Haramball

PA10

By: Hannah Munoz – CS 480

Connor Scully-Allison - CS 680

Kurt Andersen – CS 480

Sections:

Overview:

Dependency Instructions:

- For both of the operating systems to run this project installation of these three programs are required [GLEW](http://glew.sourceforge.net/), [GLM](http://glm.g-truc.net/0.9.7/index.html), and [SDL2](https://wiki.libsdl.org/Tutorials).
- This project uses OpenGL 3.3. Some computers, such as virtual machines in the ECC, can not run this version. In in order to run OpenGL 2.7 follow the instructions at [Using OpenGL 2.7](https://github.com/HPC-Vis/computer-graphics/wiki/Using-OpenGL-2.7)
- This project uses Assimp 3.2. Instructions for downloading and running Assimp can be found at [Main Downloads](http://www.assimp.org/main_downloads.html)
- This project uses ImagicMagick 6.8.9-9. Instructions for downloading and running ImageMagick can be found at [Install Source] (http://www.imagemagick.org/script/install- source.php)
- This project uses Bullet 2.86. Instructions for downloading and running Bullet can be found at [Releases](https://github.com/bulletphysics/bullet3/releases)

Extra credit:

- Top 10
- Bouncy Bumpers
- Press and hold space bar to increase ball power for launch
- Ball count display in game
- Used all btConvexHullShape and btTriangleMesh for all objects with rigidBodies

Harambball
Technical Manual

By: Hannah Munoz

Connor Scully-Allison

Kurt Andersen

Issues:

Our first biggest issue was getting the collision detection to work properly with btConvexHullShapes and btTriangeMeshes. Once we learned the difference between the mesh types, it allowed us to determine the best possible collision meshes for each object. In the case of the ball itself, we determined that using a btSphereShape was the best because it allowed the ball to roll smoothly. We originally had it with a btConvexHullShape and it wasn't rolling like a ball should. After

we had all the basic collision on the table, we had to design a plane that would not allow the ball to fall off the table. We used a btStaticPlaneShape for this with the plane normal facing downwards.

Another major issue that we ran into was getting the paddles to stay in one position and to get them to move properly. We knew that the paddles had to be dynamic shapes, so we had to put btConvexHullShapes on them. When we would place them in position, they would just start to float around our board due to gravity. We tried changing the gravity on the objects so they wouldn't move, but they still just floated everywhere. After some research,, we found setLinearFactor and setAngularFactor. This allowed us to move the paddles to their position then only allow them to rotate around the y-axis and not be allowed to translate on any axis.

Now that the paddles were positioned properly it came down to adding the physics behind them. We initially tried using the addTorqueImpulse function, but the objects turned into helicopter blades and we couldn't stop them, so we backed off from that idea. For a short time, we had the objects snap between two positions, up and down. This was a temporary fix and only a band-aid because if you pressed the paddle button while the ball was near, the ball would just fall through and there would be no collision. After some thought we went back to the addTorqueImpulse function, but this time we decided to implement DT with the function as well. On the initial button press torque was added to the object, while the button was held the flippers would be allowed to rotate until a certain amount of time had elapsed. When that time elapsed, linearVelocity and angularVelocity were set to 0 to stop the rotation of the object. When the button was finally released, the objects would move back to their starting, downwards facing positions.

Another thing that was difficult for our group to accomplish was taking multiple keyboard inputs at one time. We wanted the user to be able to press both flipper keys at once and have them both move at the same time. For this to happen, we implemented a vector of key presses in engine.cpp. This would keep track of which button was pressed, held, and released. In our graphics.update function we check every piece of the vector to see which keys are being pressed. The only thing we had to do is check for which index each key was pressed, so in the objects update, it wouldn't get called multiple times for looping through the entire vector, it would only get called with the appropriate keypresses.

When our group switched from flat colored objects to textures, we ran into an obnoxious problem. Earlier in the semester when we did PA6, the texture loading project, we ran into the problem of having multiple textures saved to one object. In our PA10, our game board has multiple textures placed on it. In order to overcome this problem, we broke our game board into multiple separate object files and placed the textures on them individually. This way each object loaded individually with their own separate textures.

What would we have done differently?

For this project, we would have like to have spent more time on the spotlights. For some reason that we were not able to figure out, our spotlight tends to stick to the origin of the view space.

We also would have liked to figure out how to not let sticky keys activate in our game. It can cause nightmares with the camera if the left or right arrow key get stuck. We would have probably made it so that our initial lighting projects were all done with textures so we would have been more used to working with textures and shaders together rather than learning to do textures and shaders together on this project.

