ICG 2025

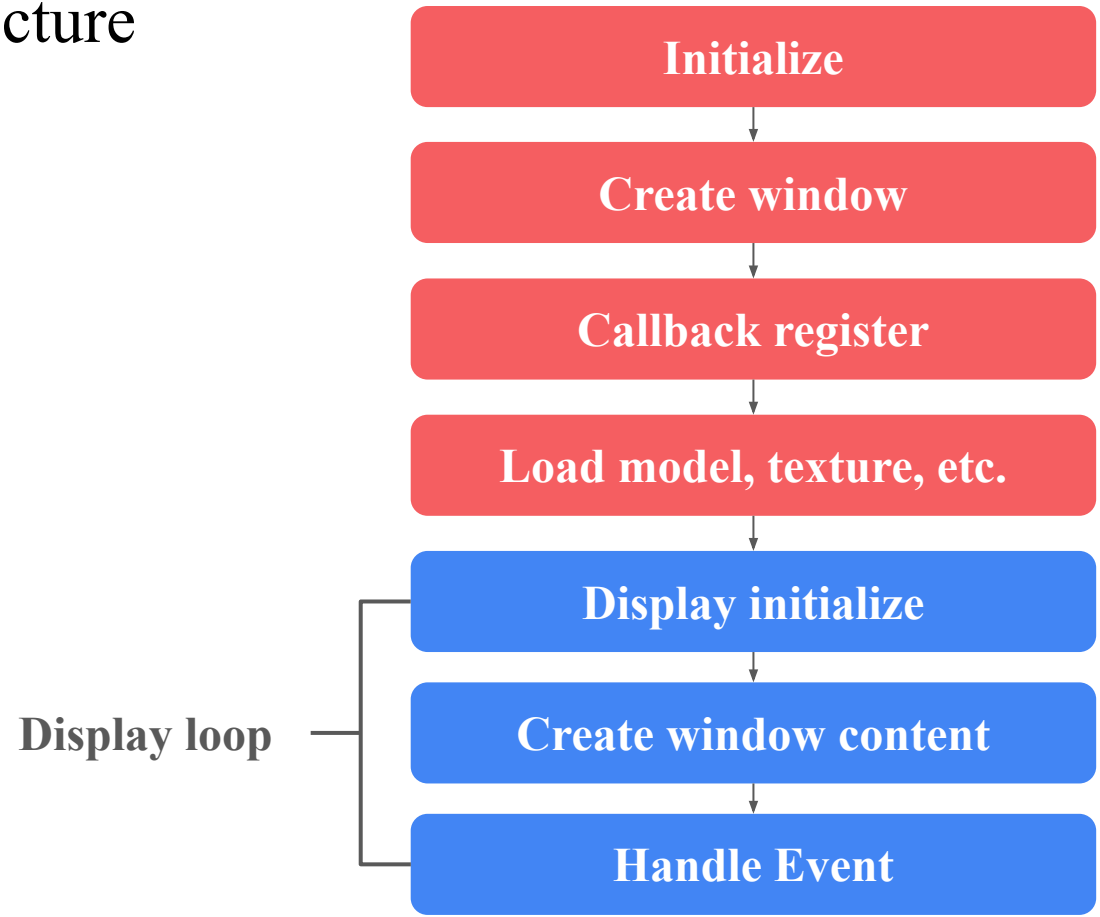
HW1

IDE & Kit

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

- Visual Studio 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
- tinyobjloader (provided in zip)
  - obj loader

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

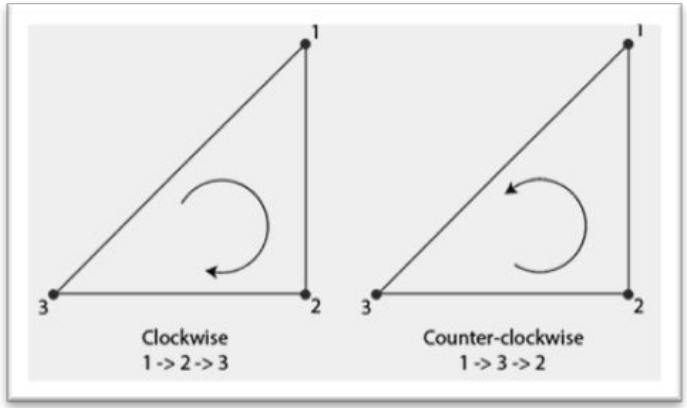
- void glEnable(GL_DEPTH_TEST)
  - While depth test is enabled, OpenGL tests the depth value of each fragment against the content in the depth buffer. If the test passes, the fragment is rendered. If not, the fragment is discarded
- void glDepthFunc(GLenum func)
  - Specify how the test is performed
  - 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

