Pinball

CS 480 - PA#10 Report

Jordan Andrieu, Deev Patel & Braeden Richards

Friday, November 16th, 2018

Contents

Overview

This project is a simple, simulated version of the classic game of pinball. The user is given 3 lives to score as many points as possible. The user can score 100 points for hitting one of the three cylindrical bumpers and 25 points for hitting one of the four rectangular bumpers at the top. The number of lives and points can be seen in both the terminal and the IMGUI menu window. The ball can be launched into the board with the "b" key. See the *User Manual* for more operation details. The project meets all the basic requirements for PA10. Notably, it incorporates modeling, lighting and collisions into a usable game.

Extra Credit

In addition to the basic requirements, the project implements the following extra credit items.

- Spotlight that follows ball
    - In addition to the two static spotlights, there is a third spotlight that follows the ball around. Its intensity/height can be adjusted via the z/x keys.
- Top 10 Scoreboard
    - In the IMGUI menu, there is a scoreboard that keeps track of the top 10 scorers. A user can join the scoreboard by scoring the necessary amount of points and entering a name into the terminal when prompted at the end of the game. When the game is started again, the user will be on the top 10 scoreboard.

Lighting & Shaders

This project incorporates lighting to give a more realistic look and feel to the game. Globally, the game has an ambient light level that can be changed by the user using the +/- keys at runtime. Locally, each object in the game has specific diffuse and specular characteristics that define how the light affects them. These settings are defined in the configuration file that is loaded at the start of the game. The diffuse and specular characteristics of the ball can be changed during runtime using the m/l keys as described in the *User Manual*

In addition to the adjustable ambient light, there are three other sources of light. All three are spotlights. Two are fixed with respect to the board and one moves with the ball. The spotlight that follows the ball has an adjustable height that can be changed with the z/x keys.

In terms of implementation, the project provides two different ways of calculating the effect of light. The default approach is a per-fragment lighting model that more or less conforms to the Phong reflection model proposed by Bui Tuong Phong. This approach interpolates the normals and calculates intensities at each pixel value. Thus, the calculations are done in the fragment shader. A less computationally expensive model can also be employed by pressing the *v* key. This switches the shaders in the program to a per-vertex model that conforms to the Gourand model proposed by Henri Gouraud. Here, the intensities are calculated at each vertex and interpolated to each pixel. So, the calculations are less expensive and done in the vertex shader. This model, while faster, does not give the same realistic feel as the per-fragment lighting model. The differences can be seen in Figure 2, located in the screenshots section of the *User Manual.*

Dependencies & Building

This project is built using cmake. To build and run the program in its basic form, go to the project source directory (PA10) and run the following.

```
mkdir build
cd build
cmake ..
make
./Pinball
```

One can also the build/run the project with different settings as shown below. Note that all the parameters/variables are shown with their default values. Only change them if you know what you are doing.

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DUSE_COMPLEX_BOARD_MESH=On
make
./Pinball -l launch/DefaultConfig.txt
```

The project was made with OpenGL 3.3, but it should work with most newer versions. In addition, the following external libraries are required. Note that everything was built and tested on Ubuntu 18.04. As such, all installation instructions are meant for Ubuntu 18.04.

- **Assimp** - Open Asset Import Library
    - More Info: https://github.com/assimp/assimp/wiki
    - Installation: `sudo apt-get install libassimp-dev`
    - Primary Usage: loading models of all the objects

- **Bullet**- Bullet Physics SDK
    - More Info: https://github.com/bulletphysics/bullet3
    - Installation: `sudo apt-get libbullet-dev`
    - Primary Usage: detecting collisions involving the ball
- **GLEW** - OpenGL Extension Wrangler Library
    - More Info: http://glew.sourceforge.net/
    - Installation: `sudo apt-get install libglew-dev`
    - Primary Usage: Implementing various part of OpenGL graphics pipeline
- **GLM** - OpenGL Mathematics Library
    - More Info: http://glm.g-truc.net/0.9.7/index.html
    - Installation: `sudo apt-get install libglm-dev`
    - Primary Usage: math related with rendering and movement
- **Magick++** - ImageMagick C++ API
    - More Info: http://www.imagemagick.org/Magick%2B%2B/
    - Installation: `sudo apt-get install libmagick++-dev`
    - Primary Usage: loading textures for models
- **SDL2** - Simple DirectMedia Layer
    - More Info: https://wiki.libsdl.org/FrontPage
    - Installation: `sudo apt-get install libsdl2-dev`
    - Primary Usage: window creation & user interactions

Note: One can install all the necessary dependencies at once.

