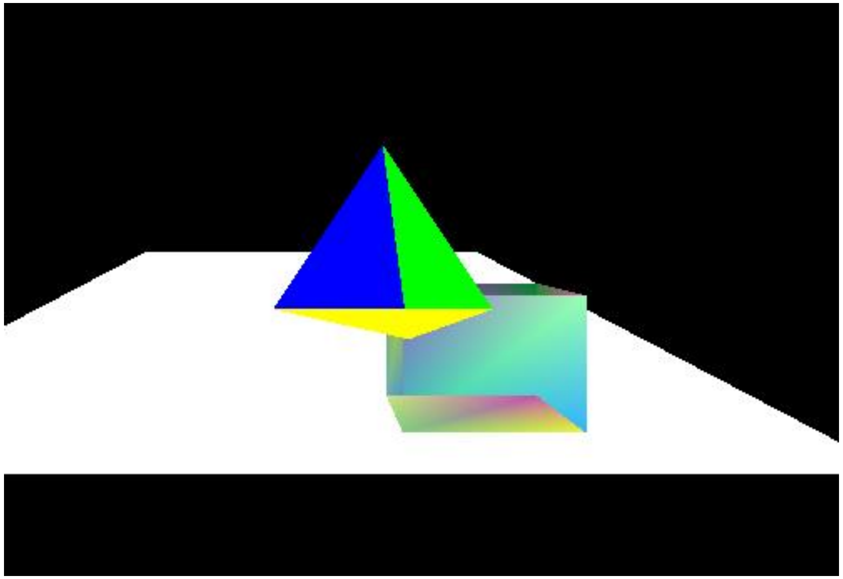


```
in layout(location=0) vec3 position;
in layout(location=1) vec3 vertexColor;

uniform mat4 modelTransformMatrix;

out vec3 theColor;

void main()
{
    vec4 v = vec4(position, 1.0);
    vec4 newPosition = modelTransformMatrix * v;
    gl_Position = newPosition;
    theColor = vertexColor;
}
```

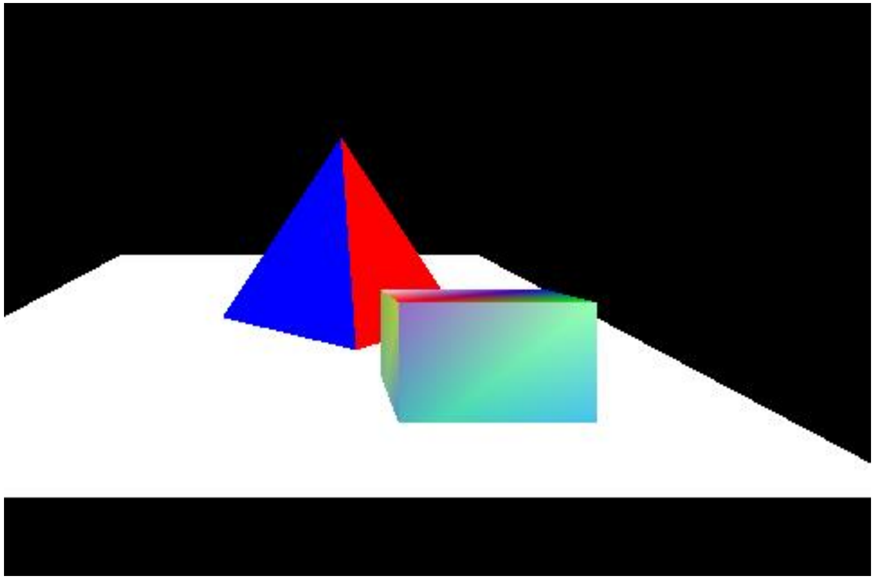
Assignment 1

Basic Requirements:

- Enable depth test to realize occlusion.



