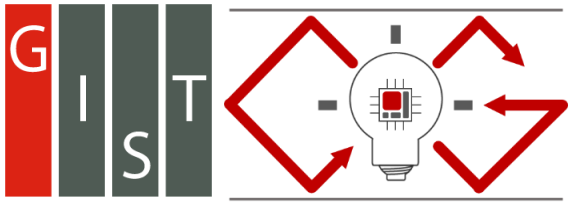
Programming Assignment 2

2026 Computer Graphics



Computer Graphics
Laboratory

2026/04/06

Seongil Kim

Contents

Notice

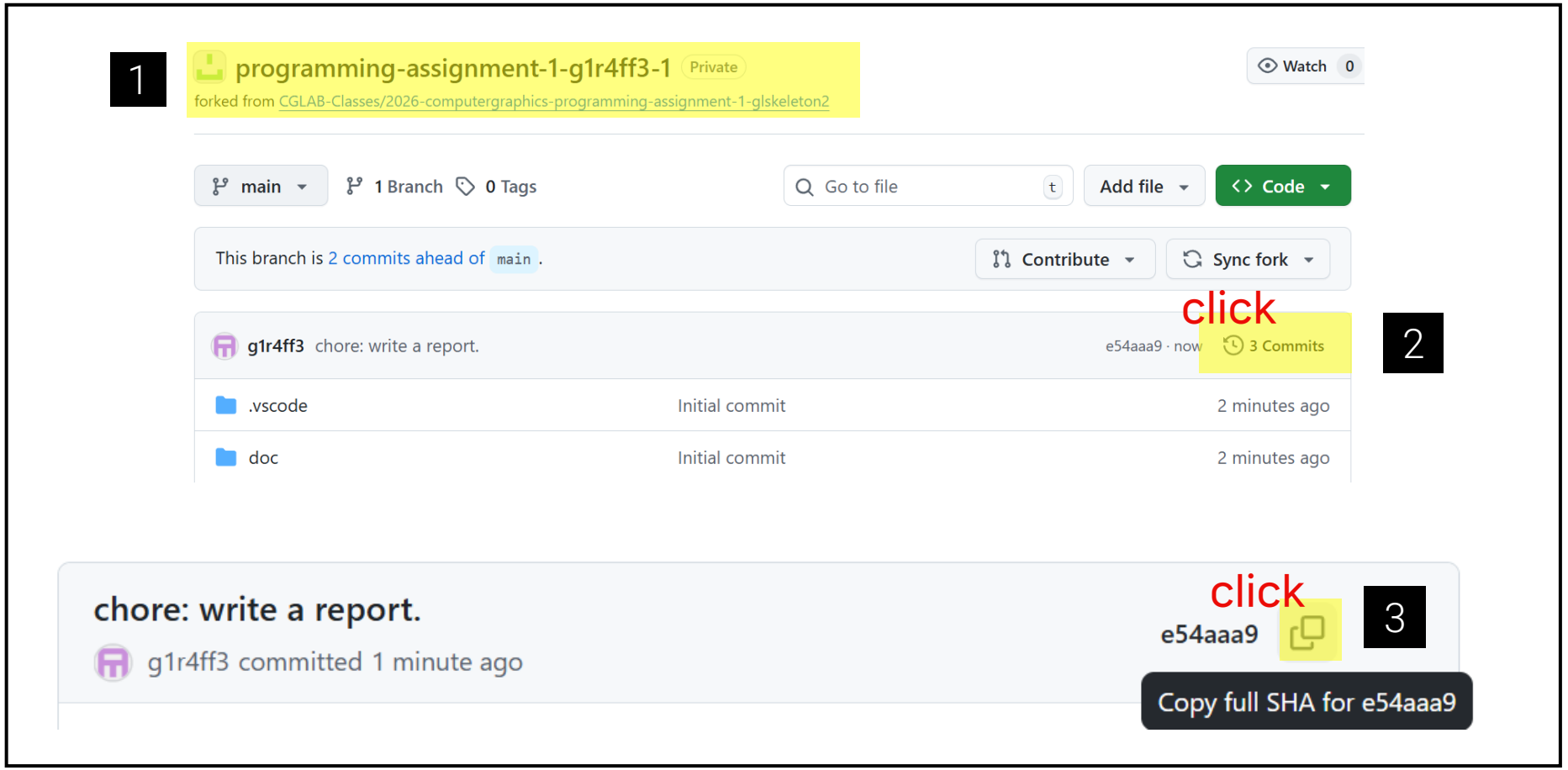
Submission

- Deadline: 23:59:59, Sunday, April 26th, 2026 (KST, +0900)
  - GitHub server clock

- To submit your assignment, two things must be done BEFORE deadline.
  - You must push your commit to your repository
  - You must comment the last commit id (SHA-1 hash) in github issue board
  - You must match username of pushed commits with GitHub username