```
sudo apt-get install libassimp-dev libbullet-dev libglew-dev
libglm-dev libmagick++-dev libsdl2-dev
```

User Manual

Keyboard Controls

- Game Controls
    - b: Launch ball and begin game
    - g/h: Move left/right bumpers
    - r: Reset everything - (in the rare instance of something getting stuck)
- Camera Movement
    - left_arrow/right_arrow: Move along the x-axis
    - up_arrow/down_arrow: Move along the z-axis
    - i/o: Zoom toward/away from the focus point
- Lighting
    - Global
        - f: Change to fragment lighting - Phong
        - v: Change to vertex lighting - Gourand
        - +/-: Adjust ambient lighting
    - For Ball Only
        - z/x: Move spotlight up/down with respect to ball
        - m/l: Adjust diffuse lighting of ball only
        - shift + m/l: Adjust specular lighting of ball only
- Other
    - t: Toggle (open/close) IMGUI menu window
    - ESC: Properly close all windows and exit program

IMGUI Menu

There is an IMGUI (https://github.com/ocornut/imgui) menu that runs in a separate window from the pinball window. At the top, it shows the current camera position, and eye focus. The user has the option to change these values by tying in any number desired. Note that one must press the update button for the new position and focus locations to go into effect. The middle of the menu shows the number lives left and the current score achieved. This is helpful for the user to keep track of where they are in the game and if they are in danger of losing. Finally, at the bottom of the menu is a top 10 scoreboard that shows the 10 highest scores ever achieved for the game. The menu can be opened and closed via the *t* key. An example of what the menu looks like can be seen in Figure 3 in the screenshots section.

Terminal & Scoring

