Warmup

Due: Feb. 9, 2016

Introduction

Welcome to CS1972! In this assignment you’ll be creating the basic framework of the game engine you will be developing for the rest of the semester. It will introduce first and third-person movement in a 3D world, basic application and game world organization, and some important OpenGL concepts such as texture loading. By the end of these two weeks, you’ll also have a simple 3D game built on top of your engine!

All assignments in this course will follow a similar format of requiring both engine features and a game built on the engine, so they all have two sets of requirements: one for the engine, and one for the game. These requirements will be further split up into primary, secondary, and tertiary requirements, where primary requirements are most imperative to complete and necessary for your following projects. Your handin must meet at least the primary set of requirements, as well as a set of global requirements, to receive credit. For more information, see our first powerpoint for our grading policy. Global requirements can be found under Docs and Resources, as well as in each project rubric.

To make sure you fulfill all of the requirements for a given checkpoint, you should copy the rubric for the checkpoint into your README. The rubric will contain the same requirements as this document, but should be easier to copy directly into your README. Rubrics can be found at /course/cs1972/rubrics/<asgn>/grade.txt.

Before moving on, we recommend reading the Course Missive on the Docs and Resources page to make sure you understand what will be expected of you in this course.

If you want to see an example of what Warmup might look like, feel free to run cs1972 demo warmup{1,2} on any department machine.

Support Files and IDE

Getting Started

If you haven’t yet, read the first few pages of the winter assignment. It contains instructions for setting up a work environment on your personal computer as well as more detailed descriptions of the support code.

To get the support code for this and all future assignments, copy the contents of /course/cs1972/asgn/warmup to your project directory for Warmup (probably something like ~/course/cs1972/warmup). This should give you a basic Qt project including a Qt pro file that configures and helps build your project, as well as directories containing some starter code and resources. This code should compile and run right away, give you a black window and a framerate counter.

We strongly recommend using the Qt Creator IDE for this course. It has excellent refactoring, code completion, and debugging capabilities in addition to a built-in UI designer should you need to develop a level editor. Run the qtcreator command and open warmup.pro to load the project.
Files ending in pro are text files containing the project configuration (a list of sources, compiler flags, and platform-specific build commands). There are several things that make using Qt Creator especially easy:

- Ctrl+R: Build and run your project (standard out appears in the Application Output pane)
- Ctrl+K: Quickly open any file in the project by name (in addition to any class or function)
- Ctrl+Click: Jump to the definition of any symbol (variable, function, macro, etc.)
- F4: Switch between *.h and *.cpp files with the same name

This list is by no means complete: feel free to let us know about any other convenient commands you find.

Support Code

A major part of game engine development is being able to design large and complex software systems. For this reason, the support code for CS1972 is minimal. You will need to implement the majority of each project from scratch, so make sure to allocate time for design.

That being said, we realize you have limited time and want you to focus on what’s interesting, so we have provided a few support files to get your started:

- **view.{h,cpp}**: Defines a View widget extending QGLWidget. This is a starting point for your game engine; it sets up a full screen window with mouse capture and a variable-update game loop. Every update of the game loop calls tick() to handle game updates and triggers paintGL() to redraw the view. You will want to fill in these methods when implementing Warmup.

- **mainwindow.{h,cpp,ui}**: Initializes a main window containing a View widget. You should not need to modify these files.

- **main.cpp**: Starts the program.

- **util/ResourceLoader.{h,cpp}**: Loads and compiles shader files. You should not need to modify these files.

- **util/CylinderData.h**: Contains mesh data for a cylinder. You will need to use this data to render a cylinder in week 2.

- **util/CommonIncludes.h**: Contains include statements for universally needed classes and libraries, such as glm. You may add whatever you want to it, however it will be included in many, many files, so try to keep it as small as possible.

Note: when you create new folders, you may want to add them to INCLUDEPATH and DEPENDPATH in warmup.pro so you can #include files inside them directly.
Resources

In addition to the support code, we may also provide resources such as textures and models. In this assignment, the only resource we provide is a grass texture entitled grass.png (from opengameart.org). Feel free to find other textures or models while doing your projects, but keep in mind that you cannot use any associated code.
Week 1 - Due Feb. 2

Warmup 1 will get you used to working in 3D space. Though there won’t always be a ton of requirements related to gameplay, this is ultimately a class about making games, so have fun with these assignments, and feel free to build more than is required! What is listed here is just the bare minimum for getting a complete on a checkpoint.

Design Check

- List the steps involved with setting up a first person camera. How will you get or compute the parameters you need?
- How will you define an application? What about a screen?
- In broad terms, describe the steps and OpenGL calls necessary to render the floor.
- How will you implement gravity, the floor, and jumping?

