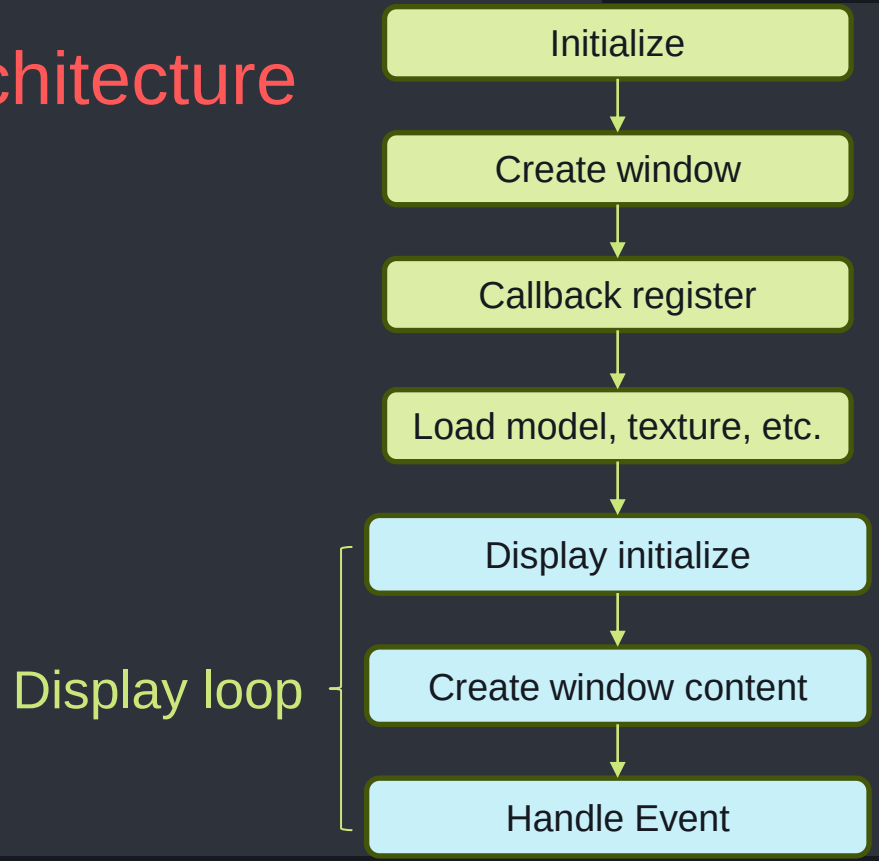
New File

```
Introduction to Computer Graphics {
    [HW1]
}
```

IDE & Kit

* Visual Studio 2019 - Community (download here)
* 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

* 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

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

Initialize & Window

* GLFWwindow* glfwCreateWindow(int width, int height, const char*title, GLFWmonitor* monitor, GLFWwindow*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(windowWidth, windowHeight, "HW1", NULL, NULL);
if (window == NULL)
{
    cout << "Failed to create GLFW window\n";
    glfwTerminate();
    return -1;
}
```

Initialize & Window

* 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

* int gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

  Initialize GLAD to get the OpenGL function pointer

```
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    cout << "Failed to initialize GLAD\n";
    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

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

  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 $\leq$ the depth stored in the buffer

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)

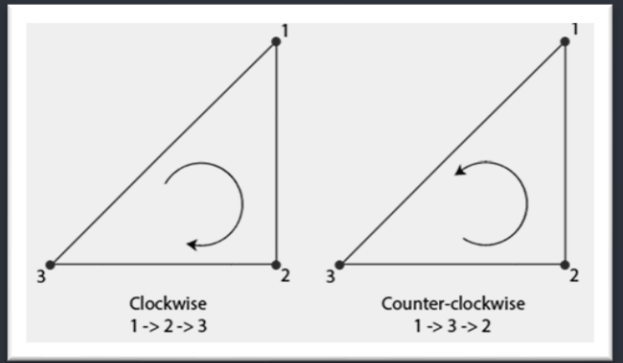
  mode:GL_CW, GL_CCW

  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(0.0f, 0.0f, 0.0f, 0.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