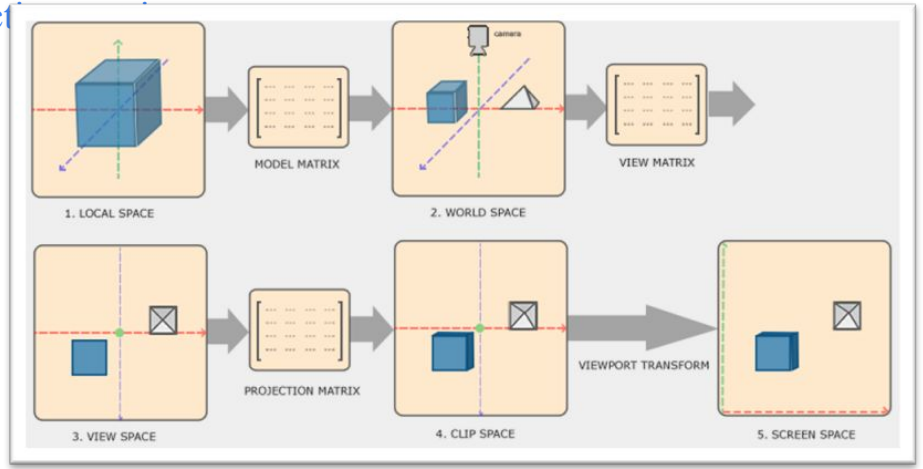
- void glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)
  - Set the color value that is used to reset the color buffer
- void glClear(GLbitfield mask)
  - Clear the specified buffer
  - mask:
    - GL_COLOR_BUFFER_BIT: clear color buffer
    - GL_DEPTH_BUFFER_BIT: clear depth buffer

```
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**,
  const glm::vec3& **color**)
  - **Draw the target model**
  - name: the target model (“cube”, "fish1", “fish2”, “fish3”)
  - model / view/ projection: model / view/ projection
  - color r/ g/ b: red/ green/ blue color (flaot 0~1)

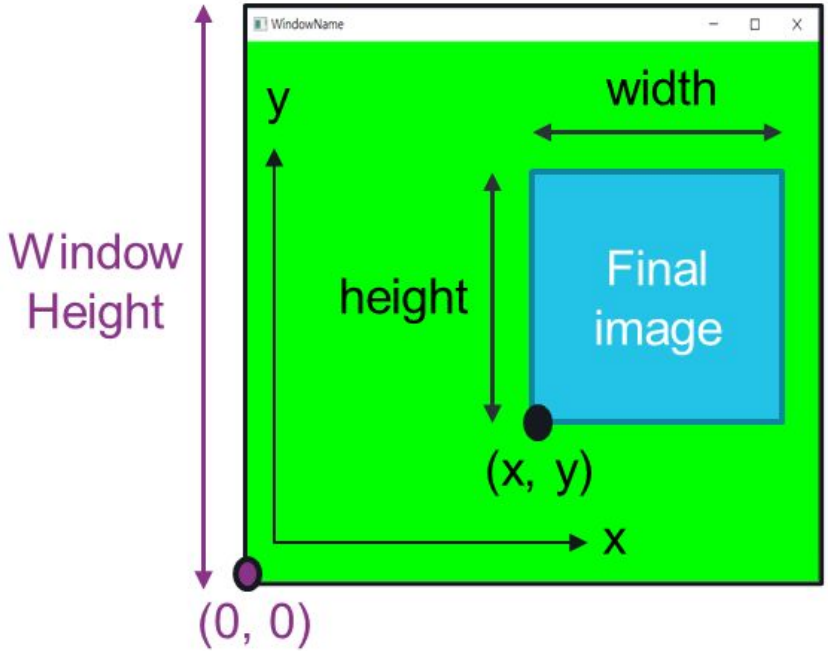
```
drawModel("cube",
          model, view, projection,
          glm::vec3(0.9f, 0.8f, 0.6f));
```



