Platformer

Due: Apr. 5, 2016

Introduction

In this assignment you will create your own 3D platformer from scratch. You will start out by implementing collision detection and response for a player moving in a world made of triangles. Once this is in place, you will add enemies that use path-finding over a navigation mesh. Finally, you will add your own simple gameplay, including a way to win the game.

Note that while this project is called platformer, the game you make does not necessarily need to be a platformer. These implementations simply allow for a world with platforms: you could make a shooter, a racing game, or any number of other genres for this project!

Demos of the checkpoints are available by running `cs1972 demo platformer{2,3,4}`.

Support Files

For the first week of this project, you’ll be working in a collision debugger project that we’ve created to make implementing ellipsoid-triangle collisions a bit easier. This project can be found at `/course/cs1972/asgn/collision_debugger`. However, by the end of week two, you will move all of your collision code into your engine into a new “geometric” engine. Until then, feel free to work in the collision debugger project, but keep in mind how transferable your code will be.

For weeks 2 through 4, you will need OBJ models loaded into your game from external files. Because most, if not all, of you have written a parser before, we have written a class that loads and renders an OBJ file. This can be found at `/course/cs1972/asgn/platformer`, along with some sample OBJ files and png’s to texture them with fake shadows.

Week 1 - Due Mar. 8

Platformer1 will take place entirely in the collision debugger. You’ll implement the basic sweep tests and visualize your results to ensure their correctness. You should not be touching the collision debugger support code here - the focus here should be on your collision code. It should work no matter what kind of graphical environment you’re working in.

Design Check

- What transformations do we apply to collide an ellipsoid with a triangle?

- Give analogies for the three cases required to collide a sphere with a triangle. For example, “sphere-plane collision is the same thing as [blank].”

- What are the two relevant pieces of information to return when an ellipse collides with a triangle?
Common Engine Requirements

Shape representations apply regardless of the system they’re being used in, so implementations of ellipsoid and triangle raycasting do not need to be strictly tied to your “geometric” engine.

Primary Requirements

- Analytic ellipsoid raycasting (ray-sphere intersection)
- Analytic triangle raycasting (ray-plane intersection)

Geometric Engine Requirements

Although you’re working in the collision debugger this week, be sure to write your code so that it will be easy to place into a “geometric” engine in the future.

Secondary Requirements

- Analytic ellipsoid-triangle collision detection
  - Ellipsoid-triangle collision
  - Ellipsoid-edge collision
  - Ellipsoid-vertex collision
  - All collision tests return the correct parametric t value
  - Collision routines return the contact point

Game Requirements

Though it isn’t much of a game, we do require that you turn in the collision debugger code with your working implementations.

Secondary Requirements

- Debugger draws the ellipsoid at its location at the time of collision

Week 2 - Due Mar. 15

Platformer2 completes the collision debugger, and then ports this code into the beginning of your “geometric” engine. By the end of this week, you’ll have an ellipsoid-shaped player moving around a world that consists of triangles.
Design Check

- What is the “collision response slide”, and why is it necessary for proper movement? Give a demonstration of a few iterations of the slide.

- What is the purpose of the hack you are implementing? At a high level, how is it implemented?

Common Engine Requirements

Like textures, OBJ models are engine-agnostic, so your OBJ code should be referenced from some convenient place like your graphics object.

Primary Requirements

- Able to load OBJ models using the support code
- Able to render OBJ models using the support code
- Stores only one copy of any single model (never load the same model twice)

Geometric Engine Requirements

By the end of this week, you’ll ween yourself off the collision debugger support code and have a “geometric” manager that handles a triangle-based world for you.

Primary Requirements

- All raycasting and collision detection requirements from Platformer1 are fulfilled
- Analytic collision response
  - Collision response slides for at least N iterations of the sweep test in the plane horizontal to collision (N = 4 is a good value)

Secondary Requirements

- Collision response implemented with at least one of the hacks described in lecture and is documented in your code

Game Requirements

Like minecraft2, platformer2 will mostly be a collision and rendering demo. Next week is when the real fun starts!
Primary Requirements

- Collision debugger support code is not present
- Movement and view controls must be the same as warmup and minecraft (including gravity and jumping)
- The camera must be able to use a 3rd-person view
- The player must be drawn as an ellipsoid

Secondary Requirements

- The game must have a menu screen that allows the player to choose between different levels (different OBJ models)
- The player must be able to return to the menu screen without restarting the program

Week 3 - Due Mar. 22

Platformer3 introduces pathfinding to your “geometric” engine, allowing NPC’s to navigate environments that are often very complicated. At this point, you should start thinking about what you want your platformer4 to look like, since next week will be entirely gameplay coding!

Design Check

- How will you determine what polygon in the mesh an entity is currently in?
- How will you determine the path of polygons from the entity to its goal?
- Give an example of the funnel algorithm in which the created path has at least one turn.

Geometric Engine Requirements

Navigation meshes provide a robust and interesting way to navigate through arbitrary terrain. We hope you’re ready to implement graph search again!

Primary Requirements

- Pathfinding and navigation across a loaded navigation mesh
  - Construct a graph from a navigation mesh loaded from an OBJ file
  - Query for the polygon in the mesh directly under a point
  - Search through the graph for a path between two points (feel free to use BFS)
  - Some form of path generation - ex. using the midpoint of portals
Secondary Requirements

- Path generation using the simple stupid funnel algorithm

Game Requirements

The funnel algorithm will likely keep you busy enough, but if you find yourself with extra time, feel free to start adding gameplay this week!

Primary Requirements

- The game must follow all primary requirements from platformer2
- The player must be able to toggle a visualization of navigation using a key input
  - The visualization must render the navigation mesh
  - The visualization must render the shortest path from the current player position to a target point
  - The target point must be visualized with an ellipsoid of the same size as the player (it does not have to collide)
  - The player must be able to change the target point by raycasting the environment (place the center at the result point plus the normal of the collided triangle)
  - The player must be able to change the target point by raycasting the target point (place the center at the result point)

Week 4 - Due Apr. 5

Platformer4 is your chance to take a week off from engine requirements (for the most part) and just make a game. We have some requirements to ensure that everyone makes an equally involved game, but be sure to get creative and make something unique!

There will be no design check this week! Have fun making your game!

Common Engine Requirements

The only engine requirement for this week is going to be the ability to render orthographic UI in your graphics object.

Primary Requirements

- Graphics object is able to render orthographic UI

Game Requirements

Looking forward to playing your games!
Primary Requirements

• The game must use orthographic projection to display some form of game UI
• The game must have at least one type of enemy that uses path-finding over a navigation mesh

Secondary Requirements

• This enemy must interact with the player in some way (e.g. push the player off ledges, shoot at the player, damage the player directly)
• 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 platformern 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.