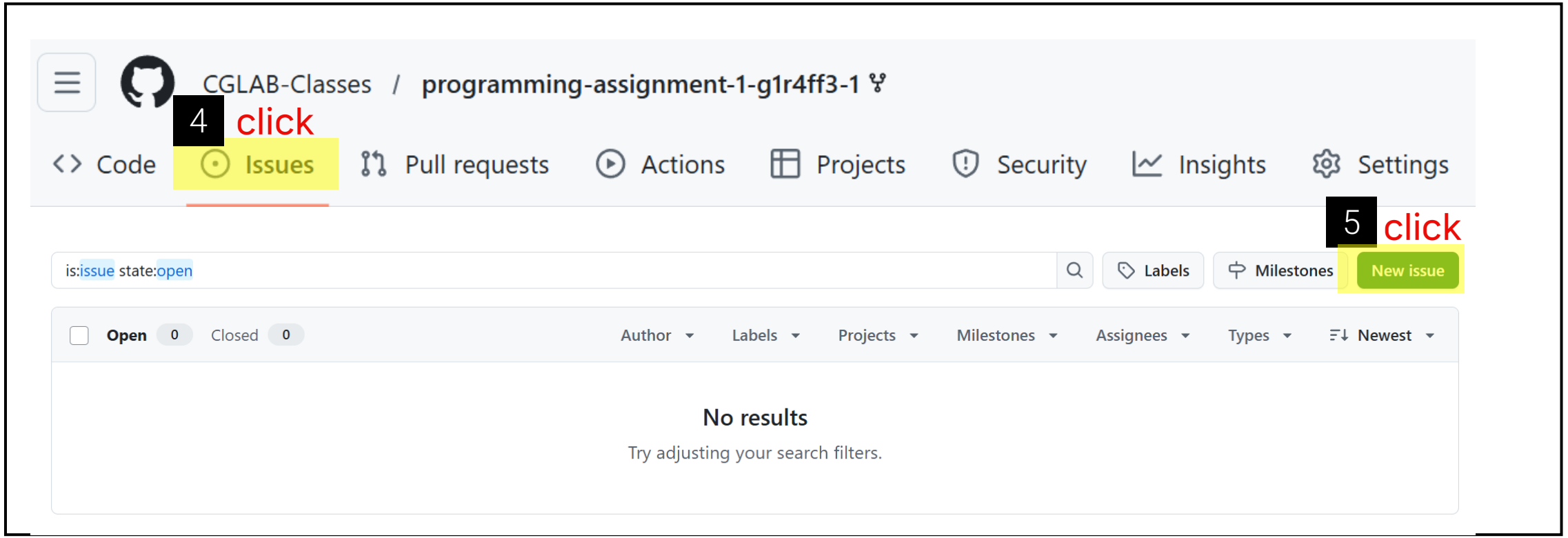
- The last commit in the issue board BEFORE deadline will be considered as submitted assignment.
  - Local timestamp in your commit will be ignored. GitHub server timestamp used instead

