COMP 371: Computer Graphics

Final Group Project

Team 6

Name	Student ID
Yefei Xue	26433979
Ivan Ilyushchenko	25985722
Edip Tac	26783287

Table of Contents

1. Introduction

The final project assigned was to design a 3D interactive software that allows the furnishing of an apartment. Our goal with this project was to implement the software using C++ with the OpenGL API to create the visuals associated with the application. The program creates a scene where a 5 room apartment is furnished, and the user is able to manipulate the environment by relocating the furniture, toggling the lighting effects of the scene, and moving about the entire apartment.

2. Background

2.1. Project Setup

The project is a furnishing program implemented using interactive graphics. Our team uses Visual Studio 2015 community platform to develop all C++ header files and source files. In the application, there is a textured apartment containing a floor, ceiling, walls, doors, and windows, and makes up a 5 room setup. Each room contains minimum 5 furniture items and 1 light source. The entire apartment is placed within a Skybox setting and allows the external world to be seen through the windows of the apartment. As can be seen in fig. 1, the program starts inside the living room of the apartment.

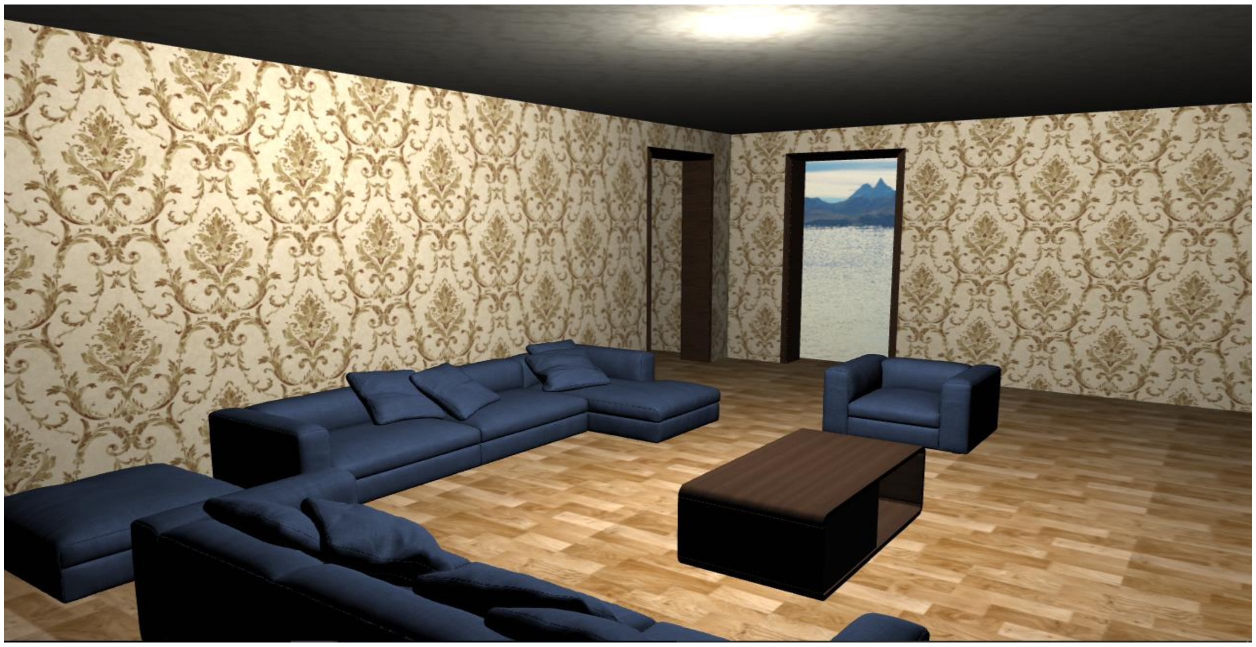


Figure 1: Starting scene of the project

2.2. User Manual

The controls used within the project make use of the keyboard as well as the mouse, for navigating within the house and manipulating the objects present. Table 1 describes the keys used for the project along with their respective purposes.