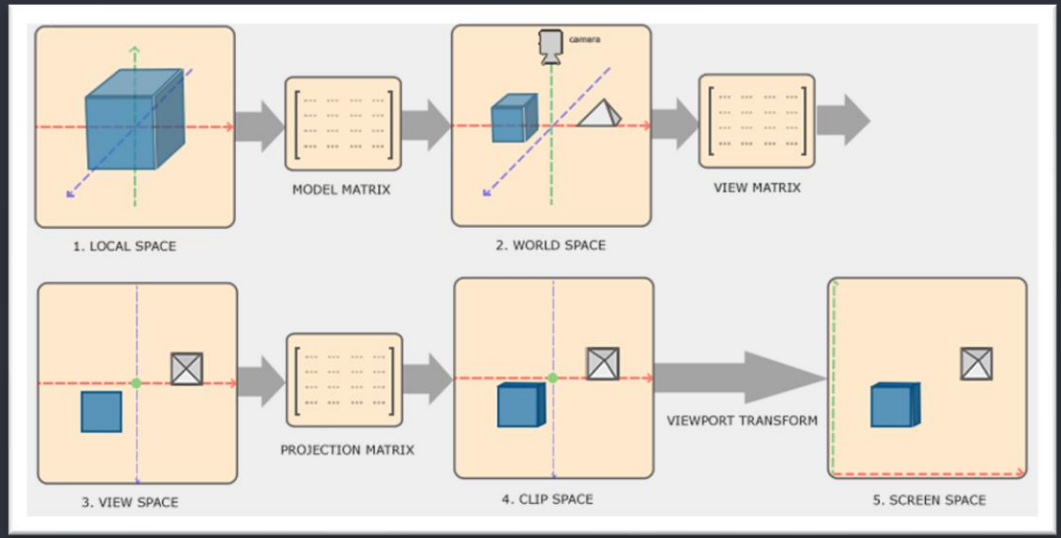
Draw a model

* void drawModel(const string& target, unsigned int& shaderProgram,
  const glm::mat4& M, const glm::mat4& V, const glm::mat4& P)

  Draw the target model (rectangle, triangle, clock, clock hand, rabbit, tortoise)

  * M: model matrix
  * V: view matrix
  * P: projection matrix

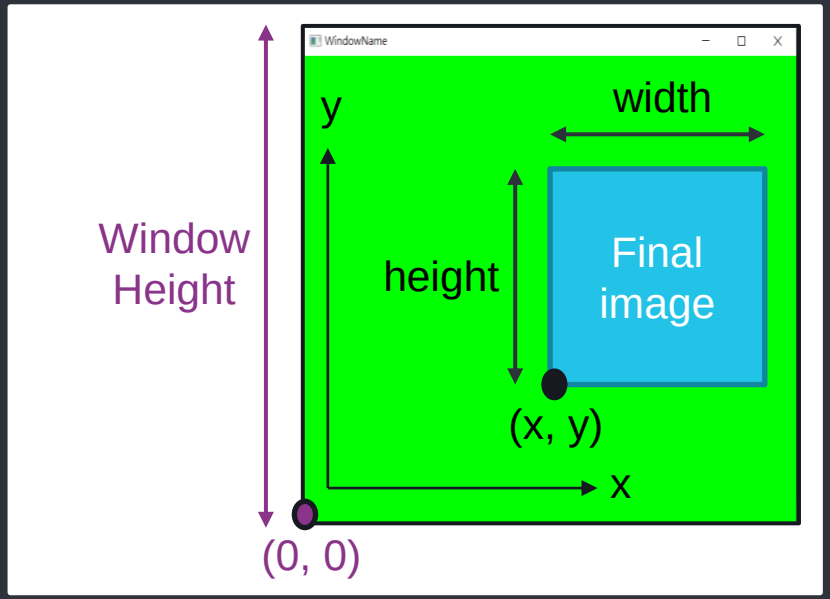
```
drawModel(
    "rectangle",
    shader.program,
    model,
    view,
    perspective
);
```



Draw a model

* void glViewport(GLint x,GLint y, GLint width, GLint height)