How to comment a commit id



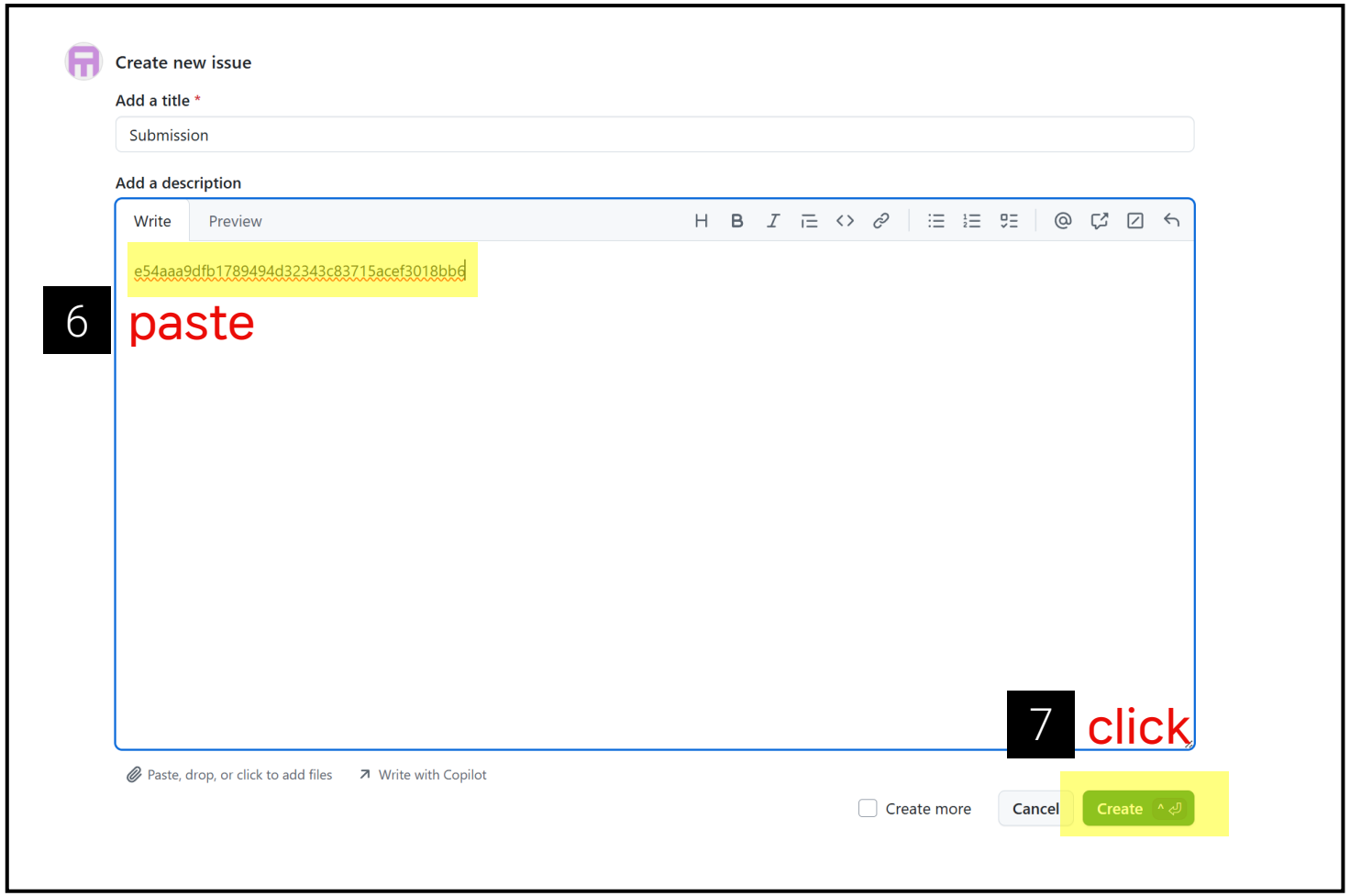
1. Go to your assignment repository
2. Click "Commits"
3. Click copy button of your last commit

How to comment a commit id



4. Go to issues tab
5. Click "New Issue"

How to comment a commit id



6. Paste your latest commit id (ctrl-v)
7. Click "Create"

Policies

- In following cases, your grade for this PA will be "0"
  - Late submission
    - late push after deadline or late last commit id comment on issue board
  - Build/execution failure
  - Making public of your assignment repository
  - Mismatch of username in a submitted local commit with the submitter's GitHub username
  - API use of a library that gives the solution directly, instead of working your assignment on your own

- Your final grade will be "F"
  - Copy (We will run a copy detector)

Assignment Overview

Prerequisites: User Interface for OpenGL

- Why use?
  - OpenGL is just a library or tool for graphics
  - We need a way to communicate or interact with OpenGL
- GLFW (Graphics Library Framework)
  - C library for OpenGL that provides windowing operations, manages OpenGL context and receives events arousing from input devices.
- Example: Register a callback for keyboard events

```
void key_callback(GLFWwindow* window, int key, int action, int modes) {
  if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
    glfwSetWindowShouldClose(window, GLFW_TRUE)
}
glfwSetKeyCallback(window, key_callback);
```

- For more information about GLFW, refer to the official quick tutorial
  - https://www.glfw.org/docs/latest/quick.html

Prerequisites: User Interface for OpenGL

- NanoGUI
  - a minimalistic cross-platform widget library that builds on GLFW for context creation and event handling
- Object-oriented pattern for communicating with OpenGL
  1. Simpler window management compared to using GLFW alone
  2. Define custom event handlers through overriding instead of using function pointers
  3. Easy layout for rapid prototyping of GUI components
- Example: Register a callback for keyboard events through overriding

```
class MyApp: public nanogui::Screen {
  bool keyboard_event(int key, int scancode, int action, int modifiers) override {
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
      set_visible(false);
    return true;
  }
};
```

- Note
  - All event handlers return a boolean
    - true = event consumed / false = propagate to parent or child
  - Unhandled events must be handled by calling a handler in parent

```
return nanogui::Screen::keyboard_event(key, scancode, action, modifiers);
```

Keyboard Events

- For pressing, releasing or repeating physical keys,

```
virtual bool keyboard_event(int key, int scancode, int action, int modifiers)
```

Parameter	Type	Description
key	int	GLFW key code (e.g. `GLFW_KEY_A`, `GLFW_KEY_ESCAPE`)
scancode	int	OS-specific physical scancode (not directly used)
action	int	`GLFW_PRESS`, `GLFW_RELEASE`, `GLFW_REPEAT`
modifiers	int	Bitmask: `GLFW_MOD_SHIFT`, `GLFW_MOD_CONTROL`, `GLFW_MOD_ALT`, `GLFW_MOD_SUPER`