Draw a model$_{2/2}$

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

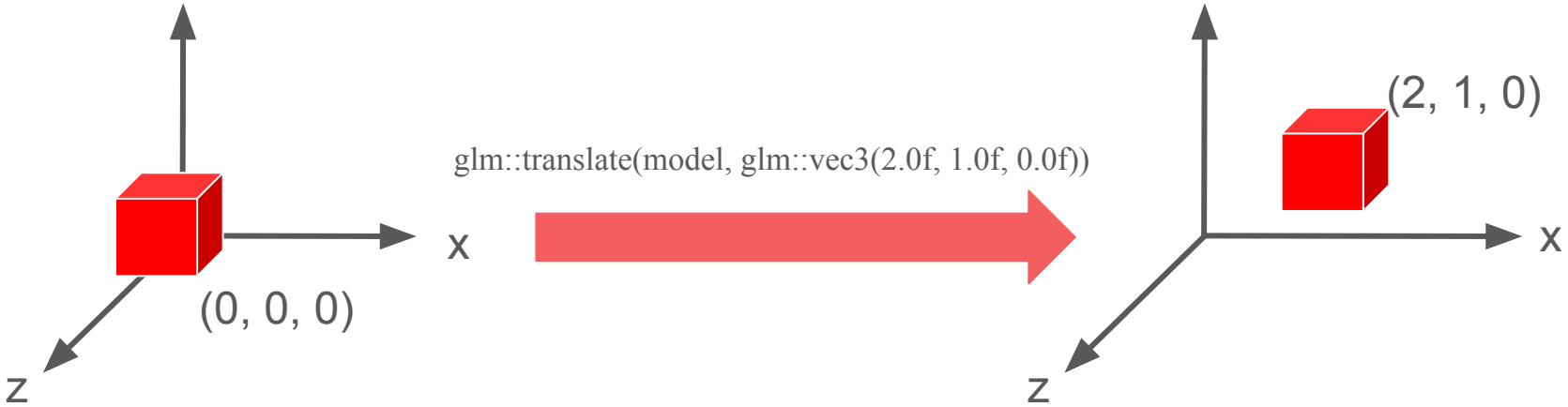
```
glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
```



Model matrix$_{1/3}$

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

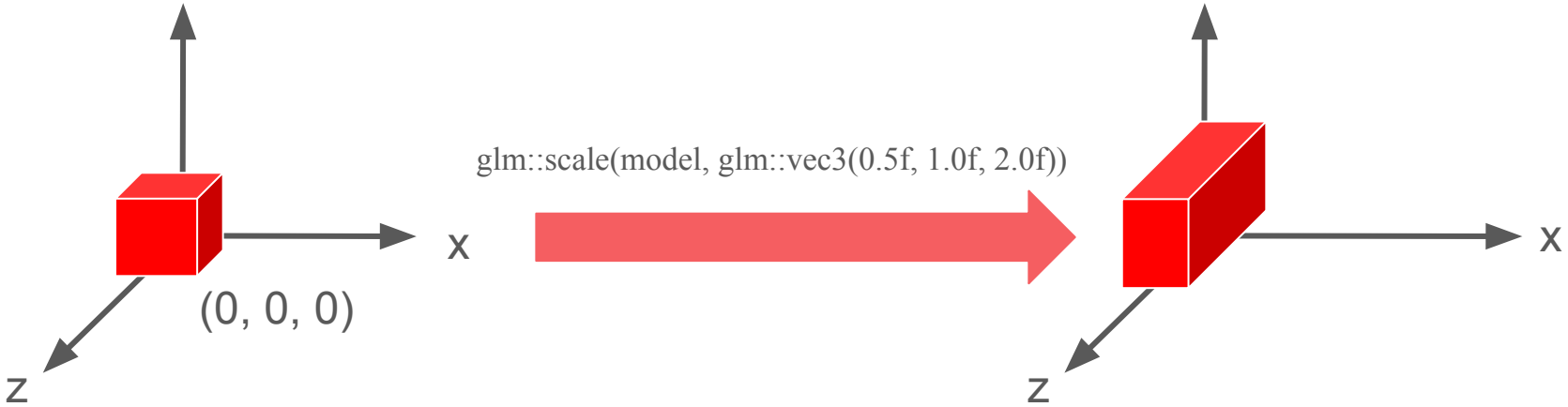
```
glm::mat4 model(1.0f);
model = glm::translate(model, glm::vec3(2.0f, 1.0f, 0.0f));
drawModel("Cube",model,view,projection,255, 0, 0);
```



Model matrix$_{2/3}$

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

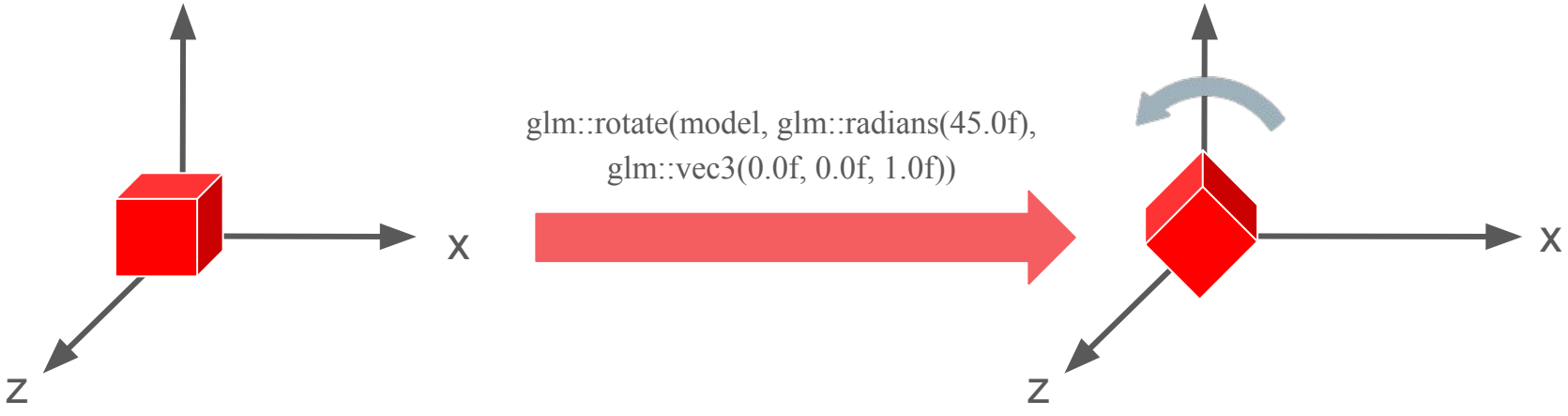
```
glm::mat4 model(1.0f);
model = glm::scale(model, glm::vec3(0.5f, 1.0f, 2.0f));
drawModel("Cube",model,view,projection,255, 0, 0);
```



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);
```