  Specify the viewport rectangle

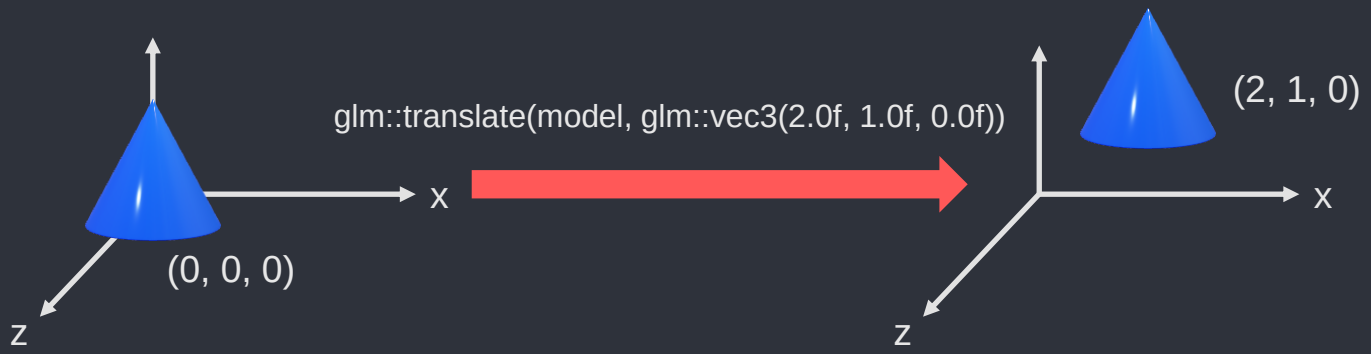


Model matrix

* glm::translate(glm::mat4 M, glm::vec3 translation)

  Return M * (translation matrix)

```
glm::mat4 model(1.0f);
model = glm::translate(model, glm::vec3(0.0f, -3.5f, -5.0f));
drawModel("rectangle", shader.program, model, view, perspective);
```

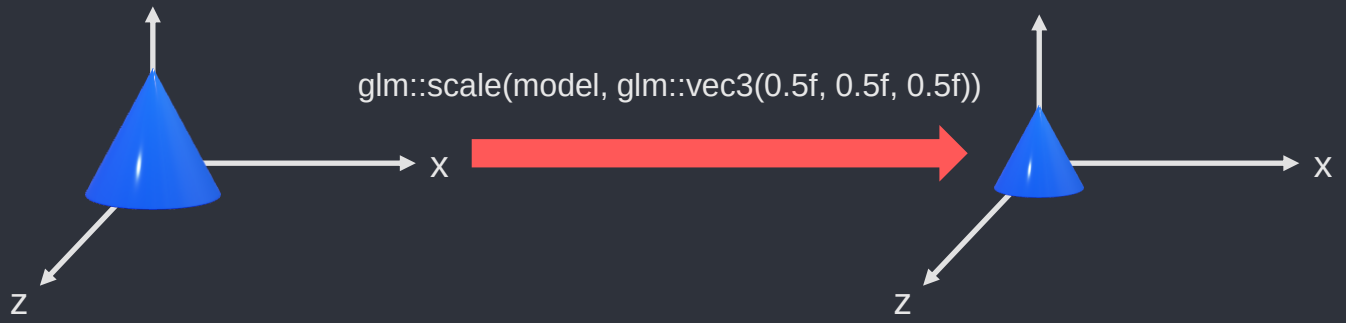


Model matrix

* glm::scale(glm::mat4 M, glm::vec3 scale)

  Return M * (scale matrix)

```
glm::mat4 model(1.0f);
model = glm::scale(model, glm::vec3(4.5f, 10.0f, 3.5f));
drawModel("rectangle", shader.program, model, view, perspective);
```



Model matrix

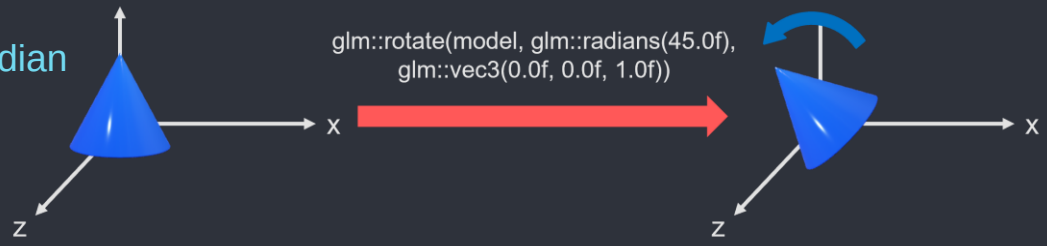
* glm::rotate(glm::mat4 M, GLfloat angle, glm::vec3 axis)