Table 1: Controls used for navigation and manipulation of objects

Key	**Resulting Control**
W	Move forward within the world
S	Move backward within the world
A	Move left within the world
D	Move right within the world
Mouse move up and down (without clicking)	Camera pitch control
Mouse move left and right (without clicking)	Camera yaw control
Mouse left-click (on object)	Selects object within the world
Mouse movement left and right (object selected)	Moving object with respect to change in yaw of camera
W (with object selected)	Moving camera and object forward
S (with object selected)	Moving camera and object backwards
A (with object selected)	Moving camera and object left
D (with object selected)	Moving camera and object right
F	Toggles flashlight (attached to camera)
1	Toggles light in the living room
2	Toggles light in the bathroom (room with toilet)
4	Toggles light in the kitchen
5	Toggles light in bedroom 1 (opposite of kitchen)
6	Toggles light in bedroom 2 (opposite of bathroom)
M	Toggles cursor ON and OFF(used for window resizing)

2.3. Student contributions

All of the team members contributed to all sections of the programming process and the entire project was done with strong collaboration. This includes, but is not limited to, object modelling, object loading, algorithm creations, debugging, and writing the report. Since the project was to be completed at a very short time-frame, teamwork was a crucial factor in getting all the requirements of this project complete.

3. Design and Implementation

3.1. Game Modelling and Libraries used

The objects and the house within the program were modelled using 3DS Max. The house itself, along with the furniture present inside it, were modelled and textured within 3DS Max using several external sources to obtain the correct textures for each object [1-4]. The SkyBox implemented as the external world was taken from the tutorial provided for the COMP 371 course, and manipulated to fit our requirements for this project.

The completed objects along with their textures were then imported into C++ and the OpenGL AP,I using Assimp[5]. This was chosen due to the ease it gives the end user for importing dozens of different model files by using a generalized data structure within Assimp, which is able to load all the data of the models into them. Additionally, the ease of access to all the data within the Assimp data structures, along with the lack of specificity for file formats added, allows the end users to load their models with ease and makes it very easy to add future work onto this project. Thus, Assimp was chosen for loading all the objects in this project. However, the model and the mesh classes associated to Assimp were manipulated to best fit the criteria of this project.

Once all the scenes were rendered, the object selection was done by following the theory of the *Picking with an OpenGL hack* as created and implemented by the tutorial at opengl-tutorial.com[6]. This theory was chosen due to the ease it provides the user in selecting objects, and since the program developed is within a fixed scene size, performance drops due to this algorithm were negligible. The main concept behind this theory is that the scene is rendered as usual, but each mesh is replaced by a unique and specific color. This color is generated by giving each object within the world space a unique ID and converting that ID into the color of each object. This makes selecting every object in the world simple since each object that is selected will be returning a unique ID that is specific to that object only. Since the assigning of unique colors does cause issues with the final scene that is created, a double buffer setup was used, where the first buffer contained the unique ID of each object which was used for selecting the objects themselves and manipulating them within the world, and the second buffer was used to display the actual textures to the scene and create a visually pleasing scene.

To perform collision detection tests between the camera and the scene, i.e. objects and the walls, as well as collision detection between the objects themselves, the Axis Aligned Bounding Box (AABB) method was used. The theory was adapted from the algorithms described by Dunn and Parberry[7].This method of bounding box was selected since the majority of the objects within the world are cuboid shaped objects, and the AABB method would be a tight fit around the world objects and is also one of the least complicated bounding volume methods available.

To perform the collision detection test between the camera and the scene, we used a 2D analysis instead of the AABB method. In our method, we used 2 vertices at a time for each triangle within the scene and we go through each triangle 3 times (to analyze each edge) in order to check every pair of vertices at a time. As indicated in figure 2, during each iteration of edge qualification, the method checks the 2 vertices belonging to an edge, and determines the angle between the edge and the camera. If the angle is greater than 90 degrees, there is no collision occurring and the camera is towards the left of the left-vertex. Additional to this, the algorithm also tests to see if the distance between the 2 vertices is less than the distance between vertex1 and the projection of the camera to the wall, which is cos(θ) * line alpha. If either of these criteria are met, collision is not present. If both of these conditions fail, then collision could be triggered. To further confirm whether collision occurs, we also test for the distance between the camera and the Edge produced between Vertex1 and Vertex2 is less than assigned limit. The limit is chosen as the distance from the camera to it's near plane + a constant value. This algorithm is best described in the code snippet below in figure 3.