The terminal shell from which the game was launched updates periodically with the number of lives left. Then, once the game is over (all lives are lost), the user's score for the entire game is printed out. Then, the user is prompted to enter a nickname. If the user's score is high enough, the entered username is added to the top 10 scoreboard. A user added to the scoreboard will be visible on any future runs of the program. Figure 4 in the screenshots sections shows the terminal output/layout upon the end of a game session.

Screenshots of Game

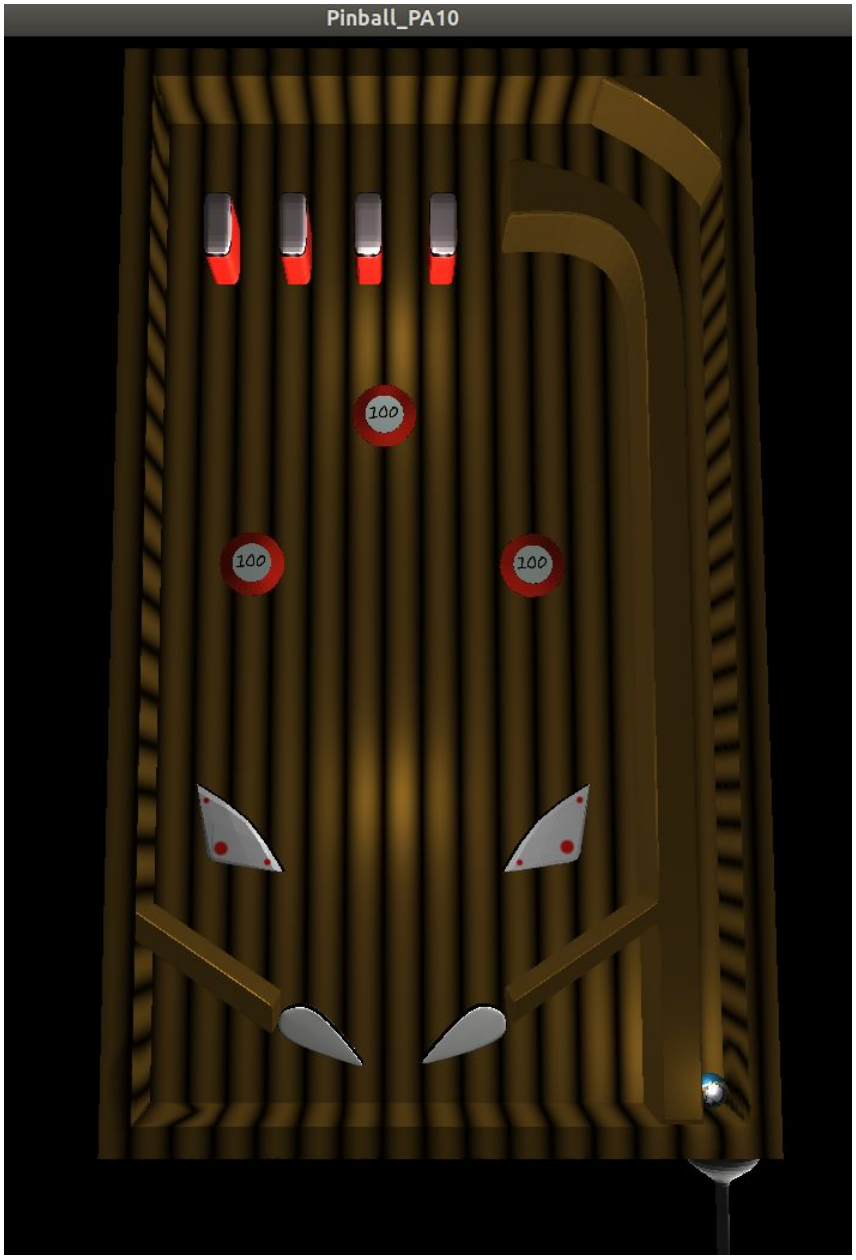

Figure 1: The game layout at startup. One can see the two static spotlights along the center of the board.

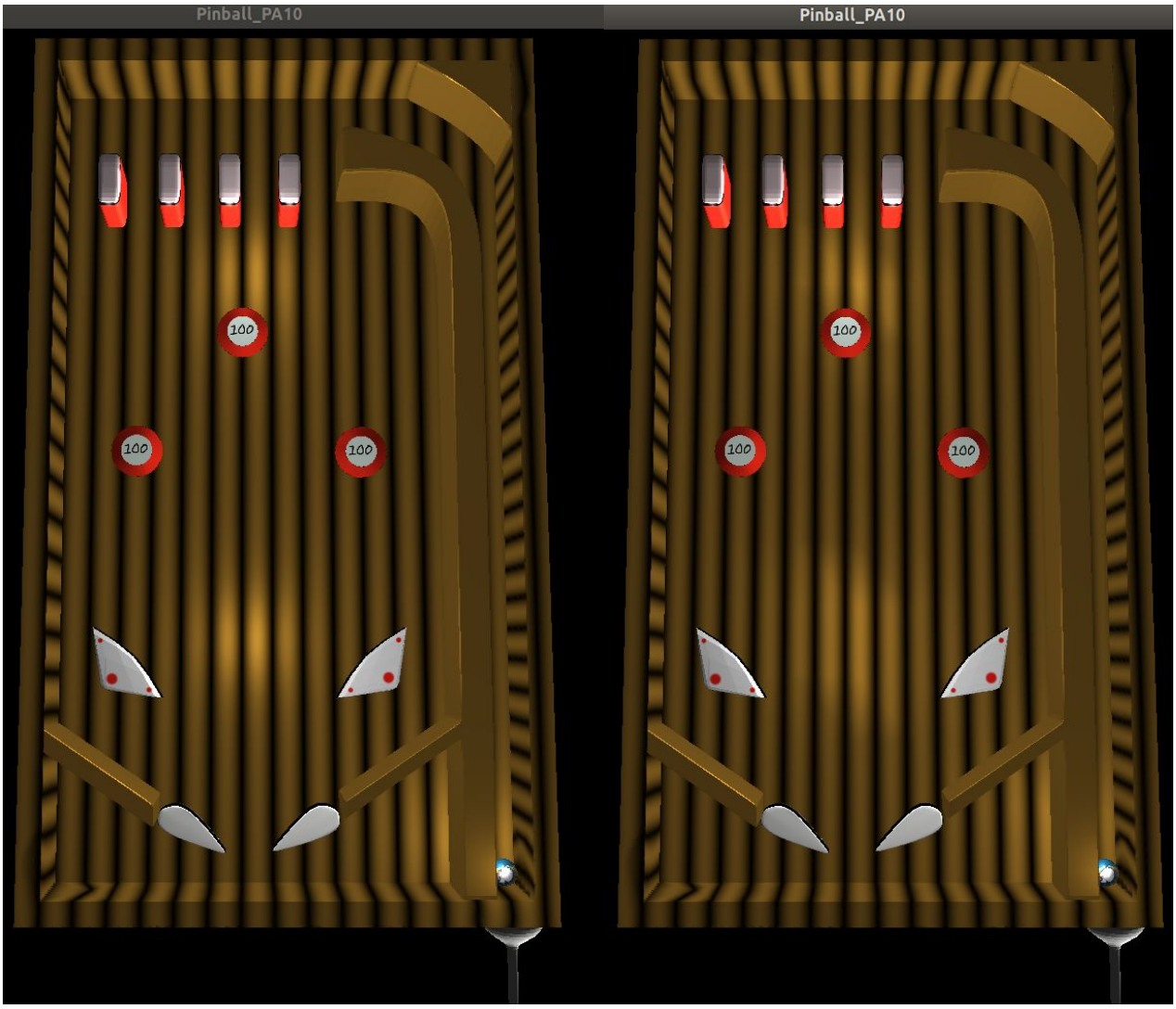

Figure 2: Comparison of applying per-fragment (left) and per-vertex (right) lighting in the game.

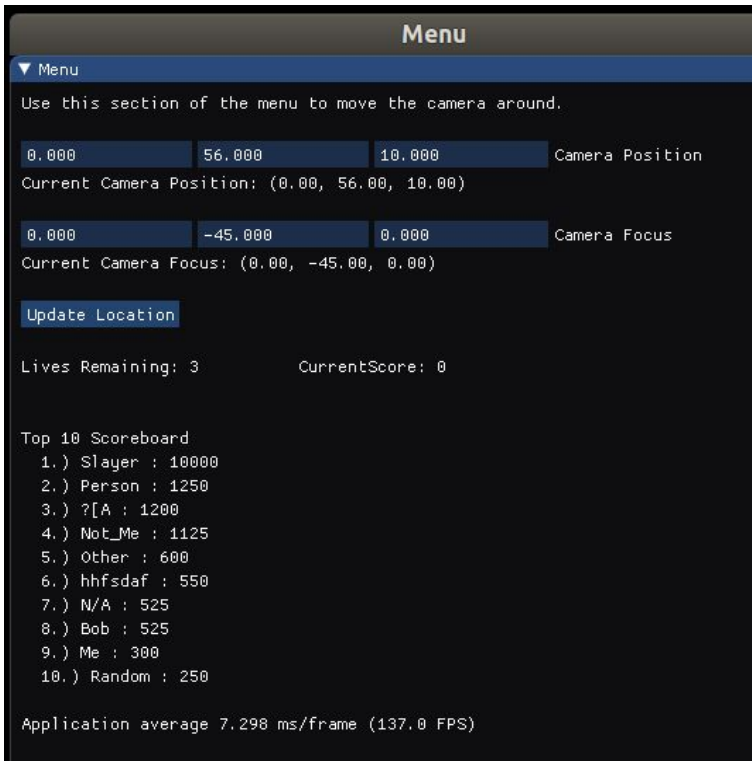

Figure 3: The Default IMGUI menu.