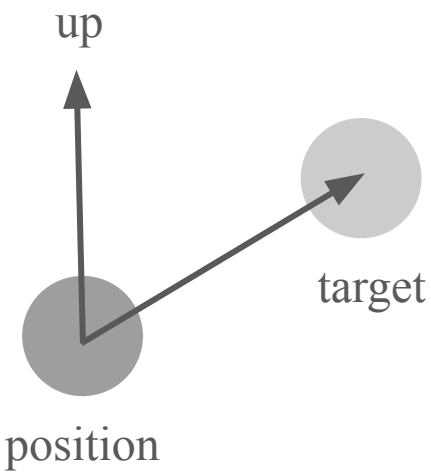
- glm::rotate(glm::mat4 M, GLfloat angle, glm::vec3 axis)
  - Return M * (rotation matrix)
  - The rotation matrix rotates an angle in radians about the given axis
- glm::radians(GLfloat degree)
  - Convert the given degree to radian



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

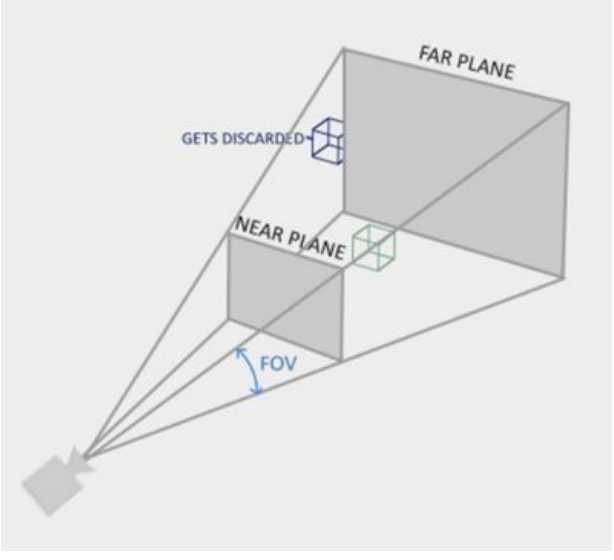
```
glm::mat4 view = glm::lookAt(
                    glm::vec3(0.0f, 50.0f, 90.0f),
                    glm::vec3(0.0f, 0.0f, 0.0f),
                    glm::vec3(0.0f, 1.0f, 0.0f));
```