- For taking characters of keyboard,

```
virtual bool keyboard_character_event(unsigned int codepoint)
```

Parameter	Type	Description
codepoint	unsigned int	UTF-32 Unicode codepoint

Mouse Events

- For pressing or releasing a physical mouse button,

```
virtual bool mouse_button_event(const Vector2i &p, int button, bool down, int modifiers)
```

Parameter	Type	Description
p	const Vector2i&	cursor position in screen space (top-left origin)
button	int	GLFW_MOUSE_BUTTON[_LEFT|_RIGHT|_MIDDLE]
down	bool	true = press, false = release
modifiers	int	same as in keyboard events

- For tracking mouse motion,

```
virtual bool mouse_motion_event(const Vector2i &p, const Vector2i &rel, int button, int modifiers)
```

Parameter	Type	Description
p	const Vector2i&	current cursor position in screen space (top-left origin)
rel	const Vector2i&	displacement vector between current one and previous one
button	int	bitmask of the currently held buttons
modifiers	int	modifier bitmasks

- For handling scroll,

```
virtual bool scroll_event(const Vector2i &p, const Vector2i &rel)
```

Parameter	Type	Description
p	const Vector2i&	Current cursor position in screen space (top-left origin)
rel	const Vector2i&	Scrolling displacement vector (rel.y() > 0 means scroll up)

Other Events

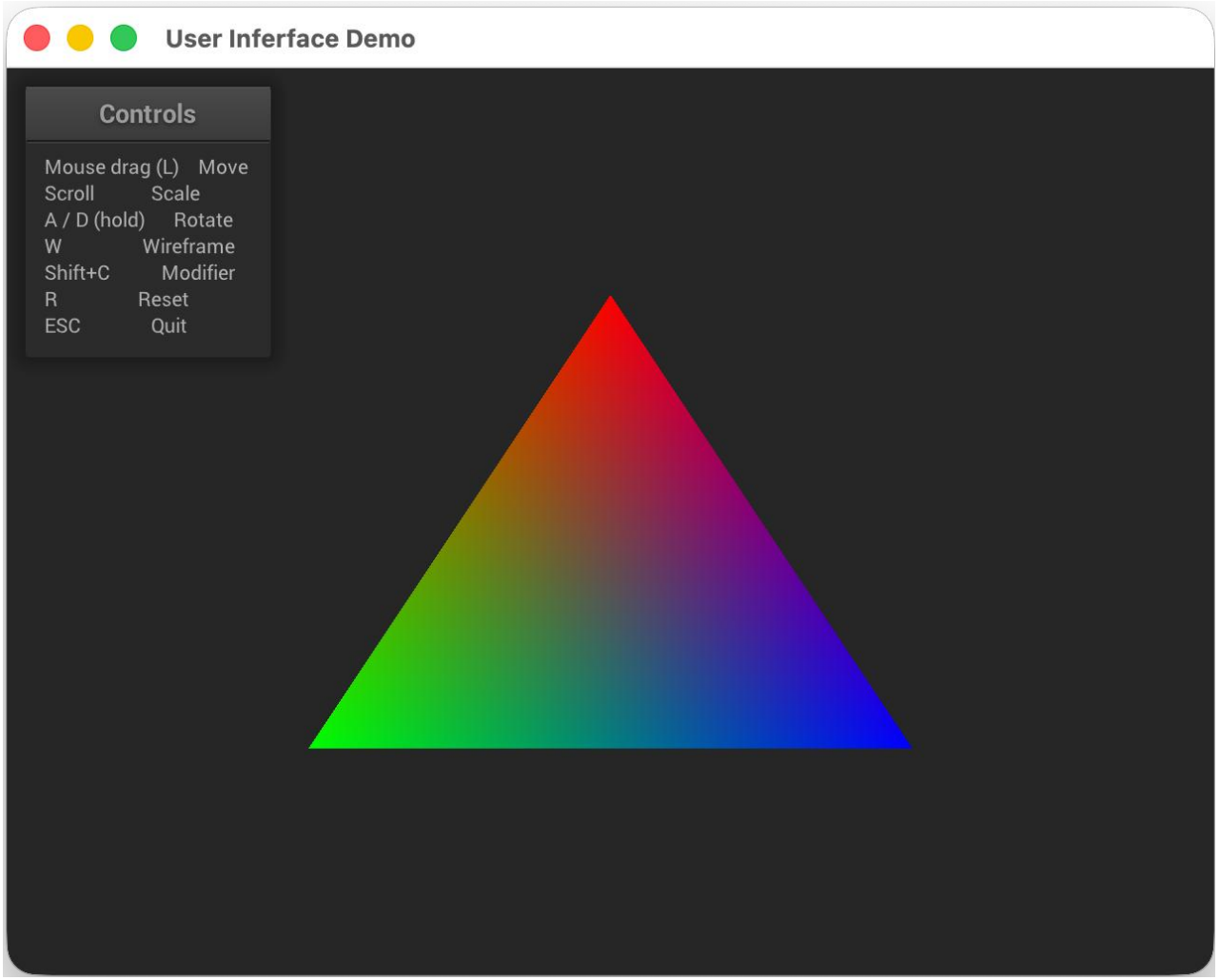
- For resizing the window,

```
virtual bool resize_event(const Vector2i &size)
```