depth test disabled

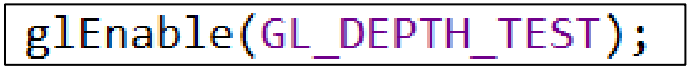


depth test enabled

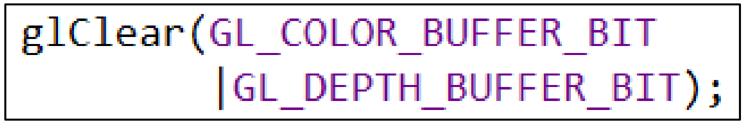
Assignment 1

Basic Requirements:

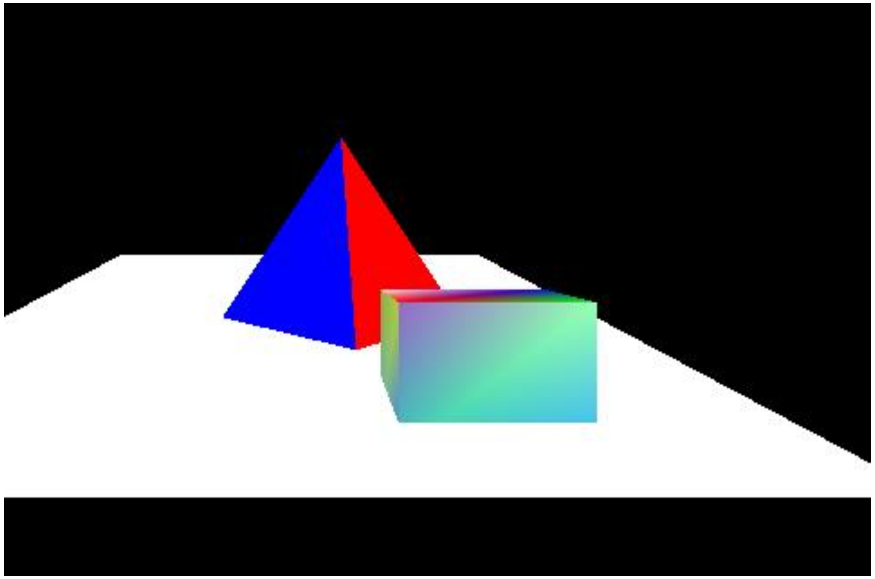
- Enable <u>depth test</u> to realize occlusion.



```
glEnable(GL_DEPTH_TEST);
```



```
glClear(GL_COLOR_BUFFER_BIT
           |GL_DEPTH_BUFFER_BIT);
```



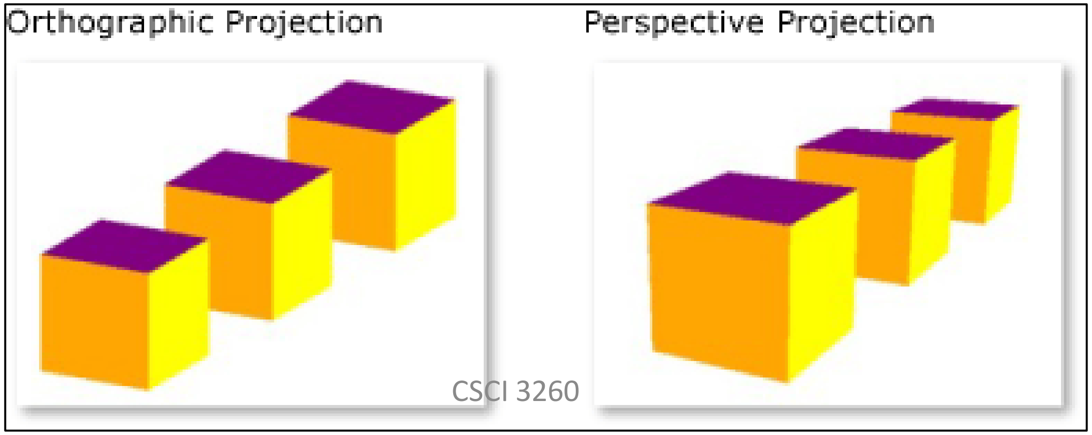
with depth test

Basic Requirements:

- Use perspective projection to draw the scene.