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

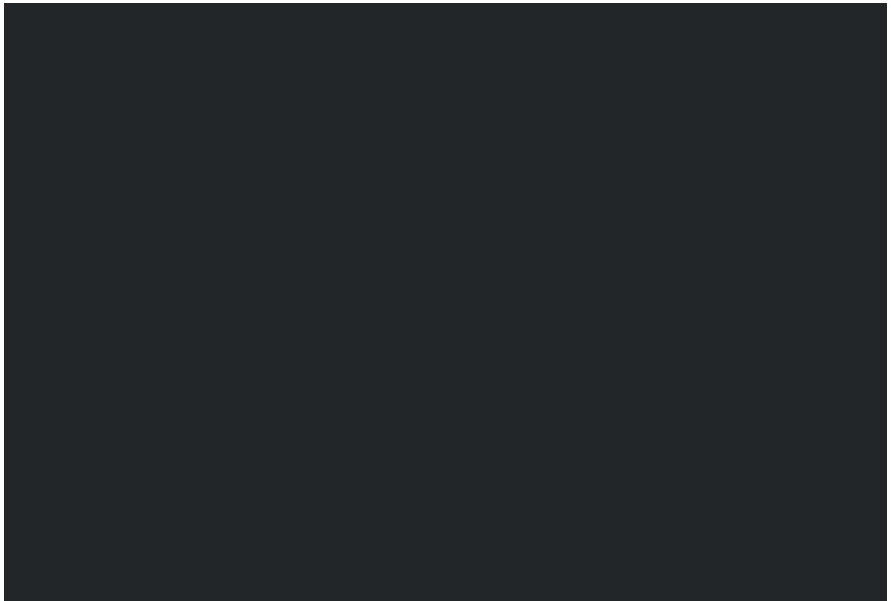
```
void keyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}
```

- 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

At the edge of the AI wave, GPUs have become ubiquitous among users. To help applications leverage GPU benefits, many essential software components are being GPU-accelerated—including terminals. Inspired by the ASCII aquarium commonly found in Unix-like terminals, we will migrate this idea to a GPU-accelerated environment using OpenGL, creating a vivid 3D Aquarium.



Demo

- video

Requirement 1/6

Camera:

Position: (0, 10, 25)

Target: (0, 8, 0)

Up: (0, 1, 0)

Fov: 45

near: 0.1

far: 1000

Aquarium Base(cube):

Position: (0, 0, 0)

Scale: (70, 1, 40)

Color: (0.9, 0.8, 0.6)

Main character

Position: (0, 5, 0)

Scale: Body(5, 3, 2.5)

Color: (0.4, 0.4, 0.6)

Seaweed

Position:

(7.0f, 0.0f, 0.0f), (-7.0f, 0.0f, -10.0f), (-7.0f, 0.0f, 5.0f)

Fish 1(fish1)

Position: (0, 15, 0)

Scale: (2, 2, 2)

Fish 2(fish2)

Position: (7, 3, 0)

Scale: (2, 2, 2)

Fish 3(fish3)

Position: (-3, 7, -7)

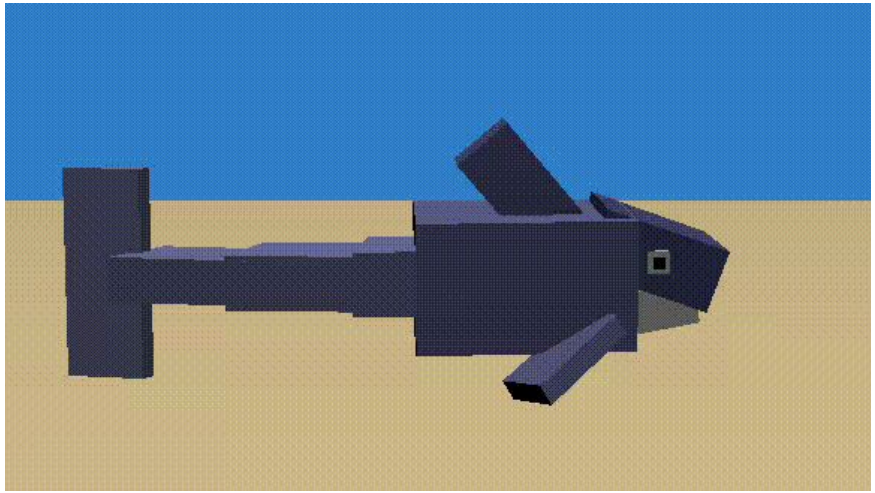
Scale: (2, 2, 2)

Requirement$_{2/6}$

Main character (Head + Body + Fin + Swaying tail) :

- Head (4 cube)
  - **1 for eyes. 1 for pupils. 1 for head. 1 for mouth.**
- Body (One cube) - the origin
- 3 Fin (3 cube each)
  - **On the body**
- Swing tail(4 cubes)
  - **4 cubes with hierarchical connection**
  - Sway: sine wave on each cubes
  - No sway on the last cube
- For player fish,

