ICG 2024

HW1

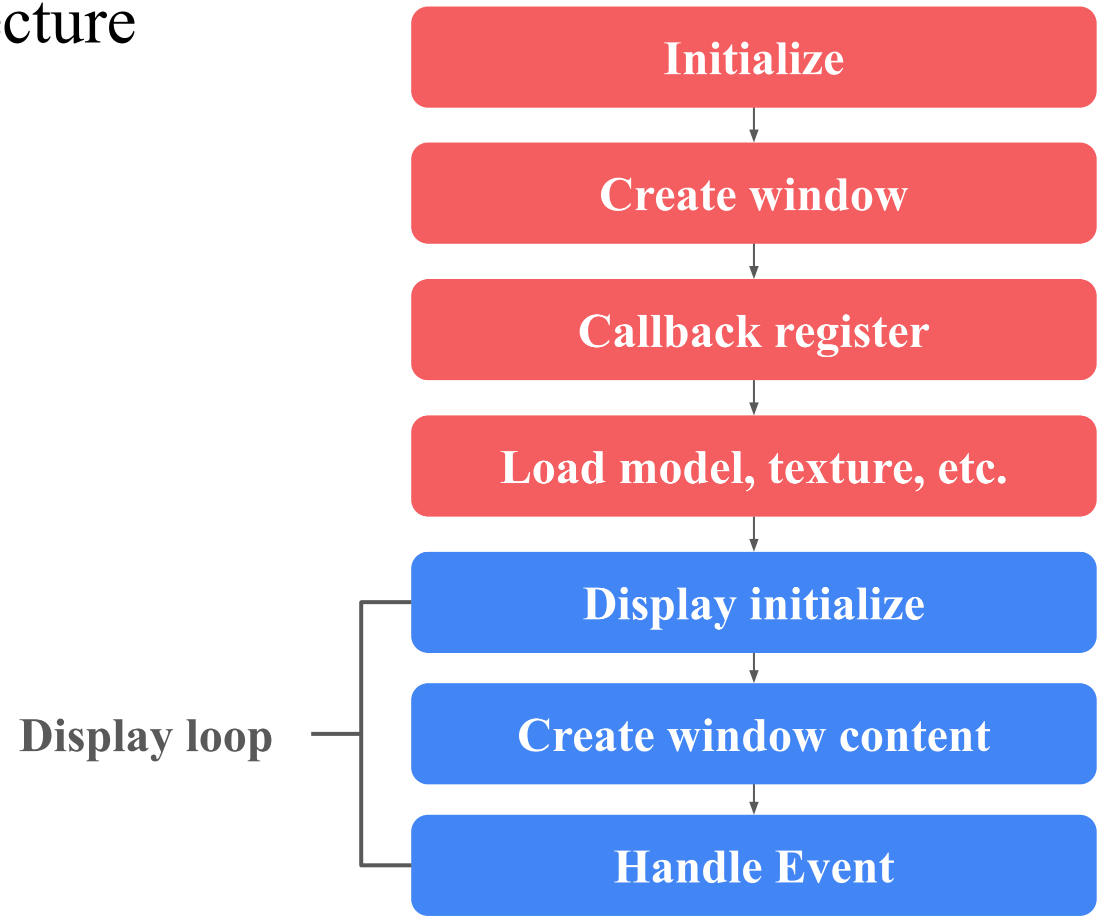
2024/10/1

IDE & Kit

We have already posted the installation guide on E3: “HW0"

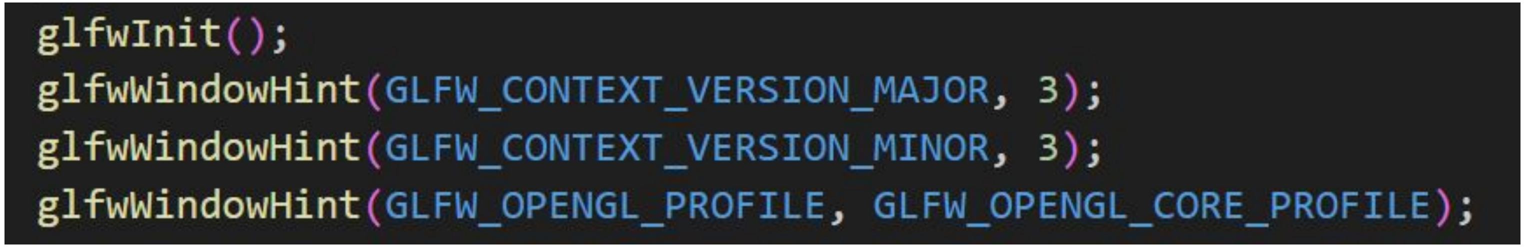
- Visual Stdio Code
- C++
- CMake
- CMake Tools extension
- GLFW (provided in zip)
  - An Open Source, multi-platform for OpenGL
  - Provide a simple API for creating windows, receiving input and, etc.
- GLAD (provided in zip)
  - An OpenGL loading library that loads pointers to OpenGL functions at runtime
- GLM (provided in zip)
  - Math library for OpenGL