Parameter	Type	Description
size	const Vector2i&	new window size

- For detailed tutorial about how to handle events, refer to https://www.glfw.org/docs/latest/input_guide.html

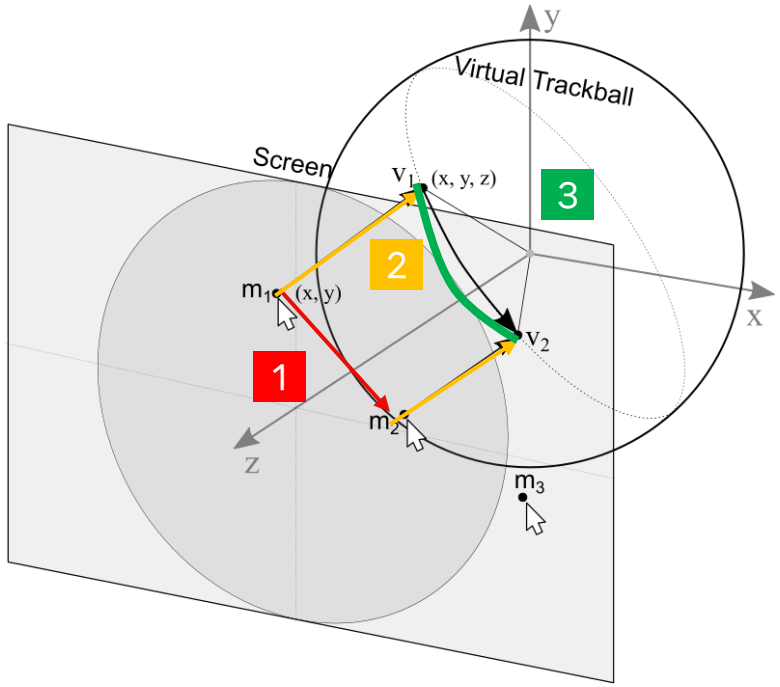
Demo – User Interface



Let's Interact with a rainbow triangle

Prerequisites: Trackball Camera

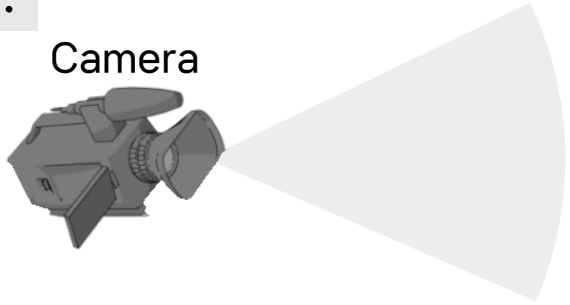
- Camera that orbits around a virtual trackball (sphere)
  - In other words, rotating a camera origin by displacement vector from user interaction
- Basic idea
  1. User inputs two mouse positions by dragging a screen in a window
  2. Project the two points onto a sphere representing virtual trackball, bounding a target object
  3. Once we have two directional vectors from the sphere origin to corresponding points, we can compute rotation matrix by Rodrigues formula or quaternion of how much we should rotate the camera
  4. Based on the rotation or the quaternion, rotate the camera