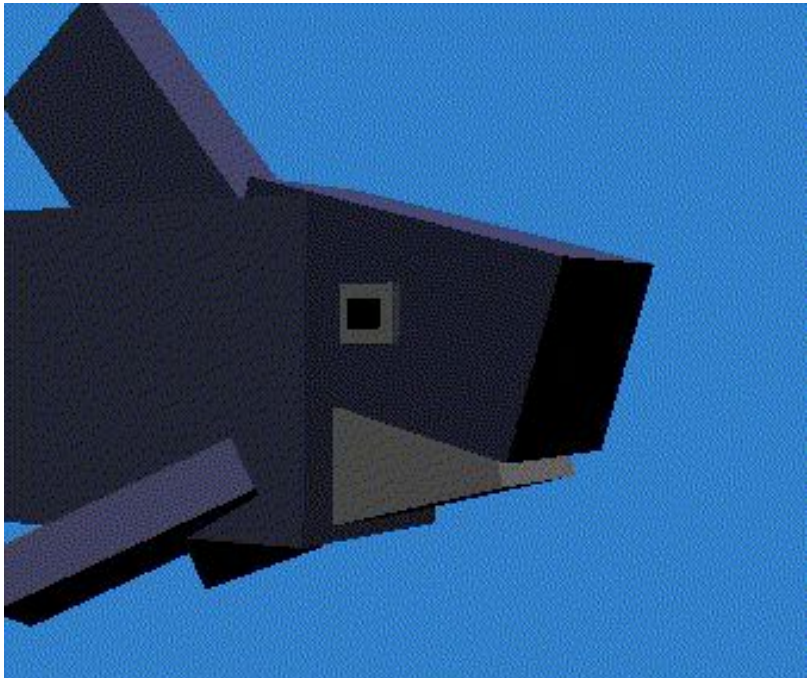
  You can be creative in design !! But keep the tail motion.



Requirement$_{3/6}$

Tooth animation :

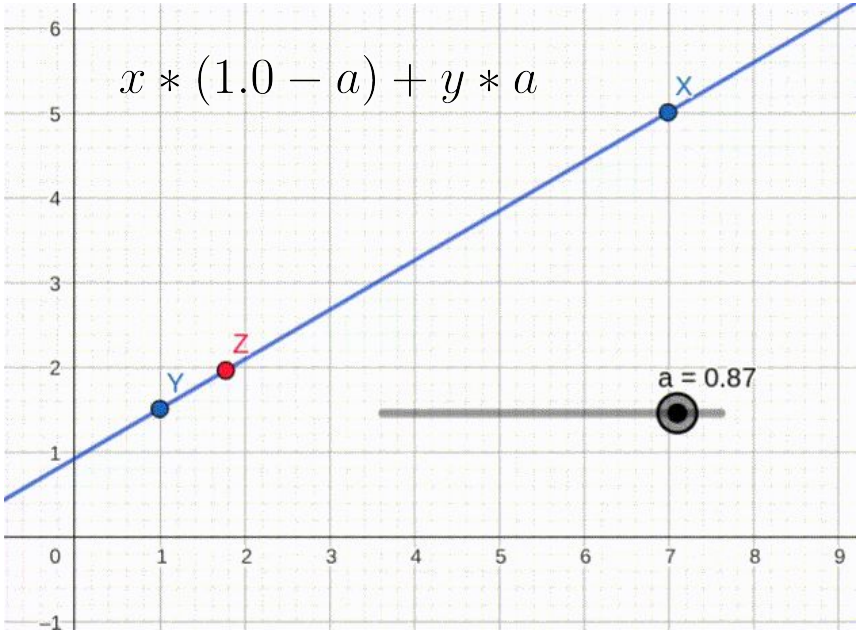
- Teeth
  - **use interpolation from one pos to the other.**
  - there would be duration time and elapse time.
  - make **the elapse time / duration time ratio** as the position ratio of the tooth.
- Mouth Open KeyBinding
  - Press M to open mouth
  - Once opened, the teeth should pop out
  - you may implement the keybinding with the requirement5/6



Requirement$_{3/6}$

- glm::vec3 glm::mix(const glm::vec3& x, const glm::mat3& y, float a)

```
glm::vec3 toothUpperRightPos =
                    glm::mix(pos0, pos1,
                        playerFish.elapsed / playerFish.duration);
```