Figure 2: Representation of camera collision detection system

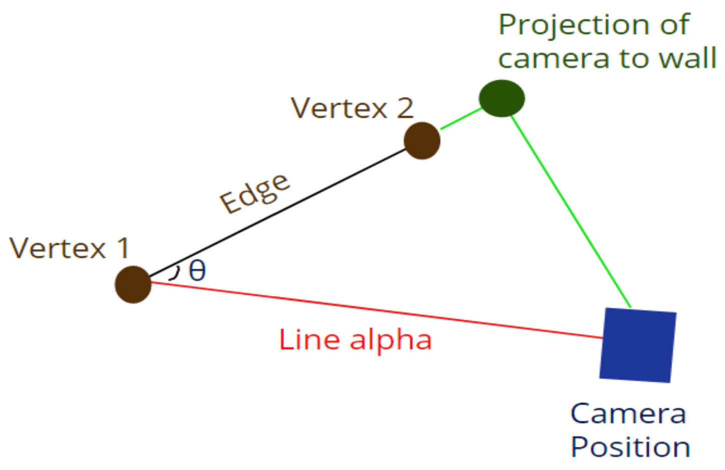
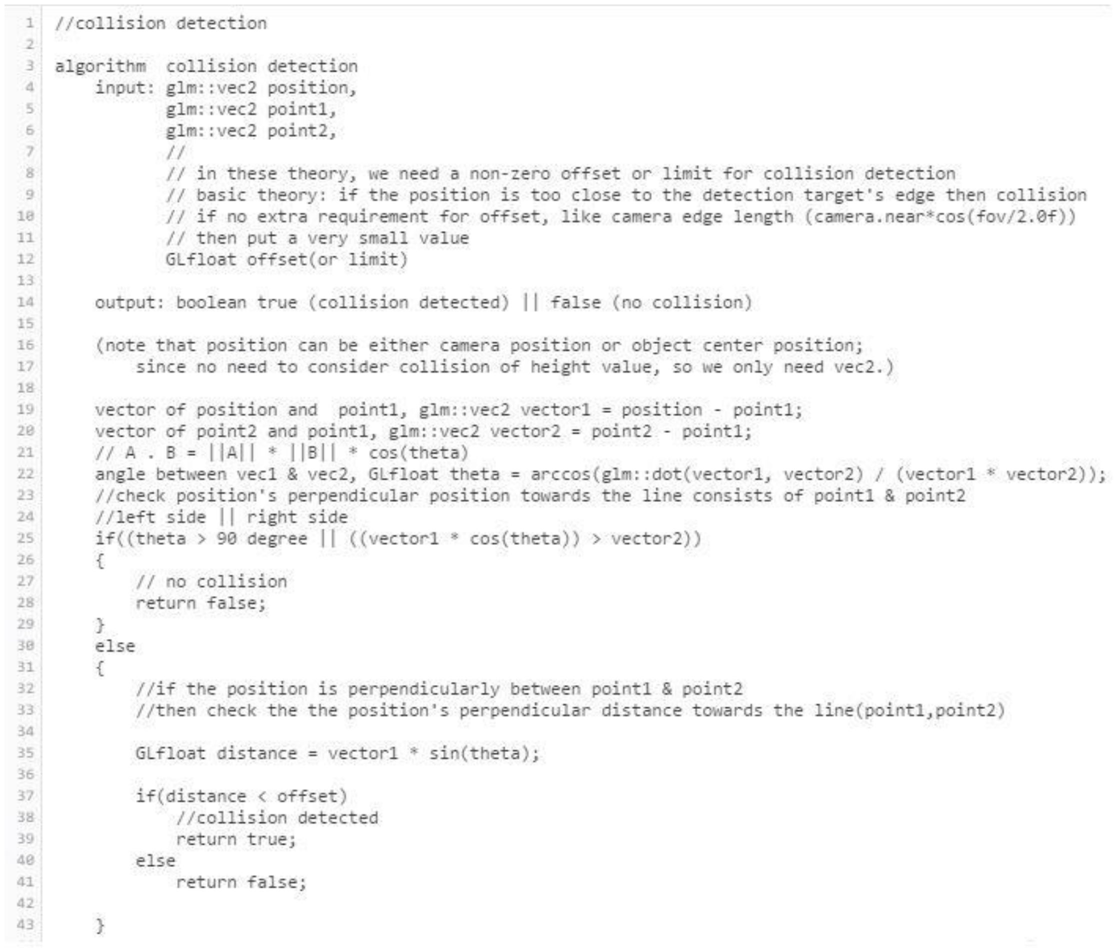