Architecture



Initialize & Window$_{1/4}$

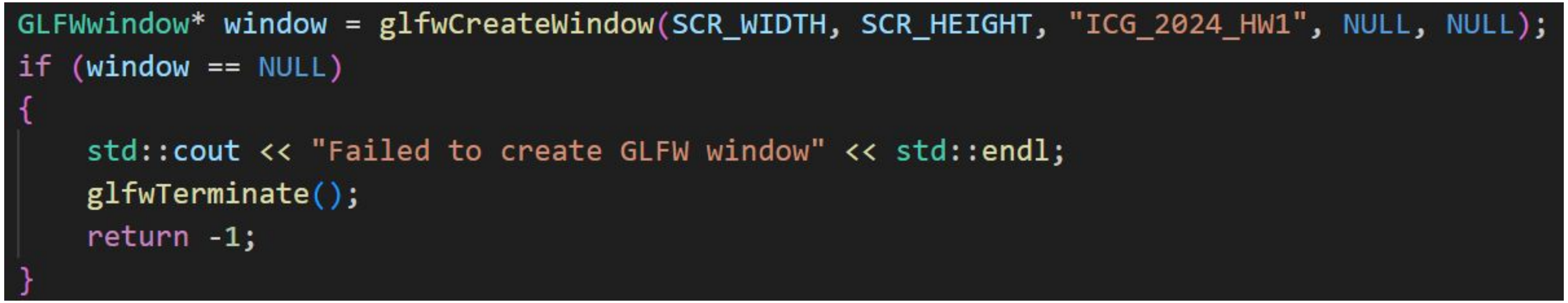
- int glfwInit()
  - Initialize GLFW
  - Return GLFW_TRUE when successful, else GLFW_FALSE
- void glfwWindowHint(int hint, int value)
  - Window settings for the next window creation
  - In this homework, we will use OpenGL 3.3 core profile



```
glfwInit();
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
```

Initialize & Window$_{2/4}$

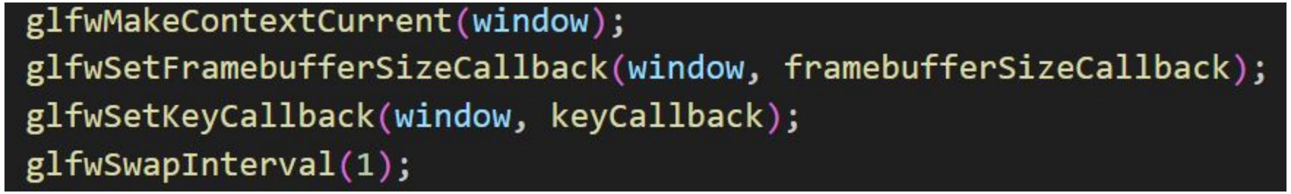
- ❖ GLFWwindow* glfwCreateWindow(int width, int height, const char* title, GLFWmonitor* monitor, GLFWmonitor* share)
  - ➢ Create a window with the specified width, height, and title
  - ➢ monitor: monitor is used for full-screen mode, NULL for window mode
  - ➢ share: the window to share resources with, NULL to not share resources
  - ➢ Return the handle of the created window, or NULL if an error occurred



```
GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "ICG_2024_HW1", NULL, NULL);
if (window == NULL)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();
    return -1;
}
```

Initialize & Window$_{3/4}$

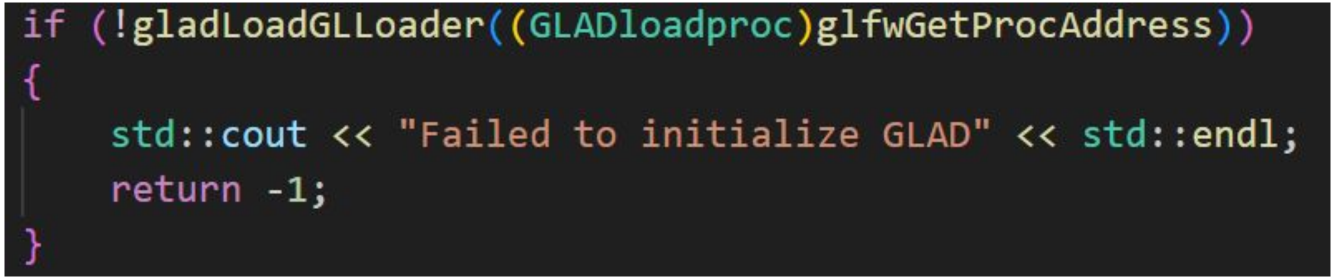
- void glfwMakeContextCurrent(GLFWwindow* window)
  - Make context current for the calling thread
- GLFWframebuffersize glfwSetFramebufferSizeCallback(GLFWwindow* window, GLFWframebuffersizefun cbfun)
  - Register a callback function for window resize
- GLFWkeyfun glfwSetKeyCallback(GLFWwindow* window, GLFWkeyfun cbfun)
  - Register a callback function for key events
- void glfwSwapInterval(int interval)
  - Set the number of screen updates to wait before swapping buffers and returning after calling glfwSwapBuffers()



```
glfwMakeContextCurrent(window);
glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);
glfwSetKeyCallback(window, keyCallback);
glfwSwapInterval(1);
```

Initialize & Window$_{4/4}$

- int gladLoadGLLoader((gladloadproc)glfwGetProcAddress))
  - Initialize GLAD to get the OpenGL function pointer

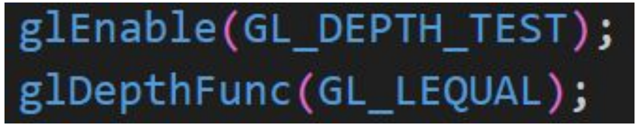


```
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    std::cout << "Failed to initialize GLAD" << std::endl;
    return -1;
}
```

Depth test

To prevent occluded faces from being rendered, we need to enable depth testing

- void glEnable(GL_DEPTH_TEST)
  - While depth test is enabled, OpenGL tests the depth value of each fragment against the content in the depth buffer. If the test passes, the fragment is rendered. If not, the fragment is discarded
- void glDepthFunc(GLenum func)
  - Specify how the test is performed
  - func: GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GRATER, GL_GEQUAL, GL_NOTEQUAL, GL_ALWAYS
  - GL_LEQUAL: test passes If the fragment depth ≤ the depth stored in the buffer

