Scenic Snooker

Ayush Kumar(170195), Atul Kumar Pandey(150164)

November 2019

1 Introduction

Snooker is a cue game originated in India around second half of 19^{th} century. It is played on a table with six pockets in it along the rails. There are many different flavors of the game with each one famous with its own specific name. Some of the prominent ones are 8-ball pool, Black-ball, 9-ball, and blank ball. We have tried to make a 3-D version of 8-ball pool. All the graphics rendering and game play have been designed by us. The game provides a real-like experience of pool game with support for view change, rolling and collision among many other. The game can be played anywhere be it near mountains or grassland. We call it scenic snooker because of the feature it provides. You get to choose form many of the environments where you want to play and it creates real time rendering of sky box and the game play.

Figure 1: 8 Ball pool

2 Implementation

The implementation of the game has 4 major steps. Each of them have been discussed below in order.

2.1 Rendering pool table

We built the model of a pool table with all its intricacies in blender. Then we exported this model into OpenGL. To load the model into modern OpenGL, we wrote an object loader. The loader takes a wavefront file (.obj) as input and loads the geometry data for vertices, normals, texture coordinates and materials. Balls in the game are shiny than that of the table surface. To keep this into account the loader is designed to parse the material property to OpenGL which uses this to illuminates the game. The figure 2 shows the pool it has been loaded into the OpenGL. Along with the table we also show the coordinate system for user's convenience to have an idea of upright position.

We further apply textures on the table. The textures are in accordance with

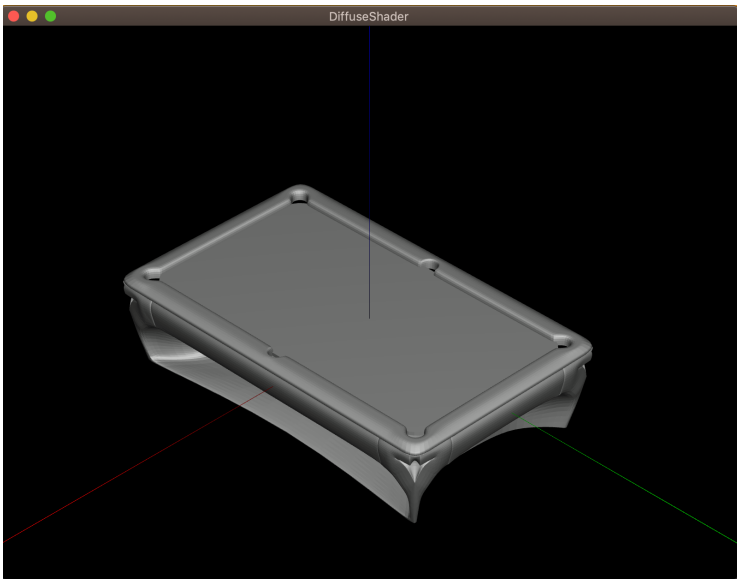

Figure 2: Pool table after basic rendering

standard game colours. The details regarding shiny and rough surfaces have also been taken care of to give a realistic look to the table. Figure 3 shows the pool table after texture mapping.

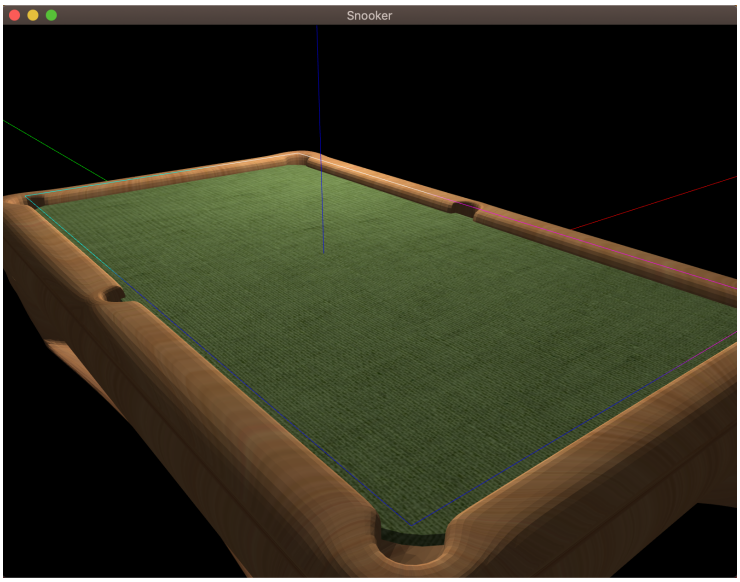

Figure 3: Pool table after applying texture

2.2 Rendering the ball and the cue stick

We are modelling each object on the table as a rigid body. The balls have been modelled as rigid spheres and can undergo collision among themselves, with the walls and the cue stick. Rolling enhances the experience of the game by giving the player the ability to use spin to play better shots.
The player uses the cue stick to apply force and torque on the cue. Cue has also been modelled as rigid body and can be aimed around the board using the mouse. The force which we hit and the place where it hits the ball, determines the initial velocity and angular velocity of the cue. The damping of linear and angular velocities has been achieved by carefully choosing friction coefficients on the ball and table surfaces. The table has also been modelled as rigid body with very low frictional coefficient on the upper surface.