  Return M * (rotation matrix)

  The rotation matrix rotates an angle in radians about the given axis

* glm::radians(GLfloat degree)

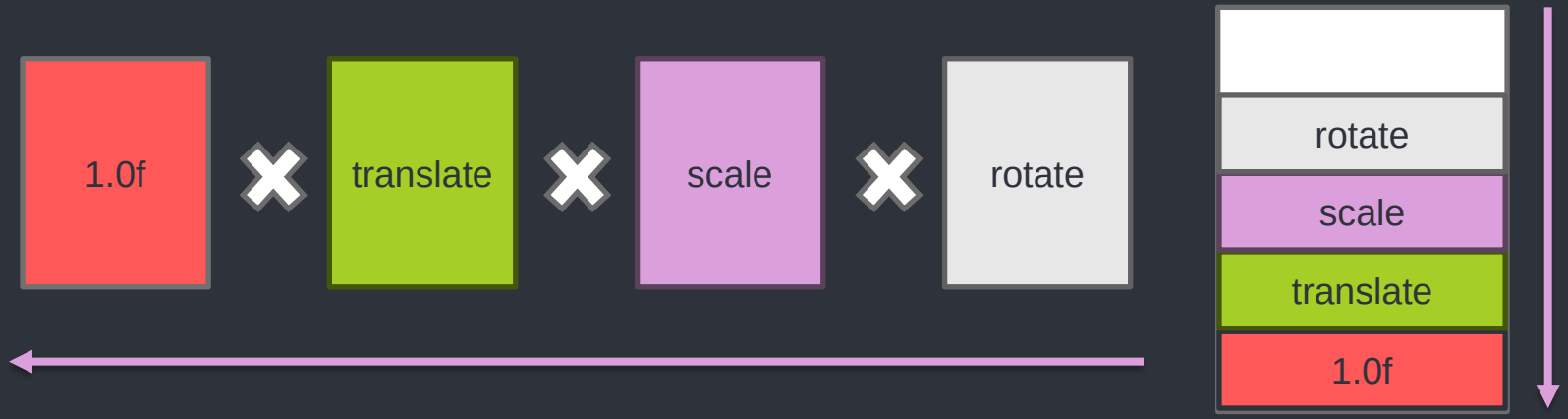
  Convert the given degree to radian




```
glm::mat4 model(1.0f);
model = glm::rotate(model, glm::radians(90.0f),  glm::vec3(0.0f, 1.0f, 0.0f));
drawModel("rectangle", shader.program, model, view, perspective);
```

Model matrix - Example

* model = glm::mat4(1.0f)
* model = glm::translate(model, glm::vec3(2.0f, 1.0f, 0.0f))
* model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f))
* model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f))