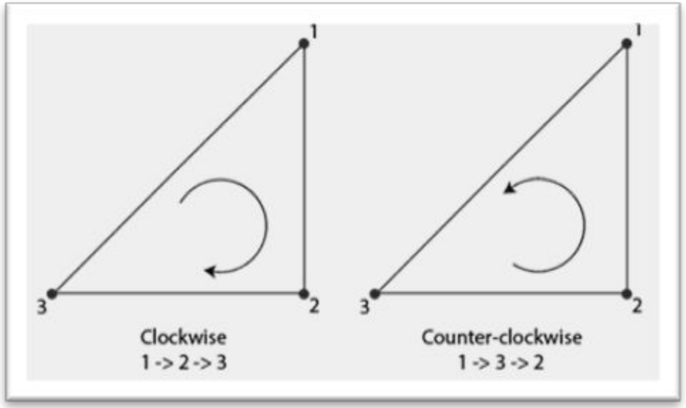


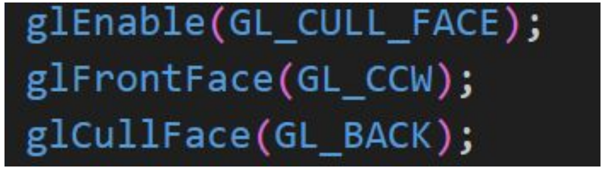
```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
```

Face culling

Face culling reduces the number of faces rendered by discarding faces that are not visible

- void glEnable(GL_CULL_FACE)
  - Tell OpenGL to enable face culling
- void glFrontFace(GLenum mode)
  - mode:GL_CW, GL_CCW
  - Faces with specified ordered vertices are defined as front
- void glCullFace(GLenum mode)
  - mode: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - Cull specified faces



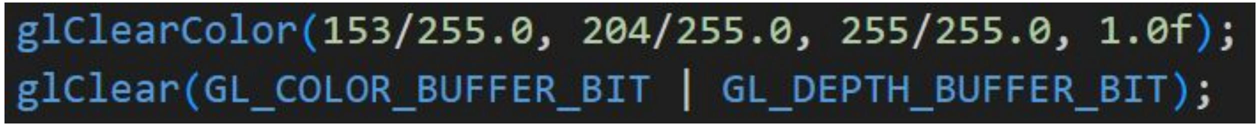


```
glEnable(GL_CULL_FACE);
glFrontFace(GL_CCW);
glCullFace(GL_BACK);
```

Display loop

Before we start to draw, we need to clear the color buffer and the depth buffer

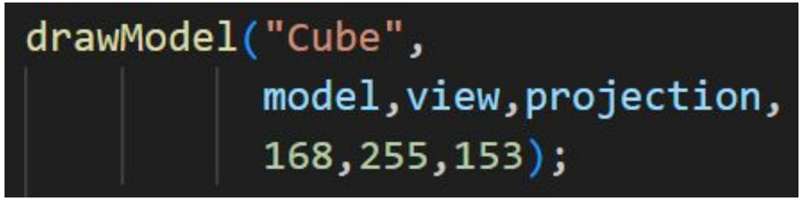
- void glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)
  - Set the color value that is used to reset the color buffer
- void glClear(GLbitfield mask)
  - Clear the specified buffer
  - mask:
    - GL_COLOR_BUFFER_BIT: clear color buffer
    - GL_DEPTH_BUFFER_BIT: clear depth buffer



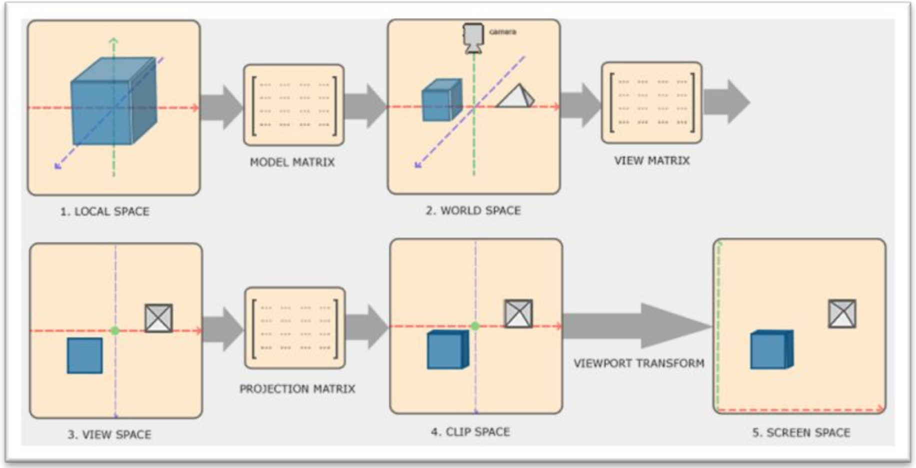
```
glClearColor(153/255.0, 204/255.0, 255/255.0, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Draw a model$_{1/2}$

- void drawModel(const string& **name**, const glm::mat4& **model**,
  const glm::mat4& **view**, const glm::mat4& **projection**, int **r**, int **g**, int **b**)
  - **Draw the target model**
  - name: the target model (“Cube”, “I”, “C”, "G", "Tree_down" , "Tree_up")
  - model / view/ projection: model / view/ projection matrix
  - r/ g/ b: red/ green/ blue color (int 0~255)



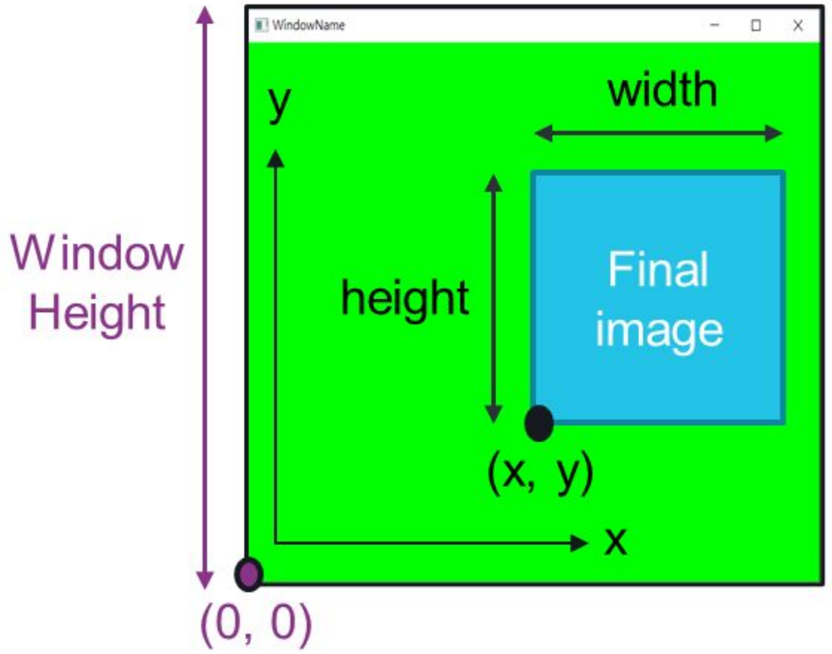
```
drawModel("Cube",
            model,view,projection,
            168,255,153);
```



Draw a model$_{2/2}$

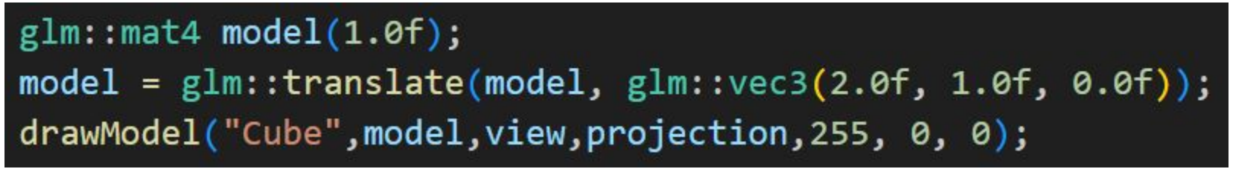
- void glViewport(GLint x,GLint y, GLint width, GLint height)
    - Specify the viewport rectangle