Figure 3: Collision detection algorithm created for the collision detection system.



```
//collision detection

algorithm  collision detection
    input: glm::vec2 position,
           glm::vec2 point1,
           glm::vec2 point2,
           //
           // in these theory, we need a non-zero offset or limit for collision detection
           // basic theory: if the position is too close to the detection target's edge then collision
           // if no extra requirement for offset, like camera edge length (camera.near*cos(fov/2.0f))
           // then put a very small value
           GLfloat offset(or limit)

    output: boolean true (collision detected) || false (no collision)

    (note that position can be either camera position or object center position;
        since no need to consider collision of height value, so we only need vec2.)

    vector of position and  point1, glm::vec2 vector1 = position - point1;
    vector of point2 and point1, glm::vec2 vector2 = point2 - point1;
    // A . B = ||A|| * ||B|| * cos(theta)
    angle between vec1 & vec2, GLfloat theta = arccos(glm::dot(vector1, vector2) / (vector1 * vector2));
    //check position's perpendicular position towards the line consists of point1 & point2
    //left side || right side
    if((theta > 90 degree || ((vector1 * cos(theta)) > vector2))
    {
        // no collision
        return false;
    }
    else
    {
        //if the position is perpendicularly between point1 & point2
        //then check the the position's perpendicular distance towards the line(point1,point2)

        GLfloat distance = vector1 * sin(theta);

        if(distance < offset)
            //collision detected
            return true;
        else
            return false;

    }
```

3.2. Challenges faced during implementation

The biggest challenge during the entire development cycle of this project was time. The limited time available to create this project during the summer session was a big influence on the quality as well as the quantity of work present within this project. Additionally, implementing the collision detection system proved to be difficult as optimization of the collision detection was an extremely time consuming process, and the lack of debugging tools in the OpenGL API made it very difficult to detect bugs within the code. Furthermore, the collision system was detected on a much older system with low memory, and the drop in frames per second of the program in those systems had a possibility of causing the collision detection method to bug and not detect collision since the object movement is associated with that of the camera, which is directly associated to the system time via glfwGetTime().This was handled by adding a limit to the movement velocity of the camera, but under severe system stress, there is a possibility of this bug still occurring.

The lack of debugger also prevented the implementation of a shadow algorithm as changes made to the program to implement shadows resulted in code breaking bugs. Lastly, the object selection and the object movement with the camera controls was the final difficulty faced during the development

4. Conclusion

Our implementation of the project generates an apartment with 5 rooms, with each room containing an individually controllable light present within it. The room are divided into a living room furnished with couches and a coffee table, a kitchen furnished with a toilet, a kitchen furnished with a stove, refrigerator, and dining table and chairs, and lastly 2 bedrooms furnished with beds, night stands, and study tables and chairs.

The project was completed successfully with all of the requirements of the project being implemented, as well as the additional shadow. For future improvements, a larger apartment could be taken into consideration, with windows and doors within the apartment that contain open and close animations to increase immersion and add a more realistic feel to the scene. Due to the time constraints, fine adjustments to the shadows were also not implemented and the code currently produces jagged shadows. A percentage-closer filter could be implemented on top of the shadow maps created in order to generate much smoother shadows into the scene.

5. Bibliography

[1] “3ds Max | 3D Modeling, Animation & Rendering Software | Autodesk.” [Online]. Available: https://www.autodesk.ca/en/products/3ds-max/overview. [Accessed: 19-Jun-2017].
[2] “3d model home furnishings,3d furniture models free download,” all3dfree.net. [Online]. Available: http://www.all3dfree.net/3d-models-furniture.html. [Accessed: 20-Jun-2017].
[3] “Furniture 3D Models | Download 3D Furniture files | CGTrader.com.” [Online]. Available: https://www.cgtrader.com/3d-models/furniture. [Accessed: 20-Jun-2017].
[4] “Furniture 3D Models - Free 3D Furniture download.” [Online]. Available: https://free3d.com/3d-models/furniture. [Accessed: 20-Jun-2017].
[5] “Open Asset Import Library.” [Online]. Available: http://assimp.sourceforge.net/. [Accessed: 20-Jun-2017].
[6] “Picking with an OpenGL hack.” [Online]. Available: http://www.opengl-tutorial.org/miscellaneous/clicking-on-objects/picking-with-an-opengl-hack/. [Accessed: 19-Jun-2017].
[7] F. Dunn and I. Parberry, 3D math primer for graphics and game development. Plano, Tex: Wordware Pub, 2002.
[8] J. De Vries, “Learn OpenGL.” [Online]. Available: https://learnopengl.com/#!Introduction. [Accessed: 20-Jun-2017].