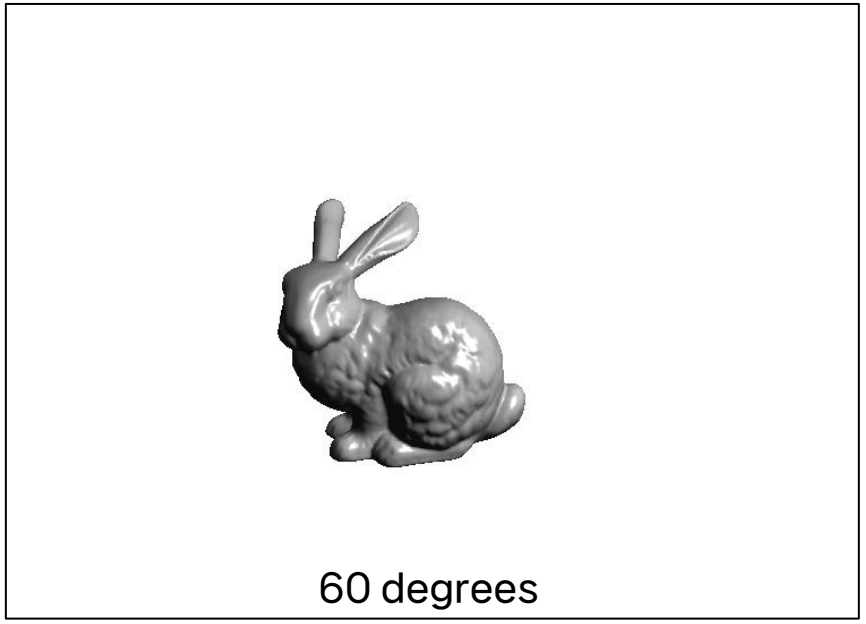


Prerequisites: Trackball Camera

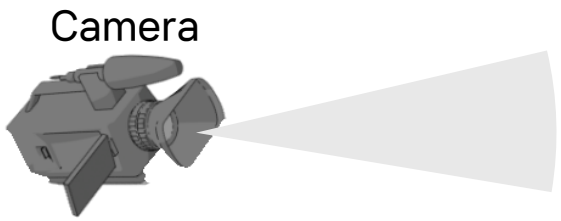
- Zoom In & Out
  - Change vertical field of view (fov) angle of camera

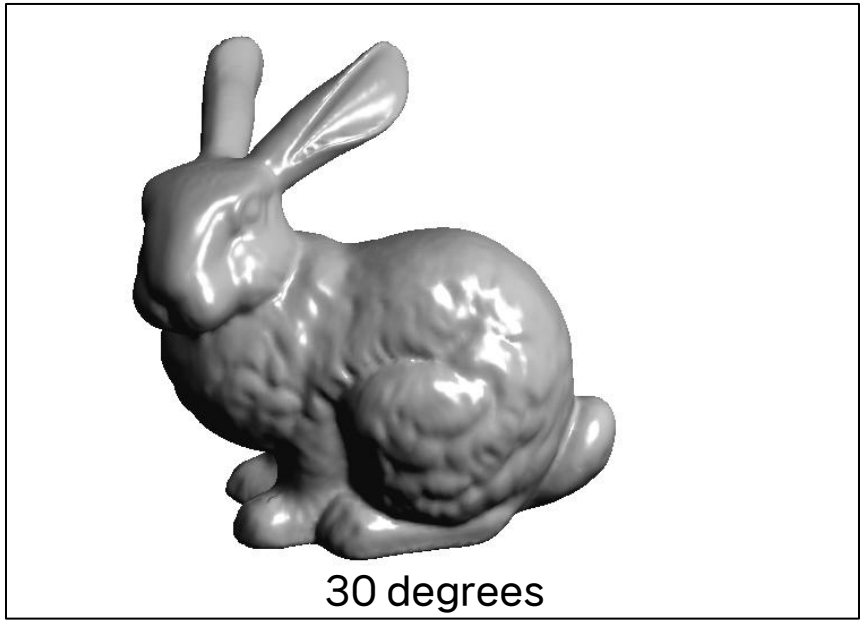
Zoom In!