```
glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
```

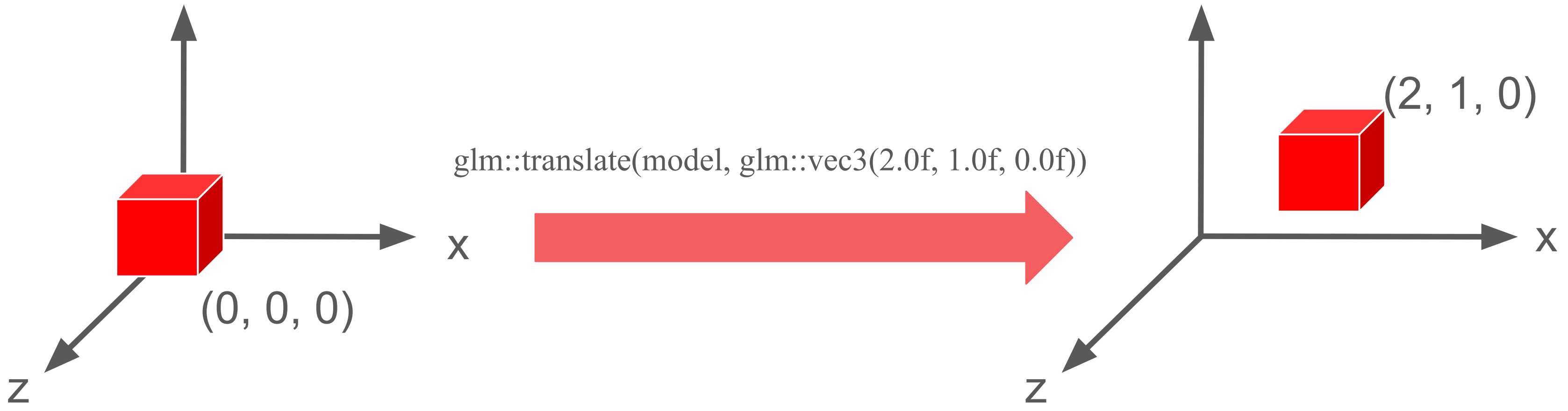


Model matrix$_{1/3}$

- glm::translate(glm::mat4 M, glm::vec3 translation)
  - Return M * (translation matrix)

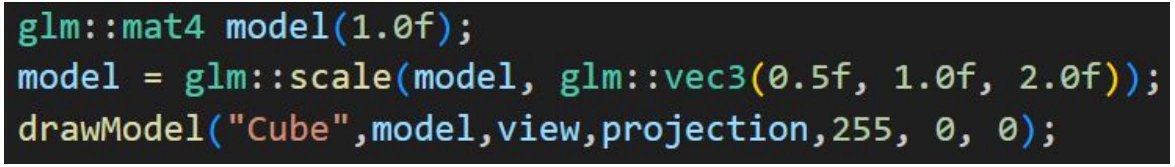