2.3 Collision detection and frame update

Collision detection is very important for the pool to work properly. There are three different kinds of collisions we need to take care of. i) Collision between ball and the stick ii) collision between ball and ball and iii) collision between the ball and the walls of the table. We use the library ReactPhysics3D for detecting collisions and making updates for the objects in each frame. Every object is updated each frame using the laws on physics. Doing this is computationally heavy but small number of balls make it possible. We are using a feature called sleeping. We only update the frame when there is at least one object on the table which is not sleeping (having a non-zero velocity or a non-zero angular

spin). This significantly improves the performance specially for the part of the game when we are planning to take the shot. Section 2.4 explains the library in detail.

2.4 ReactPhysics3D

ReactPhysics3D is c++ physics engine library which helps in 3D simulations and games. It works by defining a world. You instantiate different rigid body object in it and define their properties like mass, friction etc. When force is applied on the bodies, their trajectories of motion are calculated using rigid body dynamics. The library has support for discrete collision detection, collision response and friction, multiple collection shapes and many more. The best part is that it has no external dependency on any other library and is well documented.

2.5 Cube mapping

The environment of the game is designed using cube mapping. It is a method of environment mapping in which the environment is projected onto 6 faces of the cube. This cube is wrapped around the camera and rays from the camera are shot onto the cube faces. Point at distant environment is assigned the pixel corresponding to the point where the ray intersect the cube. This gives the player the illusion that he/she is inside an infinite environment. The final game scene at the start of the game in a football field as is shown in 4

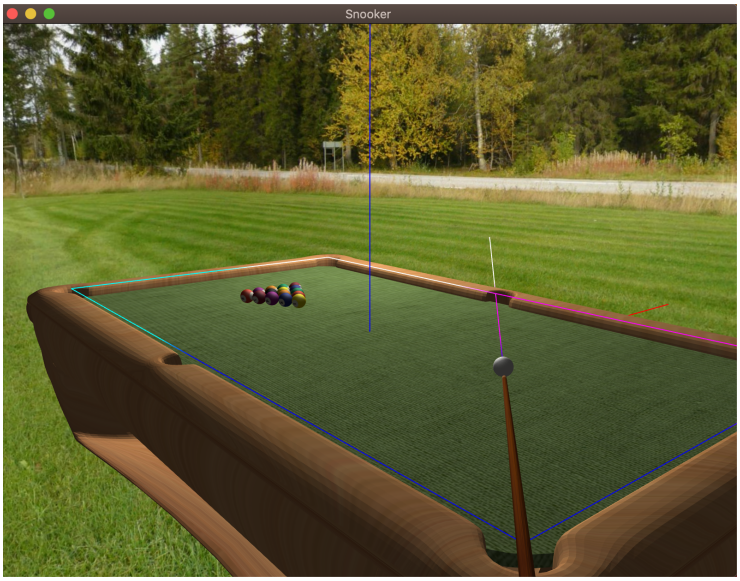

Figure 4: Final pool table before start of a game

3 Game Play

This is a two player game, rules of which are as follows.

1. Each player gets turn alternatively. The Player can move camera along the rails on a specified trajectory and can also change the point of view of the board.

2. Camera movement can be controlled using keys as well as mouse. Besides translation and rotation, camera supports yaw and pitch also.

3. The cue stick always points to the cue and can be aimed by moving the camera. It shows a line to aim originating from the cue.

4. Once the player has aimed, the player locks the camera and uses the mouse to adjust power of the shot.

5. After the mouse is released, the motion of the balls is simulated using ReactPhysics3d physics simulation engine.

6. Currently we have implemented a variant of 8-ball pool scoring rules. The player gets the point corresponding to the ball its pockets. The player with highest points ends in the end.

Some scenes of the game play are shown in fig.5 6 and 7

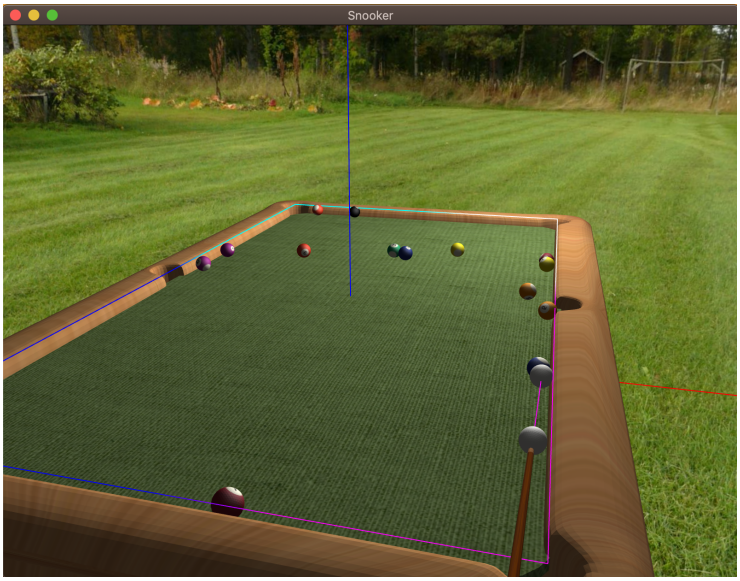

Figure 5: A scene from the game: 1

Figure 6: A scene from the game: 2

Figure 7: A scene from the game: 3

4 Frameworks & Libraries

- Modern OpenGL (3.3 or higher)
- Some of the OpenGL functions are not readily available. To query and load them at runtime, we are using OpenGL extension GLEW for this.
- To manage windows and process user input, a windowing library GLFW has been used.
- To perform master matrix calculations, we have used glm.
- Physics simulation library: ReactPhysics3D