Common Engine Requirements

The following will all be a part of your “Common” engine. For now, you don’t need to know what this means. Just know that the following features should be applicable to almost any game, and as a result, should be logically separated from, but utilized by, your game code:

Primary Requirements

- First person camera support
  - Yaws around the up (positive Y in OpenGL) axis
  - Pitches in the horizontal plane
  - Camera pitch is locked to the range \((-\pi/2, \pi/2)\)
  - Forward-backward and left-right strafe style movement of the eye
- Screen management
  - Virtual Screen class representing a logical subscreen of a game which (minimally) supports:
    * Timed updates (tick)
    * Render events (draw)
    * Input events (mouse and keyboard)
    * Window size updates (resize)
  - Virtual Application class representing a whole game which supports custom OpenGL initialization and, minimally, a list of Screens
- Custom Graphics object
  - Supports the loading of textures from image files
– Supports the storage and binding of unique OpenGL texture IDs by a string key
– Supports the storage and drawing of shapes by a string key
– Supports rendering a 1x1 textured quad
– Maintains an active shader program that is used for drawing

Game Requirements

For this week, you will not implement any real gameplay. Your handin should allow the player to walk around a world using mouse and keyboard inputs to change the camera. The player will be able to jump and not fall through a textured floor, but there will be no “point” to the game.

Primary Requirements

• The player must be able to move using standard first person camera controls, meaning:
  – Horizontal mouse movements change the yaw of the camera
  – Vertical mouse movements change the pitch of the camera
  – Standard WASD keyboard controls (W moves forwards, S moves backwards, A strafes left, D strafes right) change the eye of the camera

Secondary Requirements

• The player must never fall through the ground at $y = 0$
• The player must be able to jump off the ground using the spacebar or a mouse button
• The player can only jump when on the ground
• Gravity must act downwards on the player
• The ground must consist of planar geometry with a grass texture (does not have to be tiled).

Tertiary Requirements

• The game must have at least two screens, one of which requires player input to get to the other
• The ground consists of planar geometry with a tiled grass texture. This means the floor is a series of 1x1 quads each with the same texture, not a single quad with a stretched texture

Keep in mind that there are also global requirements that apply to every checkpoint. Be sure to confirm that you meet these as well!
Week 2 - Due Feb. 9

Warmup2 will add world organization functionality to your game engine, as well as a basic collision system and a third-person camera. By the end of the week, you’ll have your first fully playable game!

Design Check

- How will your camera support both first and third person?
- Describe the steps to determine if, and by how much, two cylinders are colliding.
- What are the steps involved with determining a collision? Which are engine-side, and which are game-side?
- What kind of gameplay will you implement?

Engine Requirements

Like week 1, all code written this week will be a part of your “Common” engine:

Primary Requirements

- Third person support built into your camera, with the ability to toggle between the two cameras
- Uses a primitive or Shape class that wraps all OpenGL calls related to drawing arbitrary geometry, including:
  - VAO/VBO creation and cleanup
  - Sending data to the VBO
  - Drawing the geometry
- Entity/World hierarchy
  - Virtual Entity class representing a single logical game object with (minimally) the following contract:
    * Timed updates (tick)
    * Render events (draw)
    * Creation of a collision shape (for now, getCylinder)
    * Instance variables for position, velocity, and an acceleration accumulator
  - Virtual World class representing a collection of Entities with (minimally) the following contract:
    * Timed updates (tick)
    * Render events (draw)
    * Entity management (add/remove an Entity)
- Cylinder-cylinder collision detection
  - Detection - determine that two cylinders are overlapping
Secondary Requirements

- Cylinder-cylinder collision resolution, and generation of a response callback
  - Resolution - translate the cylinders out of detection using the MTV
  - Response - dispatches collision callback to collided Entities

Game Requirements

Now that you have some basic collisions, you can complete your first game! We’re intentionally vague in the following requirements to encourage you to think outside the box: what can you do with just cylinder-cylinder collisions and a flat floor? We’re looking forward to seeing the results.

Primary Requirements

- All warmup1 primary and secondary game requirements must be fulfilled
- The player can use a key to toggle between first and third person cameras

Secondary Requirements

- Cylinder-cylinder collision response must be used somewhere (your game code should just do game-specific reactions to a collision, like taking damage or bouncing away)
- There must be at least 1 NPC (non-player character), such as an enemy or opponent.

Tertiary Requirements

- At least 1 NPC with basic AI. It does not need to be controlled by a complicated AI, but cannot sit still or jump in place
- The game must have a non-arbitrary and reasonably achievable win or loss condition
- The game must be resettable upon win or loss without restarting the program
- The game can never enter an unwinnable or unloseable state

Handing In

Hand in the entire directory tree for your project, including both your engine and game code. You must also include a README file that describes how to verify each requirement, and an INSTRUCTIONS file that describes how to play your game, as specified in the Global Requirements. To hand in, run `cs1972_handin warmup` from the top level directory of your project (which should be where your Qt pro file is), where $n$ is the checkpoint you are handing in. Please do not hand in the build files from your project.