```
glm::mat4 model(1.0f);
model = glm::translate(model, glm::vec3(2.0f, 1.0f, 0.0f));
drawModel("Cube",model,view,projection,255, 0, 0);
```

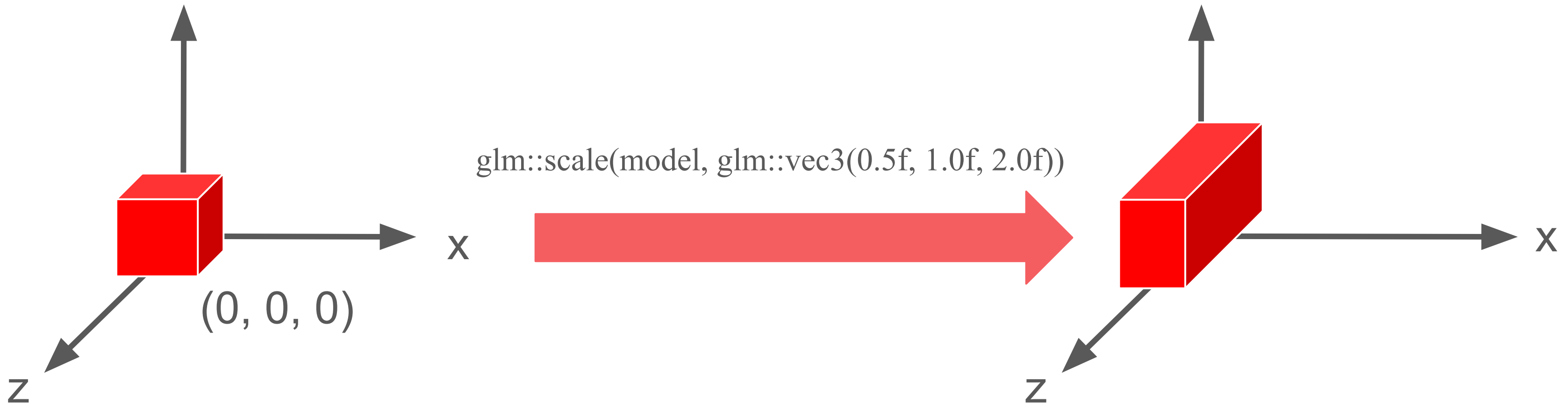


Model matrix$_{2/3}$

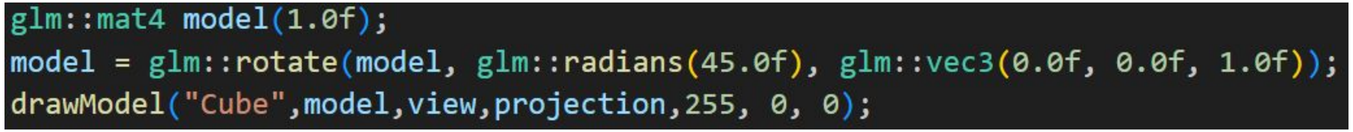
- glm::scale(glm::mat4 M, glm::vec3 scale)
  - Return M * (scale matrix)



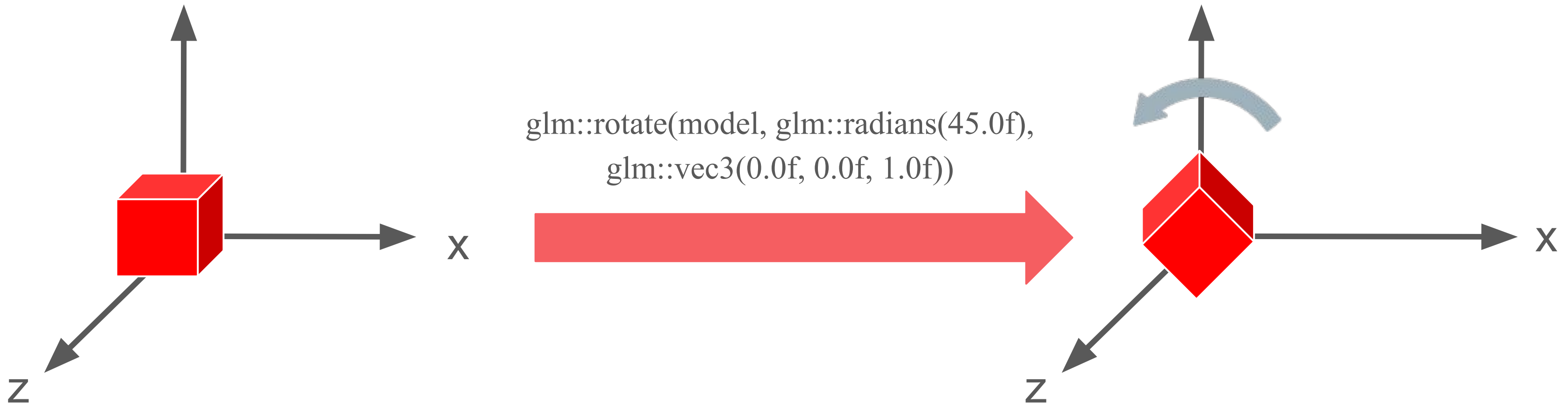
```
glm::mat4 model(1.0f);
model = glm::scale(model, glm::vec3(0.5f, 1.0f, 2.0f));
drawModel("Cube",model,view,projection,255, 0, 0);
```



Model matrix$_{3/3}$

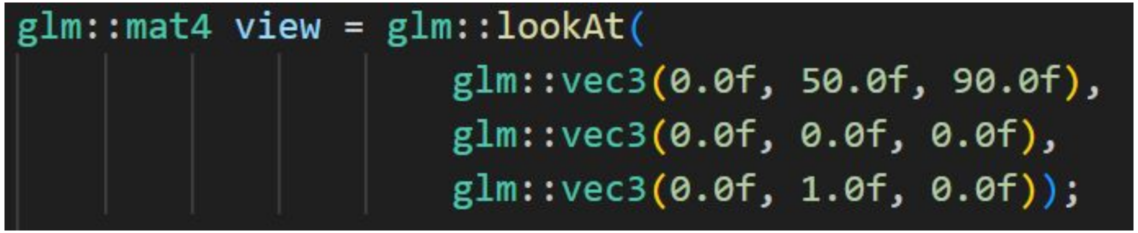