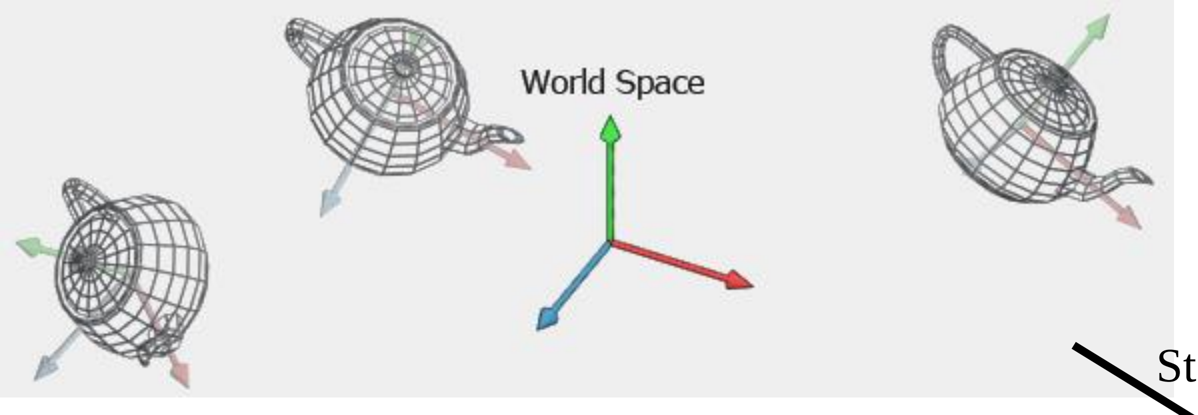
Projection: 3D scene ⇒ 2D picture

Projection methods
- Perspective projection
  - Closest things seems bigger
  - Has Vanish-Point
  - Parallel lines touch at infinity
- Orthographic projection
  - Everything seems equal
  - No Vanish-Point
  - Parallel lines never touch

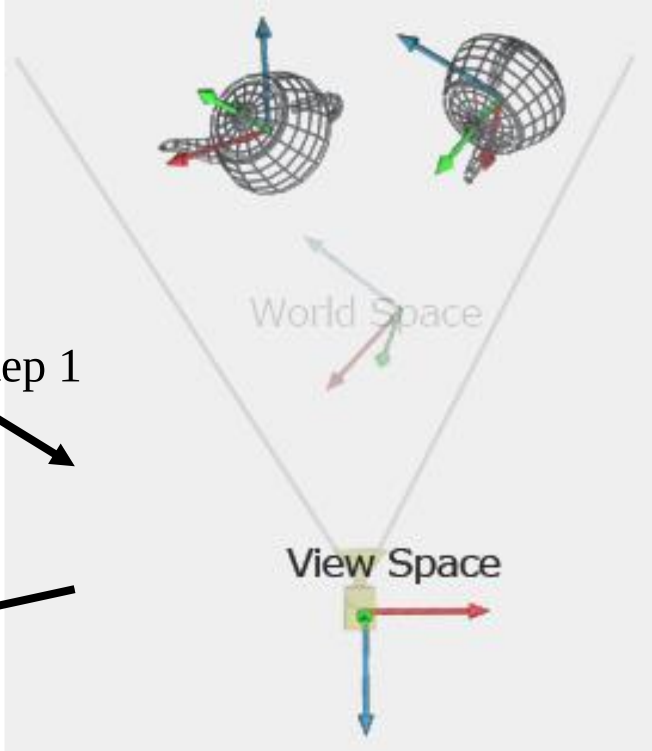


Render 3D objects

Steps to project 3D objects on the screen:

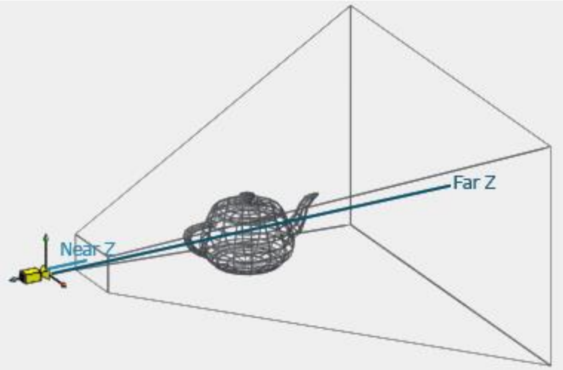


Step 1



Step 1: ModelView Matrix * ObjMatrix

Step 2

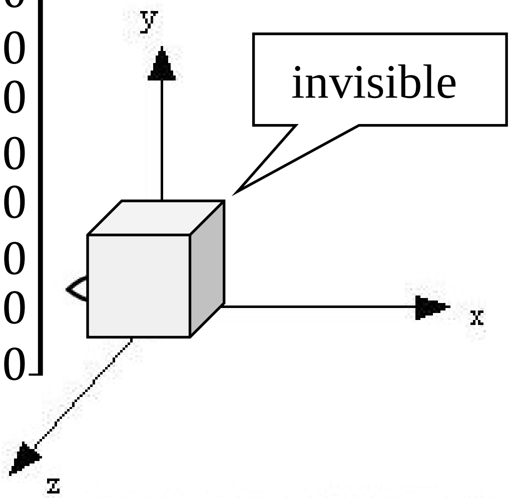


Step 2: ProjectionMatrix * ModelView Matrix * ObjMatrix

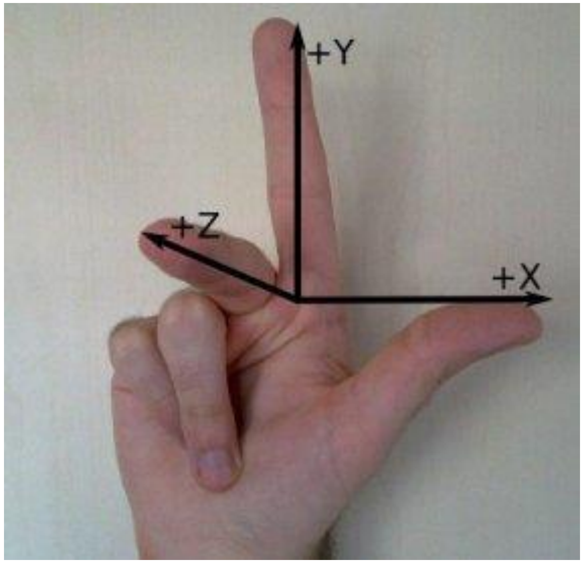
Render 3D objects

3D Coordinate System:

$$\begin{bmatrix} +1.0 & +1.0 & +1.0 \\ -1.0 & +1.0 & +1.0 \\ -1.0 & +1.0 & -1.0 \\ +1.0 & +1.0 & -1.0 \\ +1.0 & -1.0 & +1.0 \\ -1.0 & -1.0 & +1.0 \\ -1.0 & -1.0 & -1.0 \\ +1.0 & -1.0 & -1.0 \end{bmatrix}$$



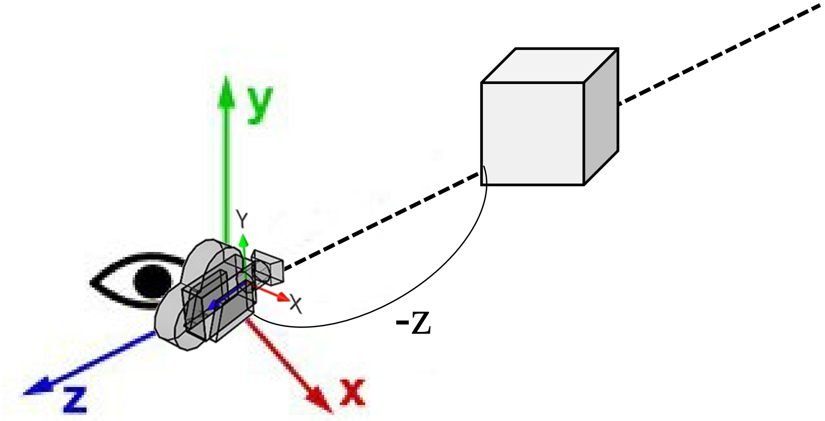
OpenGL right-handed coordinate system



If not specified, camera (eye) is placed in the original point.