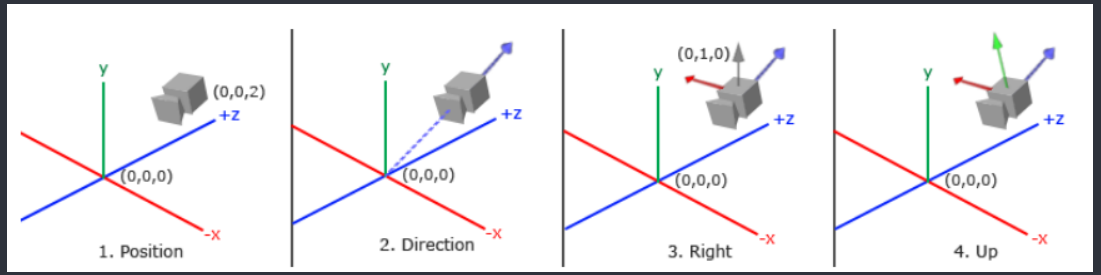


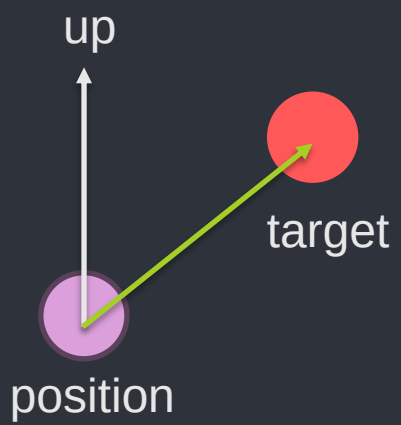
View matrix

* 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, 20.0f, 35.0f),
    glm::vec3(0.0f, 0.0f, 0.0f),
    glm::vec3(0.0f, 1.0f, 0.0f)
);
```



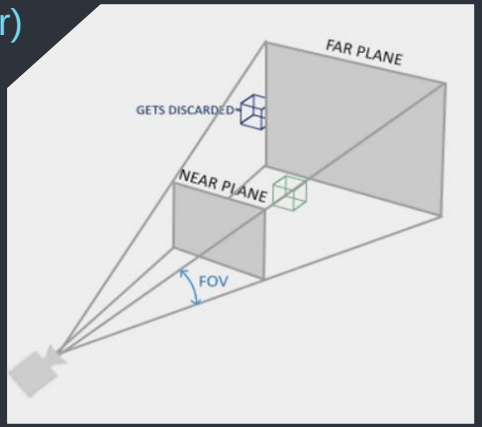


Projection matrix

* 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 perspective = glm::perspective(
    glm::radians(45.0f),
    (float)windowWidth / (float)windowHeight,
    0.1f,
    100.0f
);
```