```
glm::mat4 model(1.0f);
model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));
drawModel("Cube",model,view,projection,255, 0, 0);
```

- glm::rotate(glm::mat4 M, GLfloat angle, glm::vec3 axis)
  - Return M * (rotation matrix)
  - The rotation matrix rotates an angle in radians about the given axis
- glm::radians(GLfloat degree)
  - Convert the given degree to radian

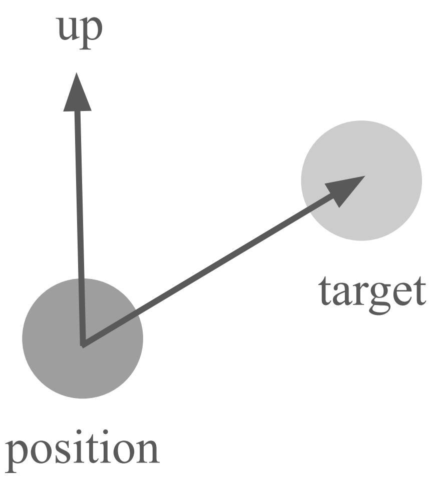

x
z
glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f))
x
z

View matrix

- glm::lookAt(glm::vec3 position, glm::vec3 target, glm::vec3 up)
  - Return view matrix with camera at position looking at the target with up vector

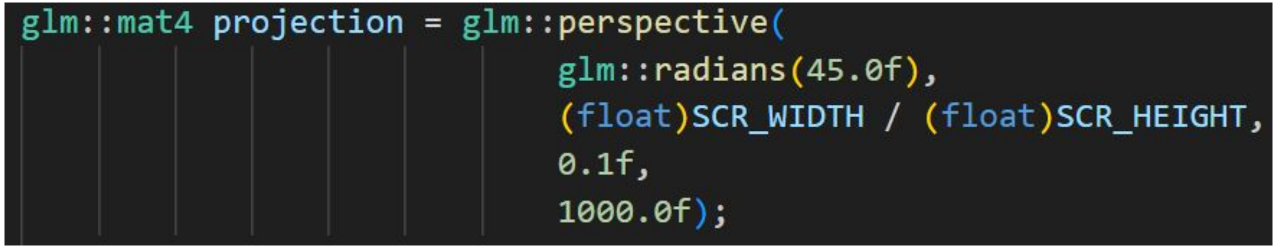


```
glm::mat4 view = glm::lookAt(
                    glm::vec3(0.0f, 50.0f, 90.0f),
                    glm::vec3(0.0f, 0.0f, 0.0f),
                    glm::vec3(0.0f, 1.0f, 0.0f));
```

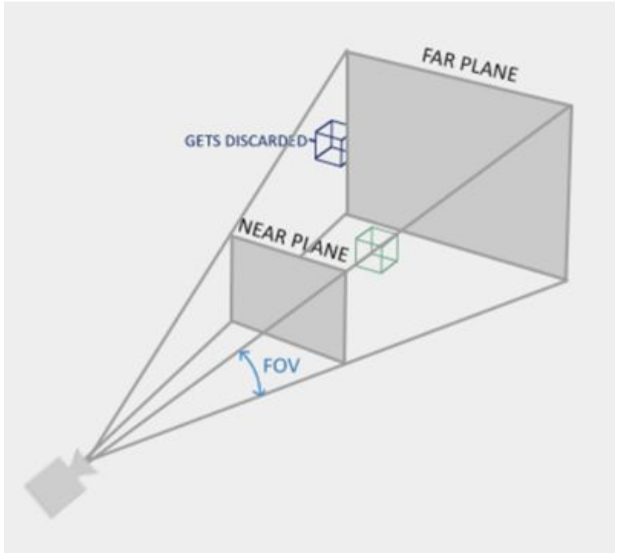


Projection matrix

- glm::perspective(GLfloat fov, GLfloat aspect, GLfloat near, GLfloat far)
  - Return the perspective projection matrix with the above parameters
  - fov: specify Field of View in radians
  - aspect: specify aspect ratio of the scene
  - near: specify near plane
  - far: specify far plane
  - Coordinates in front of near plane or behind far plane will not be drawn

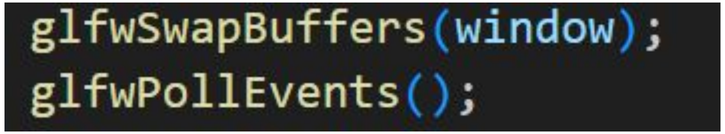


```
glm::mat4 projection = glm::perspective(
                            glm::radians(45.0f),
                            (float)SCR_WIDTH / (float)SCR_HEIGHT,
                            0.1f,
                            1000.0f);
```



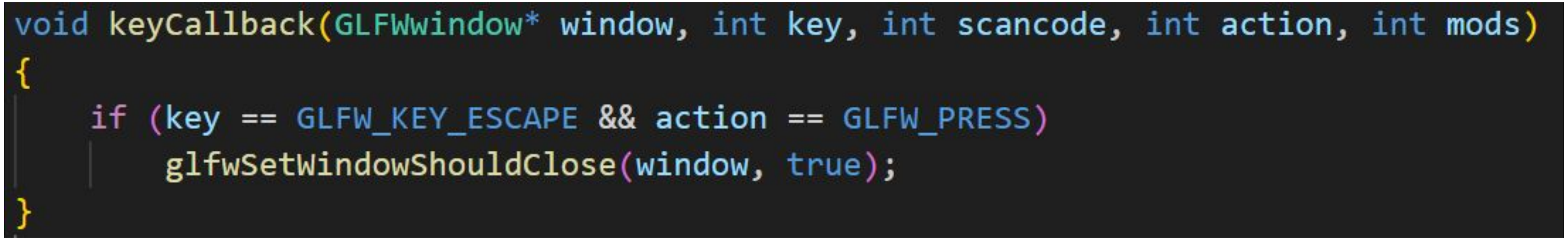
Display loop

- void glfwSwapBuffers(GLFWwindow* window)
  - Swap buffer at the end of the display loop
- void glfwPollEvent()
  - Handle any events occurring while rendering the frame