1) ModelView Matrix:

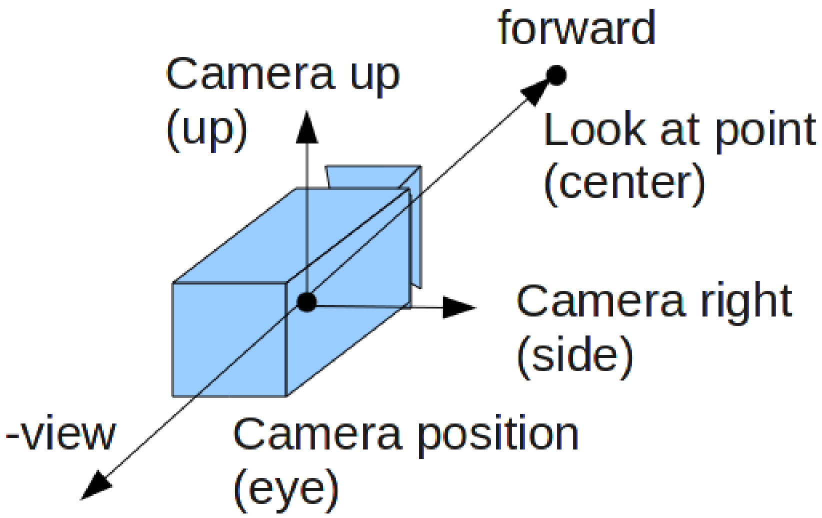
Move models to the front of the camera.



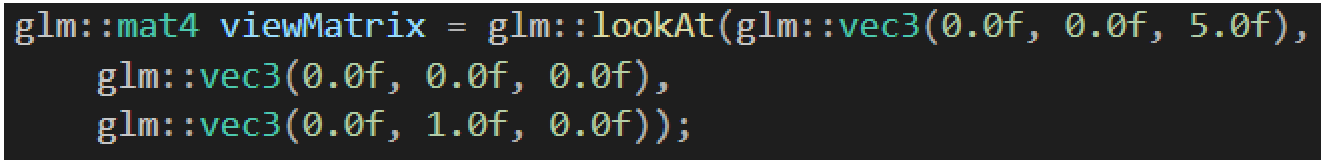
glm::translate(mat4(1.0f), vec3(0.0f, 0.0f, -z)), z is positive

1) ModelView Matrix:

You can also change the parameters of camera.



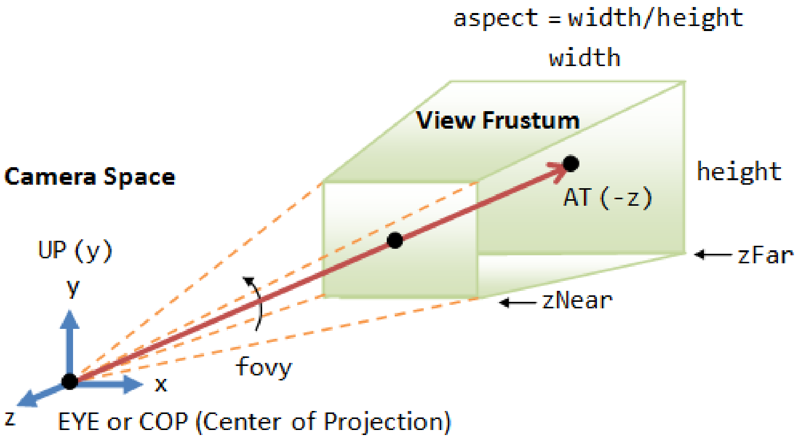
glm::lookat(eye, center, up)