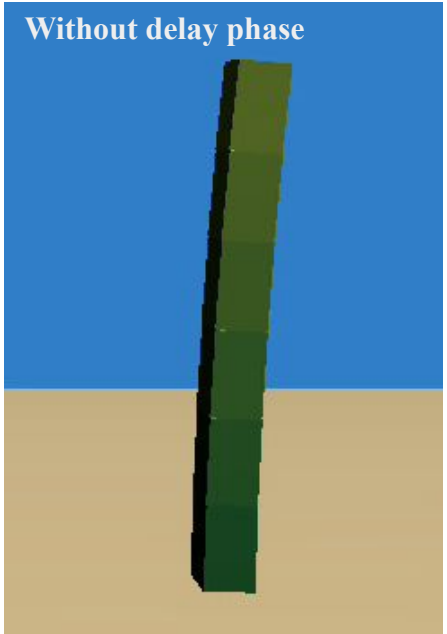


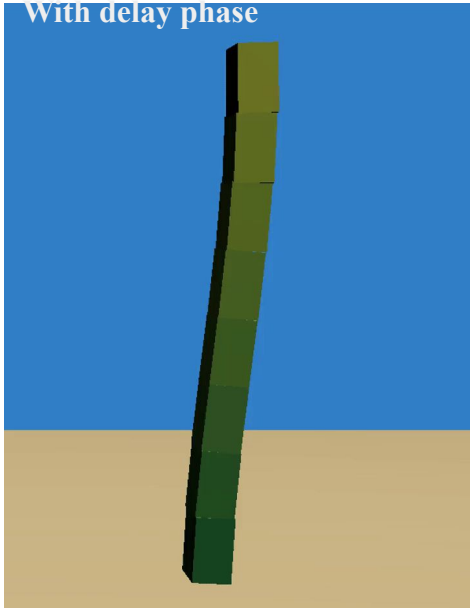
Requirement$_{4/6}$

Seaweed (7 segments) :

- Segments (cube)
  - **cubes with hierarchical connection**
  - Sway: sine wave + delay phase on each cubes
- You can be creative in design !!

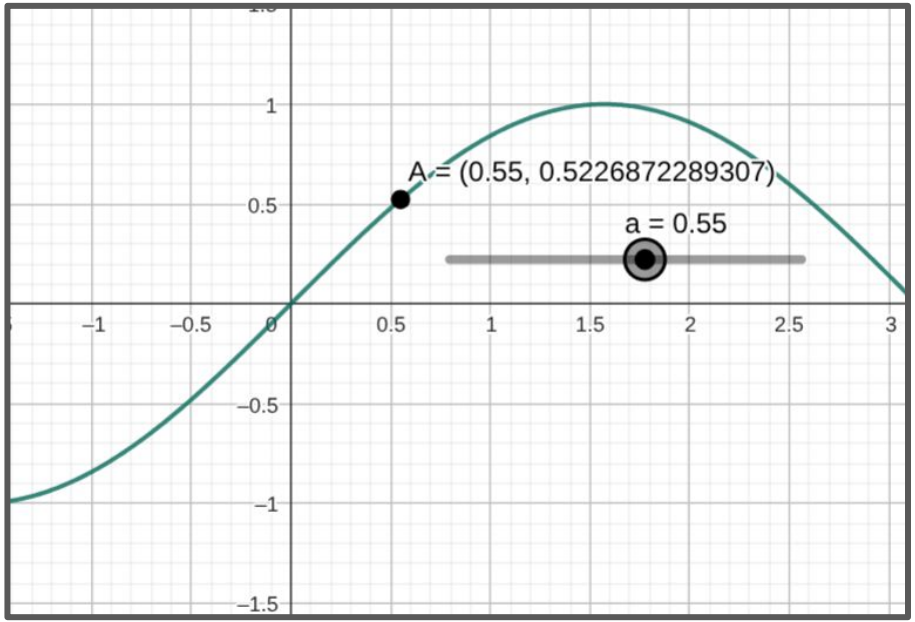
  But keep the sway motion.
- Delay more gradually.