```
glfwSwapBuffers(window);
glfwPollEvents();
```

Key callback

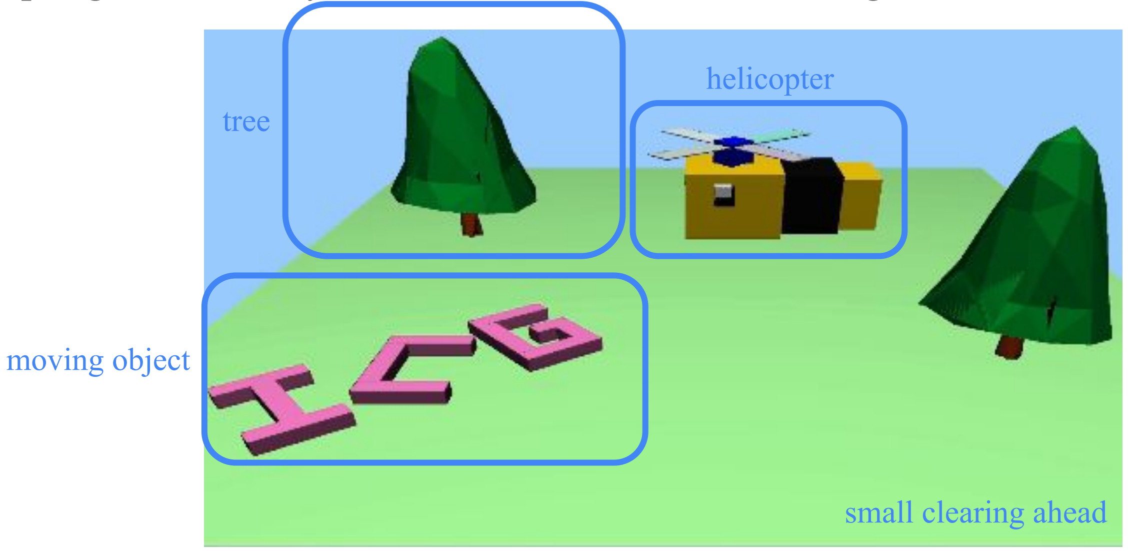