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);
}
```

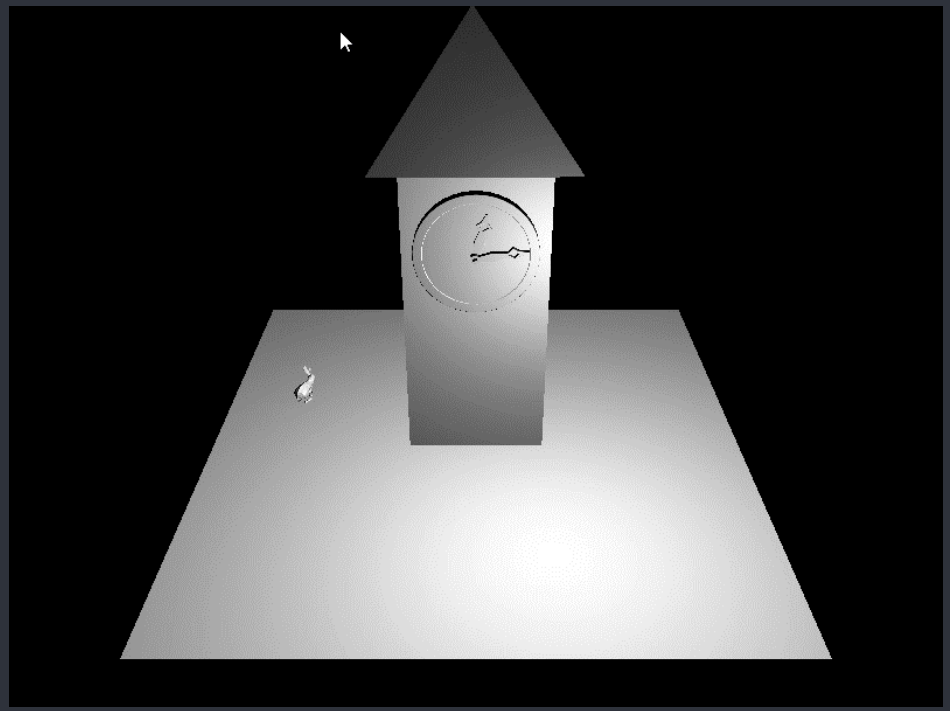
The above function is registered for a key callback. We can check for key events and act correspondingly

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 – Demo



Homework 1

Camera

Position: (0, 30, 50)

Target: (0, 0, 0)

Up: (0, 1, 0)

FoV: 45.0

Near: 0.1

Far: 100.0

Rectangle (ground)

Position: (0, -10, -3)

Scale: (20, 1, 21)

Rectangle (body of clock tower)

Position: (0, 15, 3) relative to the ground

Scale: (4,5, 10, 3.5)

Rotation: 0.5 degrees/frame about +y axis

(start rotating when “3” is pressed)

Triangle (roof of clock tower)

Position: (-0.2, 11.25, -0.35) relative to the body

Scale: (5, 4, 3.3)

Homework 1 – Initial state

Clock

Position: (0, 4.5, 4.3) relative to the body

Scale: (0.013, 0.013, 0.013)

Rotation: 90 degrees about +x axis

Minute hand

Position: (0, 0, 0.6) relative to the clock

Scale: (0.8, 0.7, 1)

Rotation:

1. -180 degrees about +y axis
2. 1 degree/frame about +z axis

Hour hand

Position: (0, 0, 0.25) relative to the clock

Scale: (1, 0.6, 0.6)

Rotation:

1. -180 degrees about +y axis
2. $\frac{1}{60}$ degrees/frame about +z axis

Homework 1 – Initial state

Rabbit

Position: (15, 1, 0) relative to the ground

Scale: (0.08, 0.08, 0.08)

Rotation: -0.7 degrees/frame about +y axis around the clock tower (not (0, 0, 0))

Tortoise

Position: (18, 1.5, 0) relative to the ground

Scale: (0.2, 0.2, 0.2)

Rotation:

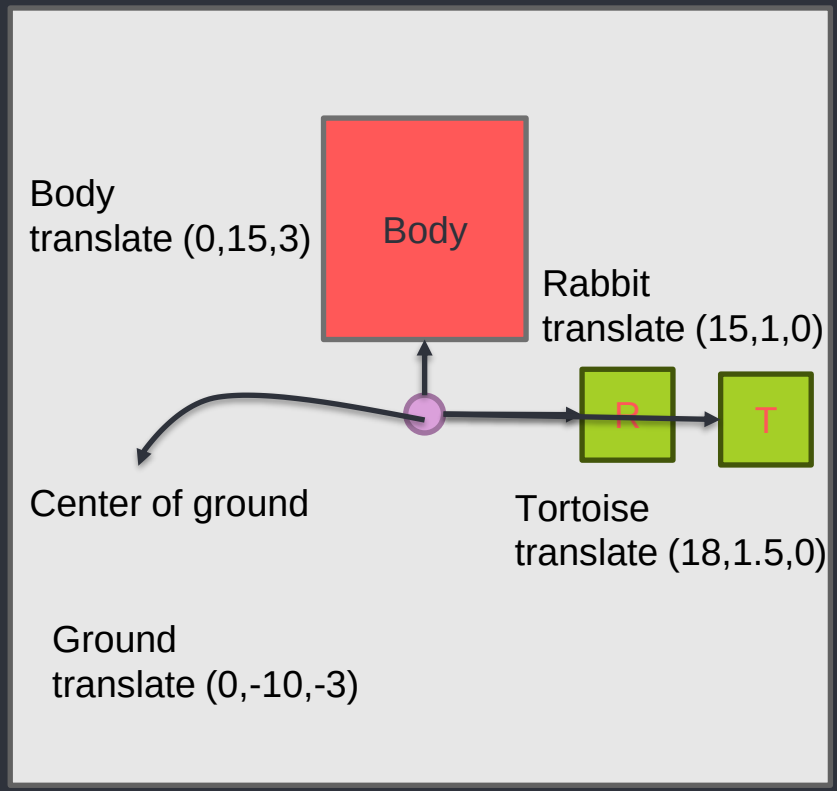
1. -180 degrees about +z axis
2. -90 degrees about +x axis
3. 180 degrees about +y axis
4. -0.35 degrees/frame about +y axis around the clock tower (not (0, 0, 0))

Homework 1 – Keyboard input

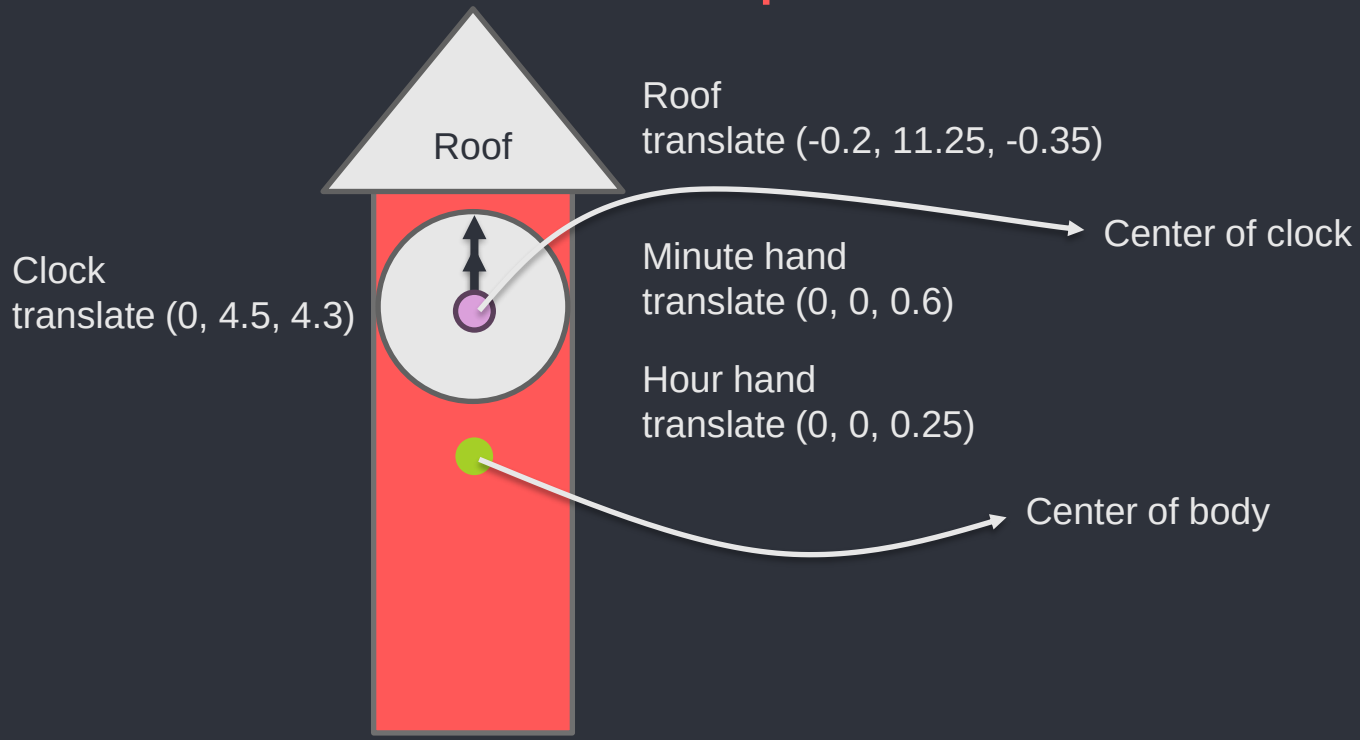
Keyboard input

1. Press 1 to double the speed of rabbit, tortoise, and clock hands
   (Press again to double the speed -> 2, 4, 8, …)
2. Press 2 to halve the speed of rabbit, tortoise, and clock hands
   (Press again to halve the speed -> 1/2, 1/4, 1/8, …)
3. Press 3 to rotate the clock tower, but the rabbit and tortoise should remain unaffected
   (Press again to stop)

Homework 1 – Relative position



Homework 1 – Relative position



Homework 1 - Score

Depth testing (pass if or equal) - 3%

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

Camera and perspective - 4%

Ground (all transformations must be correct) - 5%

Clock tower (all transformations must be correct) - 25%

Rabbit (all transformations must be correct) - 15%

Tortoise (all transformations must be correct) - 15%

All 3 models are correct - 25%

Keyboard input - 5%

Homework 1 - Submission

* Deadline: 2023/10/16 23:59:59
  * 10% penalty for each week late
  * Final score = original score * 0.9 for less than a week late
  * Final score = original score * 0.8 for one week late
  * So on…
* Zip and upload the visual studio project on E3
* Zip name: studentID_HW1.zip

Reference

https://learnopengl.com/

https://www.glfw.org/documentation