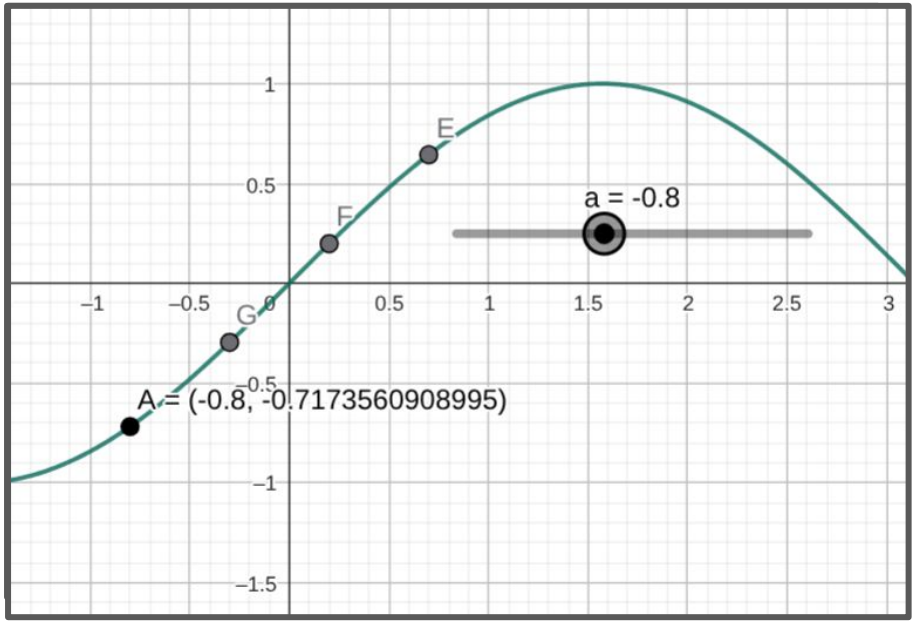




Requirement$_{3/6}$

- Tail wave v.s. Seaweed wave





Requirement$_{5/6}$

Keyboard input:

- Press W to move the character forward
- Press S to move the character backward
- Press D to move the character right
- Press A to move the character left
- Press Space to move the character upward
- Press Shift to move the character downward

- **Please use deltaTime for frame-rate independent.**

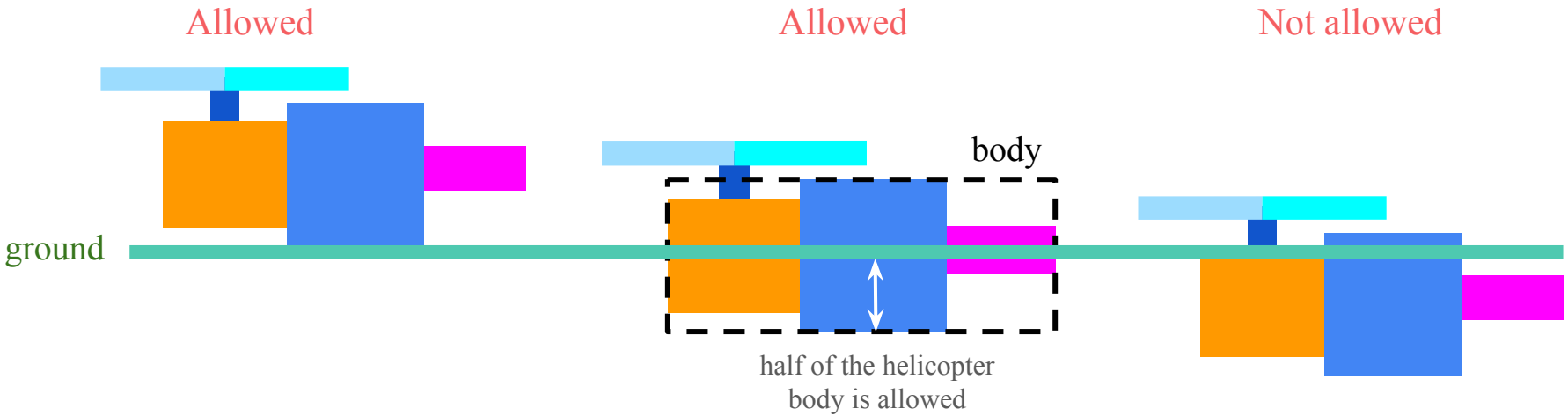
$$\text{position} += \Delta t \cdot \text{speed}$$

$$\Delta t = CurrFrameTimestamp - lastFrameTimestamp$$

Requirement$_{6/6}$

Boundary collision detection:

- Fish cannot swim below the ground.
- An error margin of less than half of the body is allowed.



Score - Basic - 90%

Depth testing (pass if or equal) - 5%

Face culling (counter-clockwise as front, cull back) - 5%

Camera and perspective - 5%

Base and Fish (all transformations must be correct) - 15%

Boundary collision detection - 10%

Seaweeds - 10% Keyboard input - 10%

Seaweed wave animation - 5%

Main character fish - 15%

Main character fish tooth animation - 5%

Main character fish tail animation - 5%

Score - Advanced - 10%

Beautify main character fish/seaweeds - 5%

Add more actions to main character fish - 5%

Depend on your creative!!!

Grades are given on a relative scale, so only the very best can earn a perfect score.

Homework 1 - Submission

- Deadline: 2025/10/8 23:59:59
  - Final score = original score * 0.8 for less than a week late (10/9 ~ 10/15)
  - No score more than a week late
- Zip and upload the main.cpp and demo.mp4 on E3
  - The demo video should demonstrate the above criteria while providing a brief description of the displayed score.
- Zip name : studentID_HW1.zip
  - e.g.

    313551000_HW1.zip

    |- main.cpp

    |- demo.mp4

Reference

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