CMP203 Coursework Report
OceanView Hotel

André David Gomes de Moura
1900301
https://github.com/andredavidm

Intro

I chose to create a scene from the video game G.T.A. Vice City released in 2002. The chosen scene is the starter room or first house which is based on a hotel room from the late '80s in Miami with a view to the beach. In the game, this room is called Ocean View Hotel and it's filled with furniture and patterns resembling those from the time the game is set in. The scene contains the use of procedurally generated primitives, stencil buffer, planar shadows, global lights and spotlights, textures, interactions with the environment and camera, and a wireframe-mode.

Why I picked the OceanView Hotel from GTA VICE CITY:

The reasons behind this pick were the following:

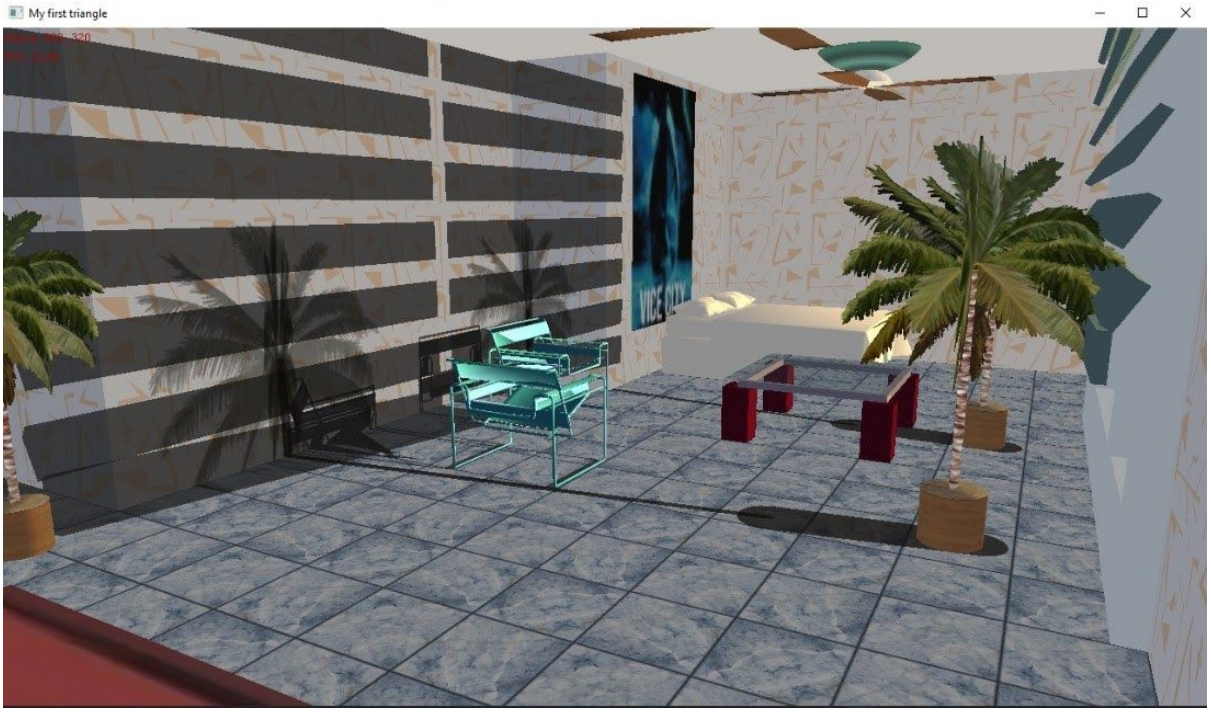
- The scene from the room contains almost every provided task or objective for the coursework
- Having a clear final objective and boundaries for a project forces problem-solving.
- On a personal level, I find working on something nostalgic to be a great motivator as well.

Controls

The controls implemented are the following:

- W - Move forward.
- S - Move back.
- A - Strafe left.
- D - Strafe right.
- Q - Free camera/Lock camera.
- O - Speed up sun orbit.
- P - Cycle through Wireframe modes.

Screenshots




My first triangle
Mouse: 600, 320
FPS: 41.42
FOLLOW THE DAMN
TRAIN C.J.!


My first triangle
Mouse: 600, 320
FPS: 31.90

Project Structure:

The project is structured in 3 different types of functions and classes. The first type of class is the primitives or basic classes that are tasked with generating basic polygons, loading textures or models and processing shadows and lights. The second type is the assembly classes, these classes arrange primitives into objects and place them in the scene or use the basic classes to create something with it. The last type is the scene class, which puts everything together in the main scene, updates variables and takes care of input.

Some of the notable classes are:

ProceduralGenerator - This class generates all of the basic geometry to later be used, it is simply structured, each function takes in variables and then generates a plane, a cube or cuboid, a cylinder a sphere etc...
The drawFloorTile() and drawWall() function both work a bit differently from the rest. These functions generate walls based on a starting point and endpoint, and a floor and roof based on a small map array.

ObjectMaker - Most of the objects in the scene built using the ProceduralGenerator functions will be found here.

RoomMaker - RoomMaker takes care of placing objects in the right position using a series of Matrix Pushes and Pops. It also uses the walls and floor functions to draw the roo, updates objects and prepares object shadows.

ModelLoader - Similar to the ObjectMaker function, this function simply prepares the objects to be used in the scene, only it loads them externally has .obj models instead of using the ProceduralGenerator class.

Lights - Lights class contains all the lights in the scene, which are 2 spotlights inside the room coming from the fans, and a global one outside of the room representing the sun, which will give the effect of the sunrise with the shadows. This class also calculates the sun "orbit" position around the scene.

Features to Highlight:

PNG:

One of the first features implemented in the scene were the trees. The trees are made using 3 ProceduralGenerator functions: the cylinder, the circle, and the plane for the billboarding effect.
The openGL functions used for the billboarding are:

```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, 0.1f);
```

This enables a test on the texture that will ignore any value with alpha channel inferior to 0.1.

Shadows:

Shadows functions receive the models and flatten them against the walls and floor which are then stencilled using the stencil test on the walls and floor. Because the sun is moving around the scene, the light position for the shadows needed to be calculated and updated based on the sun position. This prevented me to use a simple glRotatef() on the sun to orbit around the scene, therefore I would have to calculate the orbit with a function, this function is located on lights and is called sunUpdate(). Then, the sun position will be communicated to the shadows which will then update the movement of the shadows giving the effect of the sunrise. The shadows will also fade as the sun approaches evening time, this is also calculated using the sun position, also found in sunUpdate().

Obstacles and final thoughts:

One of the obstacles found right away were the loading times for the textures. I used external software like photoshop to reduce the size of the textures, but, another idea was to separate the textures into multiple loading functions for each class, this way the solution won't load textures twice.

Another obstacle was the fact that the models chosen had too many polygons or were not triangulated correctly. I ended up using Blender to lower the number of polygons and export the models triangulated, this way I could use only one vertex array render function for all models.

Debugging occupied 80% of the time working on the project. Every time something new was added, nothing worked properly. I used many different types of debugging processes, my favourite for this project was one where I assumed the error was happening for a determined reason, and then try to remove that specific error by removing some part of the code and test to see if something changed. This approach ended up solving almost every error I encountered and helped me better understand how OpenGL worked.
because I started writing the code very early and in the middle of development, my “methods” of writing classes evolved significantly from the very first line I wrote.

References:

Knoll Wassily Chairs - by facequad - https://www.cgtrader.com/free-3d-models/furniture/chair/knoll-wassily-chair-01
Bed - by annam-dayakar - https://www.cgtrader.com/free-3d-models/interior/bedroom/bed-interior-design