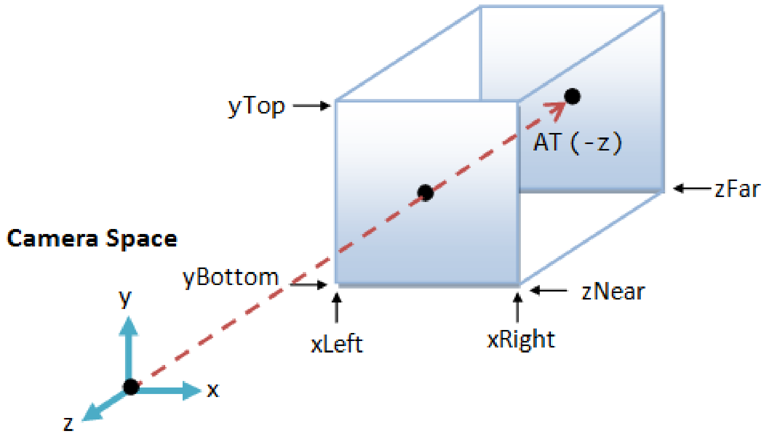
```
glm::mat4 viewMatrix = glm::lookAt(glm::vec3(0.0f, 0.0f, 5.0f),
    glm::vec3(0.0f, 0.0f, 0.0f),
    glm::vec3(0.0f, 1.0f, 0.0f));
```

2) Projection Matrix:

Converts 3D positions into 2D positions on the screen.



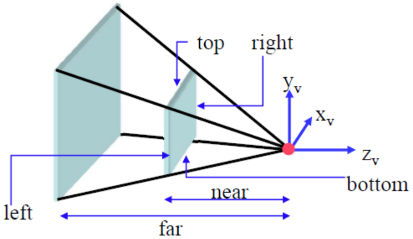
Perspective Projection: The camera's view frustum is specified via 4 view parameters: fovy, aspect, zNear and zFar.



Orthographic Projection: Camera positioned infinitely far away at $z = \infty$

glm::perspective(fovy, aspect, zNear, zFar)
Or glm::frustum(left, right, bottom, top, zNear, zFar)

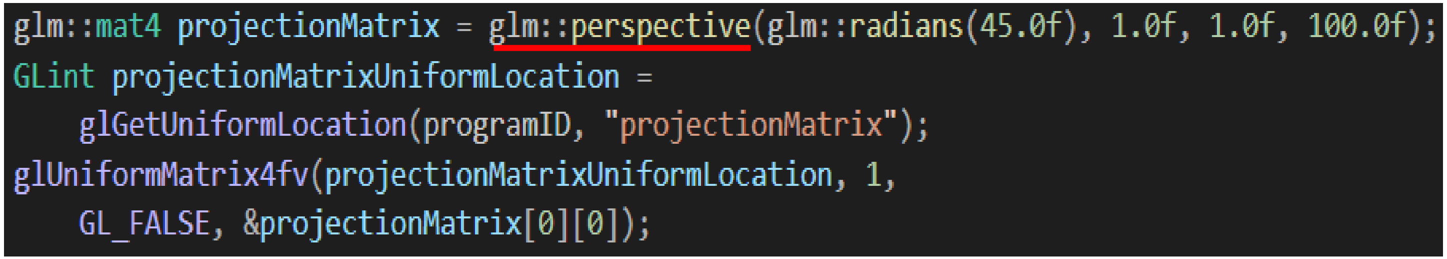
glm::ortho(left, right, bottom, top, zNear, zFar)
By default: ortho(-1, 1, -1, 1, -1, 1)



Render 3D objects

Projection codes:

main.cpp



```
glm::mat4 projectionMatrix = glm::perspective(glm::radians(45.0f), 1.0f, 1.0f, 100.0f);
GLint projectionMatrixUniformLocation =
    glGetUniformLocation(programID, "projectionMatrix");
glUniformMatrix4fv(projectionMatrixUniformLocation, 1,
    GL_FALSE, &projectionMatrix[0][0]);
```

VertexShaderCode.glsl

```
in layout(location = 0) vec3 position;
in layout(location = 1) vec3 vertexColor;

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;

out vec3 theColor;

void main()
{
    vec4 v = vec4(position, 1.0);
    vec4 out_position = projectionMatrix * viewMatrix * modelMatrix * v;
    gl_Position = out_position;
    theColor = vertexColor;
}
```