What was changed from Wednesday:

From our submission on Wednesday, we updated our game to use textures from flat colors, we updated the shaders, and we finished working on the scoreboard.

Harambball
User Manual

By: Hannah Munoz

Connor Scully-Allison

Kurt Andersen

Build Instuctions:

To run Haramball, ensure that you are in the proper build directory [PA10/build] and compile the code as follows.

cd build

cmake ..

make

To begin playing Haramball, enter the following command:

./Haramball

A new window should appear with the pinball table ready to start. The number of balls left will be displayed on the upper left of the table. The score will be displayed in the terminal once the game has ended and the user has run out of balls.

If you make a top 10 score you will be prompted to input your name at the end of the game in the terminal.

Keyboard Inputs:

TO AVOID A STICKY SITUATION(avoiding sticky keys) do not press shift and then any of the arrow keys 5 times consecutively.

Game Controls:

Spacebar - press and hold to add force to launch the ball, release to launch ball [Fig. 1]

Right shift – moves right paddle, press and hold to stay in up position. [Fig. 2]

Left shift – moves left paddle, press and hold to stay in up position. [Fig. 3]

Esc – Quits the game.

Camera Controls:

Left Arrow – pans the camera to the left.

Right Arrow – pans the camera to the right.

Up Arrow – pans the camera upwards.

Down Arrow – pans the camera downwards.

'r' – reset camera position.

Shader Controls:

'p' – switch to Phong shading. [Fig. 2 & Fig.3]

'g' – switch to Gourand shading. [Fig. 1]

Numpad Controls:

'0' – increases specularity on table [Fig. 4]

'.' - decreases specularity on table

'1' – increases bumper specularity. [Fig. 5]

'2' – decreases bumper specularity

'4' – increases ball specularity. [Fig.6]

'5' – decrease ball specularity

'7' – increases flipper specularity. [Fig. 7]

'8' – decrease flipper specularity

'9' – increase spotlight height [Fig. 8]

'6' – decrease spotlight height.

.

'+' - increase ambient lighting. [Fig. 10]

'-' - decrease ambient lighting.

'*' - increase spotlight ambient lighting. [Fig.11]

‘'/' - decrease spotlight ambient lighting.

Figures:

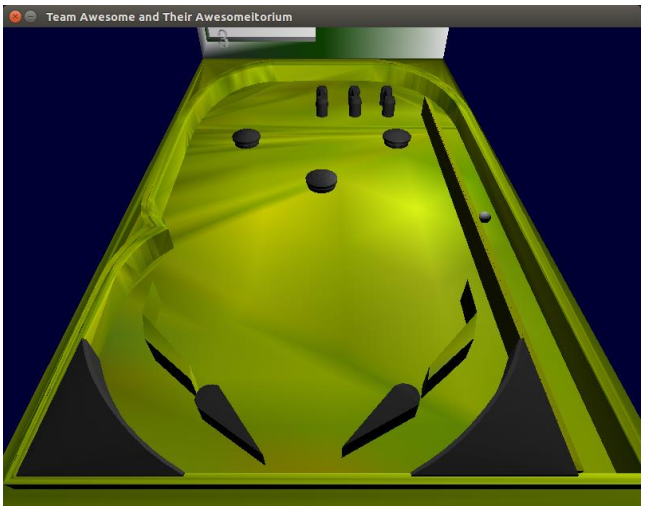

Figure 1: Releasing the space bar applies force to the ball. Also a good example of the Gouraund Shader.

Figure 2 & 3: Hitting the shift keys move the flipper. Also a Good example of the Phong Shader.

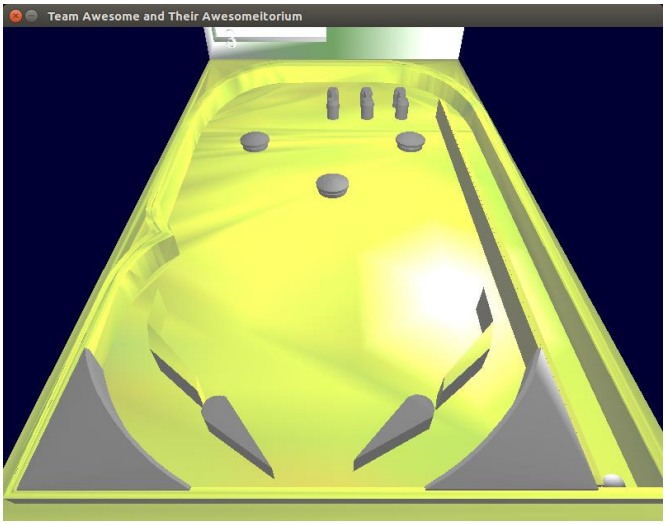

Fig. 4: Increasing the specularity of the table.

