CS5.502.M22 : Advanced Graphics, AR and VR

Instructor: Avinash Sharma
Release Date: August 3

Due Date: August 10 23:55
Assignment: 1

Note - You must use OpenGL(version 3+) that you can build using either OpenGL(C++/python) for OpenGL. **Be sure to read the submission instruction before you proceed with implementation**

Figure 1: Game

1 OpenGL

In this assignment, we will build a game. In doing so, we aim to get comfortable with making objects of greater complexity with OpenGL, working with many interacting components, and understanding how applications are built with OpenGL. In this task, you will be required to create a maze game derived from the Among Us game, in the form of a single-player variant. This game's objective is to exit a maze you have been dropped into after finishing two tasks. You must hurry because reactor meltdowns do not fix themselves.

- World: A procedural maze (different maze layout on every game startup). Maze needs to have a single exit, and if the player reaches this point after completing two tasks, the player can exit the maze. The walls of the maze are visible as lines on the screen (2D). And the ground plane is expected to have a mud texture.
- Player: A single player with some assigned health value can navigate the generated maze with the keyboard.

Figure 2: Character

- Light: The lighting of the game can be turned on or off using a key press. When switched off, you should have a small light source illuminating the area around you, which allows you to see a small part of the world around you. Bear in mind that the light cannot penetrate the maze walls, and the rest of the maze should remain in darkness. The number of seconds spent in dark mode should boost the points of the player.
- HUD (Heads Up Display): A box of rendered text on the top left of the screen, present at all times, detailing the following:
  - Health: This indicates the player's points, which can increase or decrease based on events/interactions in a manner defined by you.
  - Tasks: Tasks completed / Total number of tasks available.
  - Light: On/ Off indicated the status of the central lighting system.
  - Time: A countdown from some X number of seconds. When the clock runs out, end the game.
- Tasks
  - An imposter character (enemy) exists at a random location in the maze and begins to get to your position as soon as the game begins. A button (a circle drawn in the maze path) to vaporize (make the imposter disappear) the imposter exists at another random location in the maze. The task here is to reach the button before the imposter reaches you. Use a pathfinding algorithm to make the imposter reach you as you move around. Coming in contact with the imposter ends the game.
  - A button to release power-ups and obstacles exists in a random part of the maze. The task is to reach the button and then collect the power-ups to increase your score. You can represent power-ups/obstacles using geometric 2D shapes. Coming in contact with an obstacle decreases your score.

2 Ray Tracer

You are expected to implement a ray tracer from scratch. You can follow the detailed guide available in 1, 2. For this task, everything needs to be done with the use of basic C++/Python without using a fully developed renderer like Blender's Cycles/mitsuba, etc.

- Scene: A Cornel Box-like structure with red color on the left wall, and green on the right wall. A plane light source(area light) on the roof. Wall facing in user's camera and rest of the roof is of {128, 128, 128} in RGB (in uint8 scale).
- Box: Consists of a pyramid, a cuboid, and a sphere.
- Tasks: You are expected to document the results of all the subtasks.
  (A) Trace rays into the scene. Extract Normals in the view.
  (B) Increase SPP of note the results
  (C) Make the ground plane made of checkerboard pattern. And record results.
  (D) Make all the objects in the scene made of diffuse material and take a snap chat of the scene.
  (E) Make the pyramid metallic, while the sphere is transparent. Capture results at high spp values and low spp, comparing the timings and quality of both the results. (suitable assumptions made should be explained for the choice of materials chosen)
  (F) Implement BVH and recording timings of the scene.

3 Submission Instructions

Please zip your main directory after removing the build directory and rename it as *roll_number.zip* and upload the zip. Keep your code separate from the boilerplate you are using. While submitting keep them in two separate folders(Submission structure is given below). We will check for plagiarism on the files containing your logic. Submit the entire project directory with the dependencies and not some random individual files. Please contact any of the TAs well before the deadline if you have any doubts about this. Submission Structure for OpenGL:

```
roll_number.zip
├── dependencies (if any)
├── boilerplate
└── your_logic_files
    ├── logic_file1
    └── logic_file2
```

There is no specific submission structure for the raytracer as you are expected to code everything from scratch.