```
Lives Remaining: 0

Game Over: You Scored: 475 points!
Please Center your name: John

You made it into the top 10. You are now: 9
```

Figure 4: The output of the terminal window when the game is over.

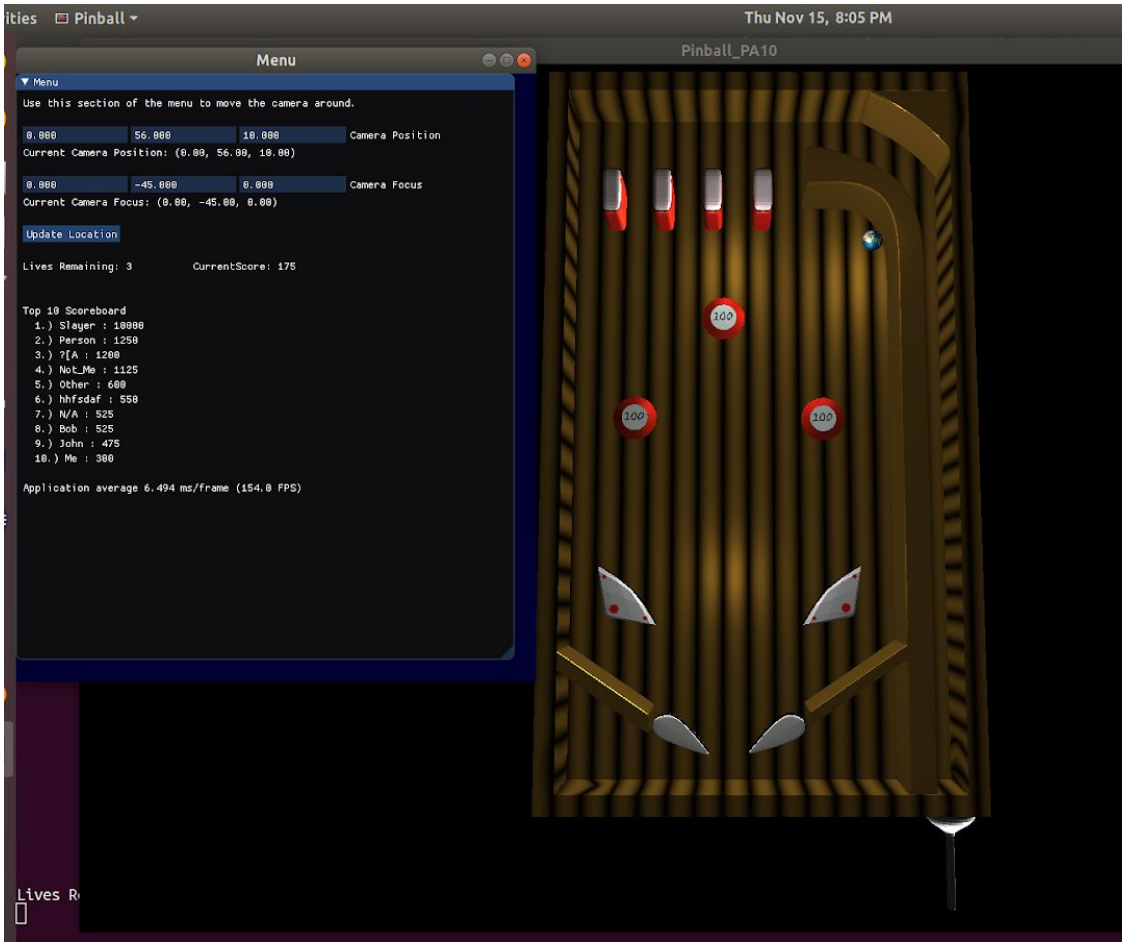

Figure 5: Game play with ball in motion.

Technical Manual

Issues

There are a few different issues with the collisions. Because the paddles are static objects and we are manually rotating them instead of applying torque, they aren't being upcast in the dynamics world which results in some odd teleporting of the ball on occasion. The left paddle is also prone to having its starting angle change. Another issue we are facing is that the score doesn't increase every time the ball contacts a bumpers, we believe this is because we're pulling the detected collisions from the dynamics world once a frame instead of using collision callback functions. We also had issues with the triangle bumpers where the bouncing is off since we couldn't figure out how to make static planes non-infinite. We also couldn't figure out why the ball seems to randomly spin, it doesn't affect the movement at all, but it spins the wrong direction, and the wrong speed, quite often.

Things We Could Have Done Differently

Given more time, there are a few things that we would have done differently. The first thing that we would have done different would be to load the paddles as dynamic collision objects in bullet. Moreover, if we had implemented layers, than the paddles could have been unaffected by the board and collisions with the ball would be much more consistent and smoother. Another big change that we would have liked to implement is a better movement system. Given more time, we would have liked to implement a movement scheme that made use of the mouse instead of just the arrows keys. This would have made the game much more interactive and intuitive. A

final thing that we could have done differently involves the IMGUI menu. We should have created different sections of the menu for the the camera movement, lives and scoring, and scoreboard. This would make is easier to play the game and see the different things the Menu has to offer.

Recent Changes

Since Wednesday, various changes were made to improve the game. The biggest and most visually noticeable change involved improvements to the models of the pinball table and its various parts. Specifically, the scaling on the table was changed to make it wider to better accommodate the launching of the ball at the start of the game. Additionally, the plunger was moved to the correct position and scaled so that it could be more noticeable from the default viewpoint. In addition to this, the lighting characteristics of the table and plunger were improved so that everything could be more visible at the default settings. All these changes were accomplished by editing parameters in the "launch/DefaultConfig.txt" configuration file.

In terms of game mechanics, the bumpers and paddles were given more restitution via the bullet physics engine. This was done so the ball would bounce off the the bumpers and paddles more, giving the user a better chance of keeping the ball in play. Before adding the extra restitution, the ball often failed to bounce as much as one would expect in a game of pinball. Another change to the game mechanics involved the dynamical inclusion (and exclusion) of a wall on the left side of the board to prevent the ball from entering the starting zone once launched. This wall was put into place via the *btStaticPlaneShape* collision object.

The final major change to the game involves the addition of usernames and a top 10 scoreboard. A separate scoreboard class was added to read and edit a text file containing the 10 highest scores achieved in the game. This class was used to display the top 10 scores in the IMGUI menu, as shown in Figure 3. In conjunction with the scoreboard, a system was added to allow the player to enter a username at the conclusion of the game via terminal input. This way, if the user scored enough points, the entered username and achieved score would be added to the scoreboard file. This change would then be reflected when the game was run again.