```
void keyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}
```

- The above function is registered for a key callback. We can check for key events and act correspondingly
- It sets glfwWindowShouldClose to true when the escape key is pressed, which will exit the display loop.
- The full list of keys can be found here.
- The full list of actions and their effects can be found here.

Homework 1 - Introduction

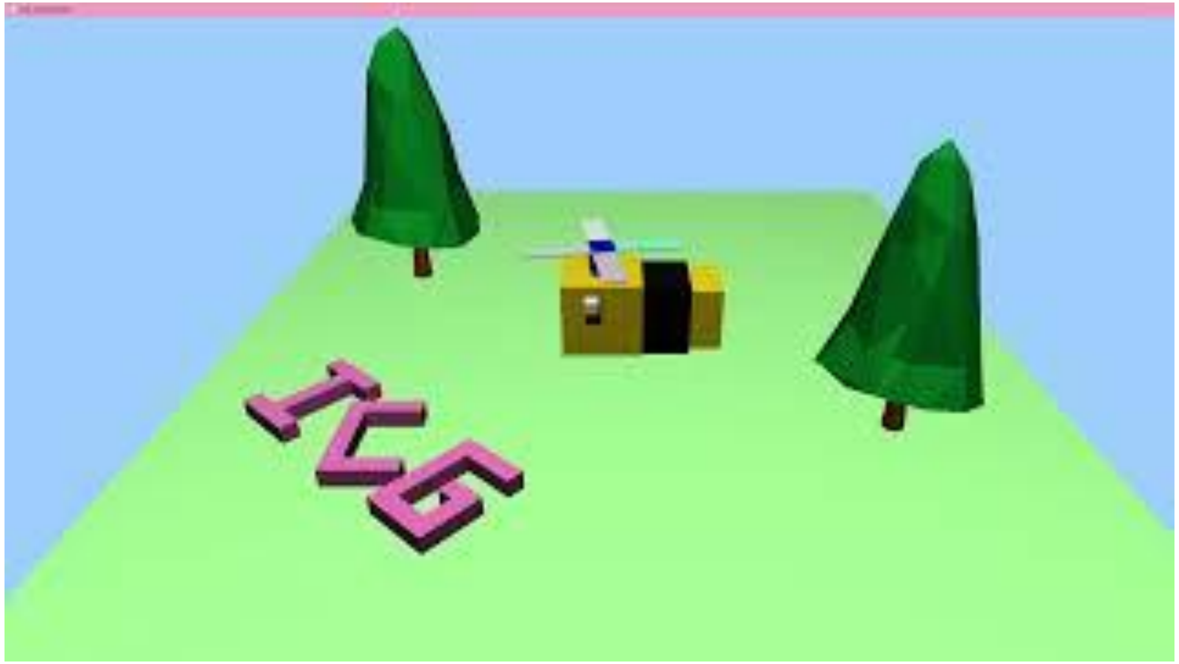
Story

Deep within the dense forest, a helicopter flew at a low altitude, the sound of its rotor blades echoing through the trees. The pilot carefully avoided the towering trees, keeping a close eye on the instruments. Suddenly, he spotted a small clearing ahead, and it seemed like something was moving on the ground, piquing his curiosity. He decided to land and investigate.



Demo

- [video]



Requirement$_{1/4}$

Camera:

Position: (0, 50, 90)

Target: (0, 0, 0)

Up: (0, 1, 0)

Fov: 45

near: 0.1

far: 1000

Ground(Cube):

Position: (0, 0, 0)

Scale: (100, 1, 120)

Tree1(Tree_up and Tree_btn):

Position: (-30, 0, -20) relative to the Ground

Scale: (4, 4, 4)

Tree2(Tree_up and Tree_btn):

Position: (30, 0, 20) relative to the Ground

Scale: (3.5, 3.5, 3.5)

C(C):

Position: (-25, 2, 25)

I(I):

Position: (-12, 0, 0) relative to the C

Rotate: 0.8 degrees/ frame about +y axis around the C

G(G):

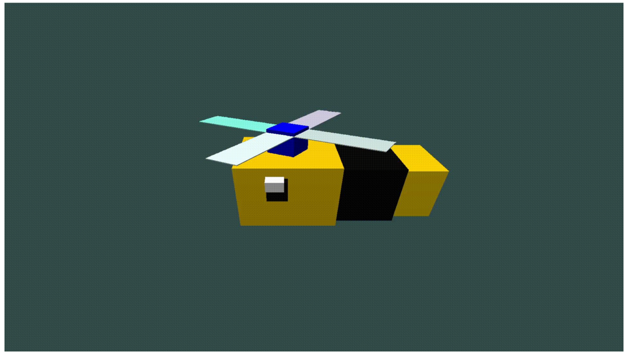
Position: (12, 0, 0) relative to the C

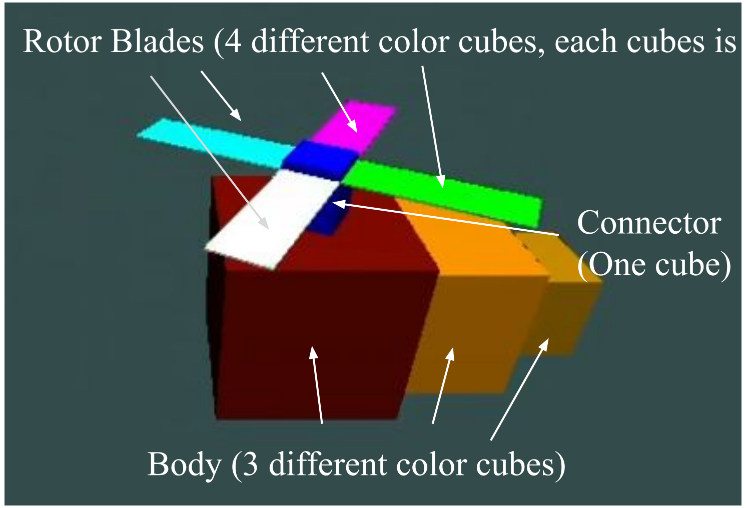
Rotate: 0.8 degrees/ frame about +y axis around the C

Requirement$_{2/4}$

Helicopter (Body + Connector + Rotor Blades) :

- Body (At least 3 different color cubes)
- Connector (One cube)
  - **On the body**
  - Rotate: 5 degrees/ frame about +y axis
- Rotor Blades (4 different color cubes)
  - **On the connector**
  - Each cubes is offset by 90 degrees
- You can be creative in design !!




Rotor Blades (4 different color cubes, each cubes is offset by 90 degrees)
Connector (One cube)
Body (3 different color cubes)

Requirement$_{3/4}$

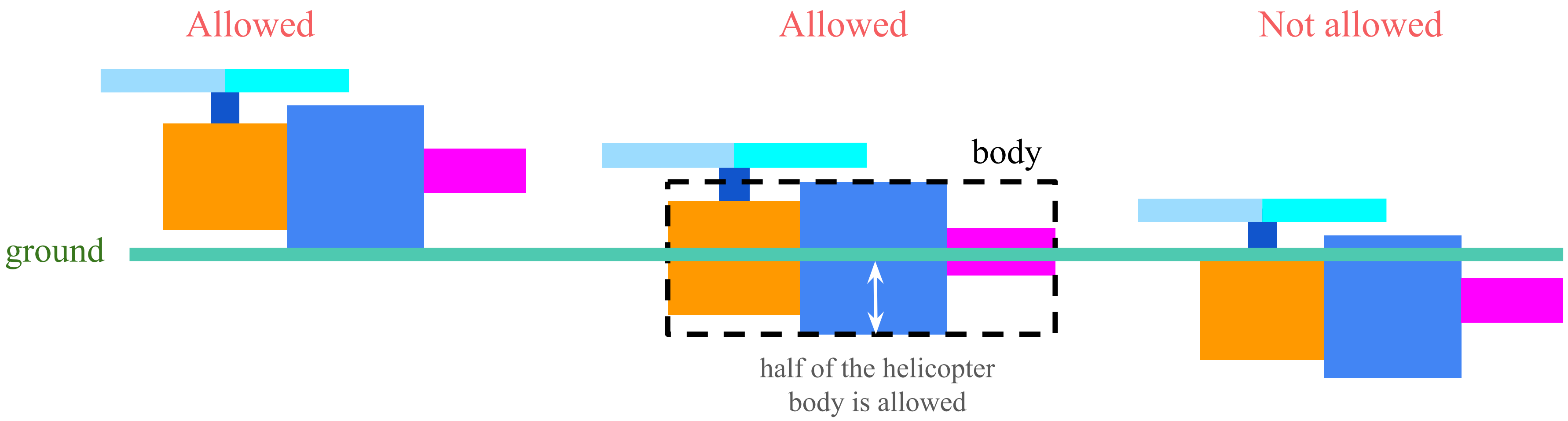
Keyboard input:

- ❖ Press W to move the helicopter in the Y direction by 1
- ❖ Press S to move the helicopter in the Y direction by -1
- ❖ Press D to move the helicopter in the X direction by 1
- ❖ Press A to move the helicopter in the X direction by -1

Requirement$_{4/4}$

Ground collision detection:

- The helicopter cannot fly below the ground.
- An error margin of less than half of the helicopter body is allowed.



Score

Depth testing (pass if or equal) - 3%

Face culling (counter-clockwise as front, cull back) - 3%

Camera and perspective - 4%

Ground and Two trees (all transformations must be correct) - 15%

ICG (all transformations must be correct) - 20%

Helicopter (all transformations must be correct) - 25%

All model are correct - 10%

Keyboard input - 10%

Ground collision detection - 10%

Homework 1 - Submission

- Deadline: 2024/10/15 23:59:59
  - 10% penalty for each week late
  - Final score = original score * 0.9 for less than a week late (10/16 ~ 10/22)
  - Final score = original score * 0.8 for one week late (10/23 ~ 10/29)
  - So on…
- Zip and upload the main.cpp on E3
- Zip name : studentID_HW1.zip
  - e.g.

```
313551000_HW1.zip
 |- main.cpp
```

Reference

- https://learnopengl.com/
- https://www.glfw.org/documentation