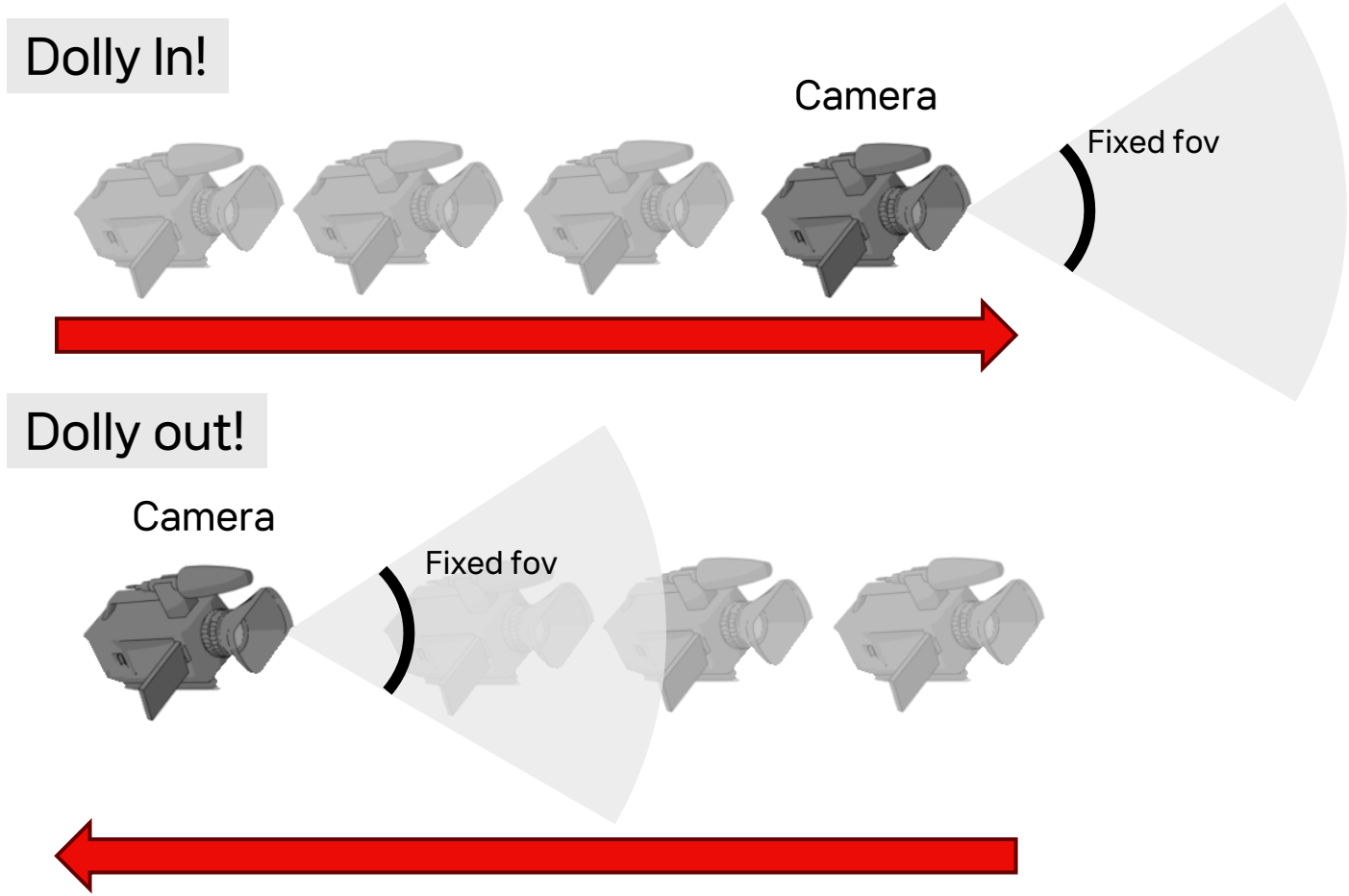
Zoom Out!

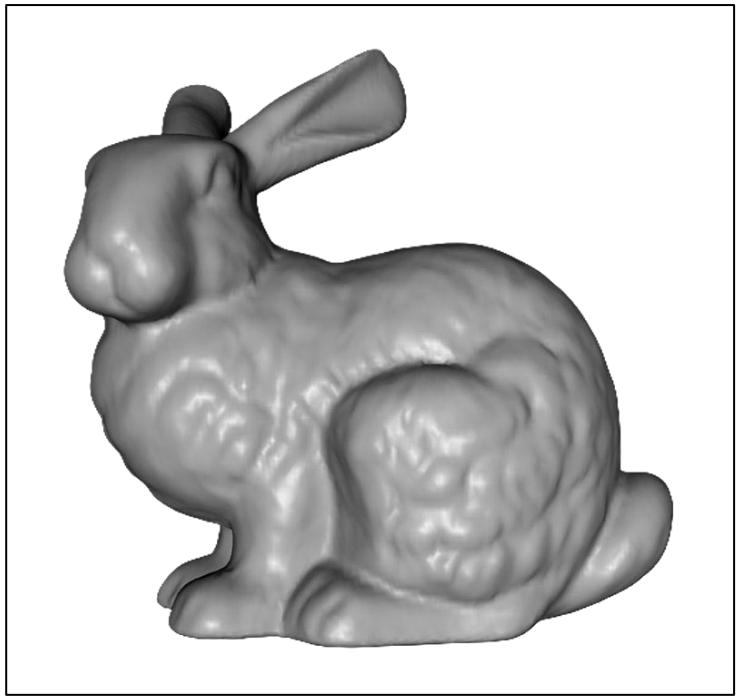




Prerequisites: Trackball Camera

- Dolly In & Out
  - Translate a camera origin along a specific direction





Scene

Tasks

1. Implement Trackball Camera with following features, and apply to your 3D model [10 Points]
   - Drag or Orbit around [8 Points]
     - Fix the lookat position to a center of your 3D model or simply world origin
     - The edge case of outside the trackball: either method is fine
   - Zoom in & out [1 Points]
   - Dolly in & out [1 Points]
2. Implement Phong illumination model in fragment shader with following light sources, and apply to your 3D model [8 Points]
   - Directional light that can be toggled by a user interface element [4 Points]
   - Point light that can be toggled by a user interface element [4 Points]
   - Number of lights is not restricted, but recommend to use appropriate one
3. Write a report [2 Points]
   - Write your name, student id, GitHub username in report.md [1 Points]
   - Attach at least two result images in report.md [1 Points]
   - For the rest of the report, feel free to describe or write your thoughts for this assignment (Optional)