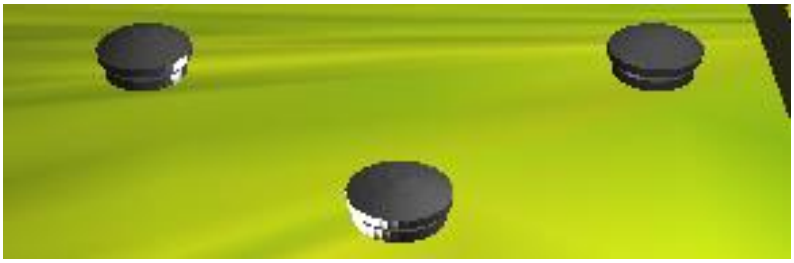

Figure 5: Increasing the specularity of the bumpers

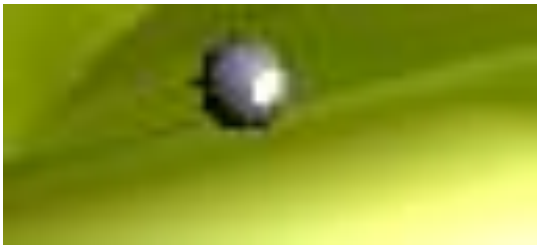

Figure 6: Increasing the specularity of the ball

Figure 7: Increasing the specularity of the flippers

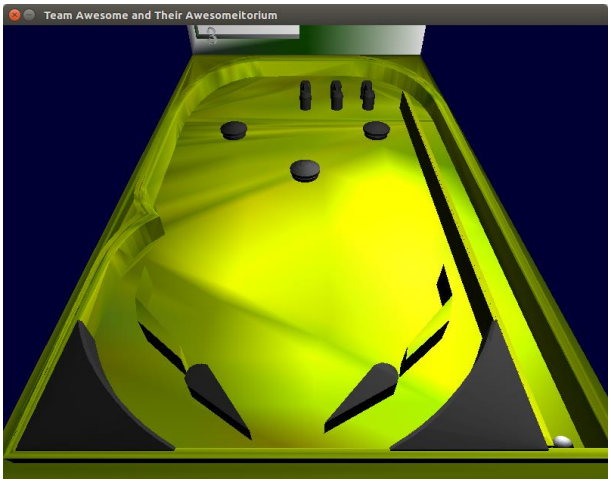

Figure 9: Increasing the spotlight's height

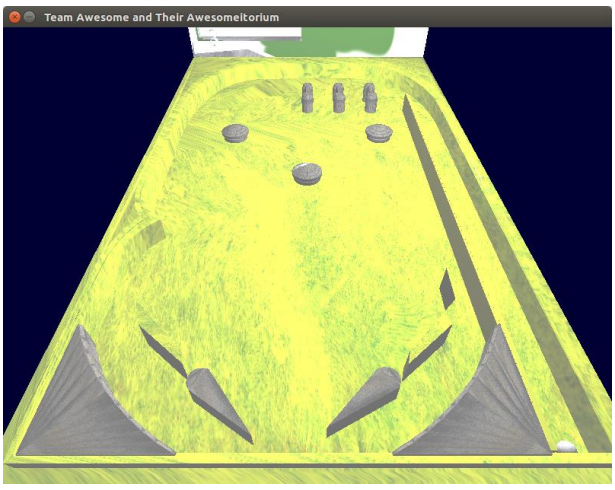

Figure 10: Increase ambient lighting

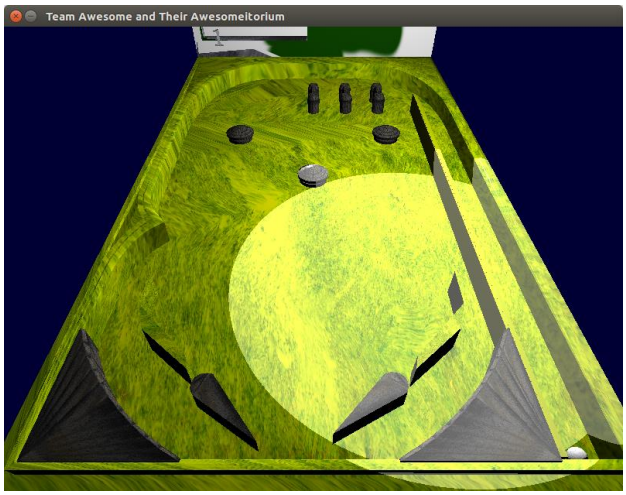

Figure 11: Increase spotlight ambient lighting