Accept your PA2!
https://classroom.github.com/a/Y-Glb7GF

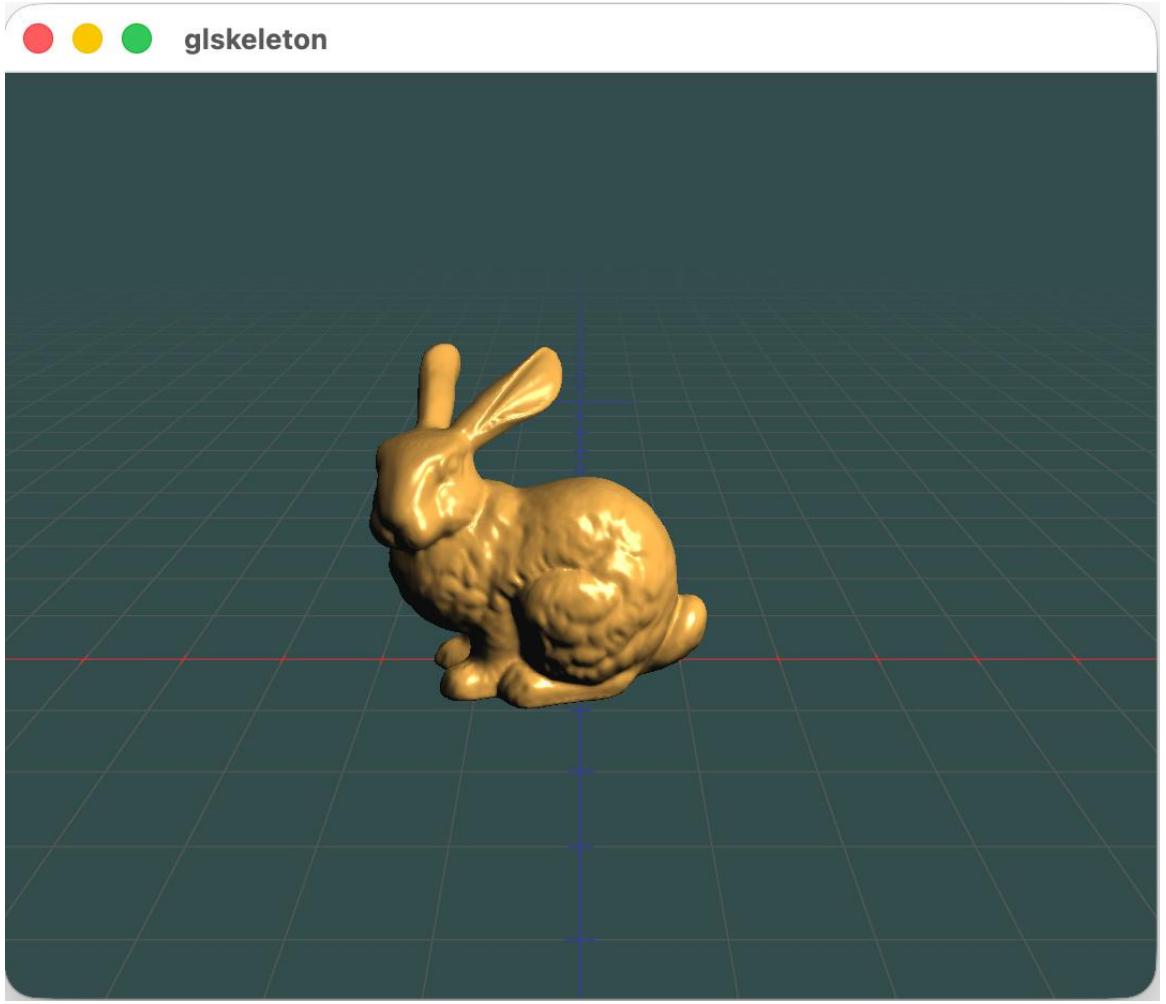
Tips for tasks

- Task 1
  - You need to have understood about transformations in the rendering pipeline
  - Take into account cases that mouse positions are outside the trackball
    - Refer to Shoemake's method or Bell's method [link]
- Task 2
  - Use uniform variables to send read-only variables from CPU and access globally between shaders (docs)
  - Shading would be incorrect if normal vectors are transformed wrongly in fragment shader.
    - Refer to resources about normal matrix (link1, link2)

Additional resources

- GLFW
  - API documentation
- Trackball Camera
  - Detailed explanation
- Phong shading
  - Basic Lighting for learning how to implement Phong illumination model practically
  - Materials for learning how to set light properties
  - Light casters for learning types of light sources
- OpenGL Wiki
  - https://wikis.khronos.org/opengl/Main_Page

Demo – PA2



Camera orbits around the bunny illuminated by a point light and a directional light

Q & A

- Email
  - Seongil Kim: sngillkym@gm.gist.ac.kr
  - Hyunjin Jung: hjjung0810@gm.gist.ac.kr
- Office: 104 Dasan Bldg