Joust

Programming Assignment 11

CS 480 Computer Graphics

Team: Specular Bois
Members: Hadi Rumjahn, David Valenzuela, Jay Woo

Date Demonstrated
December 17, 2018

Table of Contents

Overview

Project Description

Our project is a two player competitive jousting simulation game. Each player controls a piece and moves their own piece by using the keyboard. On player controls a red piece and the other player controls a blue piece. Each piece can be moved forward, backward, left, right, and can also jump. Each players' objective to is to knock the opposing player's piece off the map three times. Once either player loses three lives both players' pieces are moved to a new map. The lives for each piece are then reset back to three. There are a total of three maps. Once all three maps have been played the game can be reset for the players to play again. There is also an invincibility time period which does not make a player lose any lives if they all of a ledge after recently falling of the ledge.

Dependencies

For both of the operating systems to run this project installation of these three programs are required GLEW, GLM, and SDL2.

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

This project also uses Dear ImGui (https://github.com/ocornut/imgui), but all the necessary files are already included in the project.

This project also uses the Assimp, ImageMagick, and Bullet physics libraries.

Extra credit

Sound

There are various sounds that play while players are playing our game. Sounds play when a certain event occurs such as when a player's piece falls off a ledge or when a player's character reaches zero lives. There will also be background music that plays during each map. Adding sound contributes to a more fun player experience when playing our jousting simulation game.

SkyBox

There is a skybox which constantly follows the camera. This way no matter where the camera is moved, the skybox will always appear at the same distance. Adding a skybox makes the environment look more visually pleasing.

User Manual

Build Instructions

Download the PA10 project folder from the following website, https://github.com/david4jsus/cs480Valenzuela. Open the terminal in the PA11 folder by double clicking the PA7 folder, right clicking inside the folder, and select the open terminal here option. Inside the terminal enter the following commands shown below.

```
mkdir build
cd build
cmake ..
make
```

Run Instructions

After building the project open the terminal in the build folder. To run the project enter the command ./Joust into the terminal.

Keyboard Input

Camera movement

The camera can be freely moved throughout the environment using mouse mouse movement, holding the left click button on the mouse, and the left or right shift keys. Below is a list of keys which control camera movement. Pictures of viewing a map from different angles can be seen in Figure 1.

Hold left click and move mouse left: look left
Hold left click and move mouse right: look right
Hold left click and move mouse up: look down
Hold left click and move mouse down: look up
Hold shift and move mouse left: move left
Hold shift and move mouse right: move right

Hold shift and move mouse down: move forward
Hold shift and move mouse up: move backward

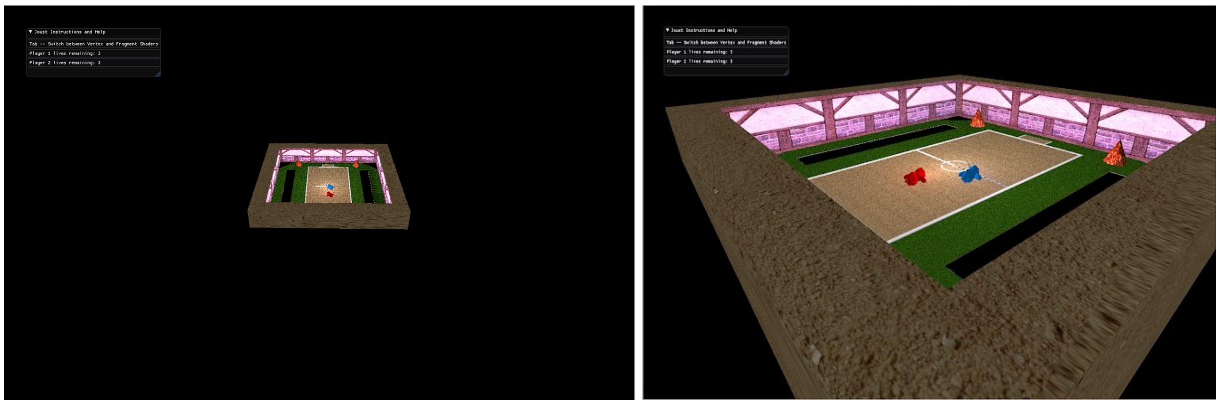

▼ Joust Instructions and Help
Tab -- Switch between Vertex and Fragment Shaders
Player 1 lives remaining: 3
Player 2 lives remaining: 3
▼ Joust Instructions and Help
Tab -- Switch between Vertex and Fragment Shaders
Player 1 lives remaining: 3
Player 2 lives remaining: 3

Figure 1. Shows a map from different camera angles and positions. A player can freely change the camera orientation and position.

Player 1 Movement

Player one can freely move their character piece by using the the keys on a keyboard.

W: move forward
A: move left
S: move backward
D: move right
Space bar: jump

Player 2 Movement

Player two can freely move their character piece by using the the keys on a keyboard.

Number pad 8: move forward
Number pad 4: move left
Number pad 2: move backward
Number pad 6: move right
Number pad 0: jump

Maps

There are three maps in which the players can play on. The first map features a castle with a large circular wall surrounding the entire map. Some building are placed within the map to make it look more aesthetically pleasing. There is a circular ledge from which players can push each other's characters off the map. The second map also features a large circular wall which surrounds the map. There is a field on which the players try to push each each other off. A circular ledge surrounds the field. The third map is box shaped. There are magma towers but they are used purely for visual purposes. This map also uses a field for players on which players can push each other on. The ledges for this map are on the left and right sides of the field. All three maps can be seen in Figure 2.

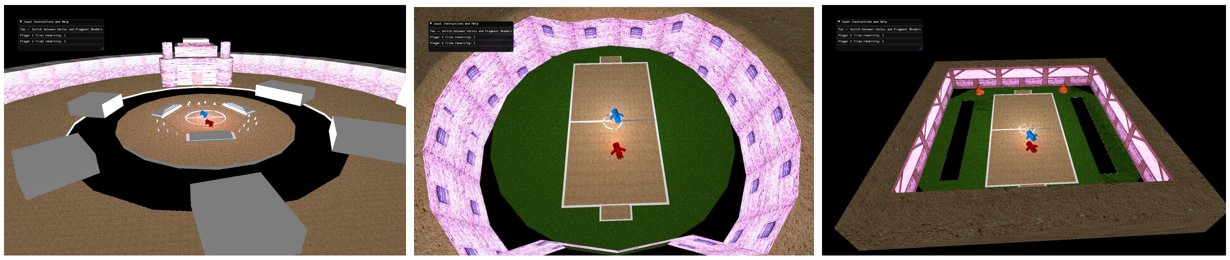
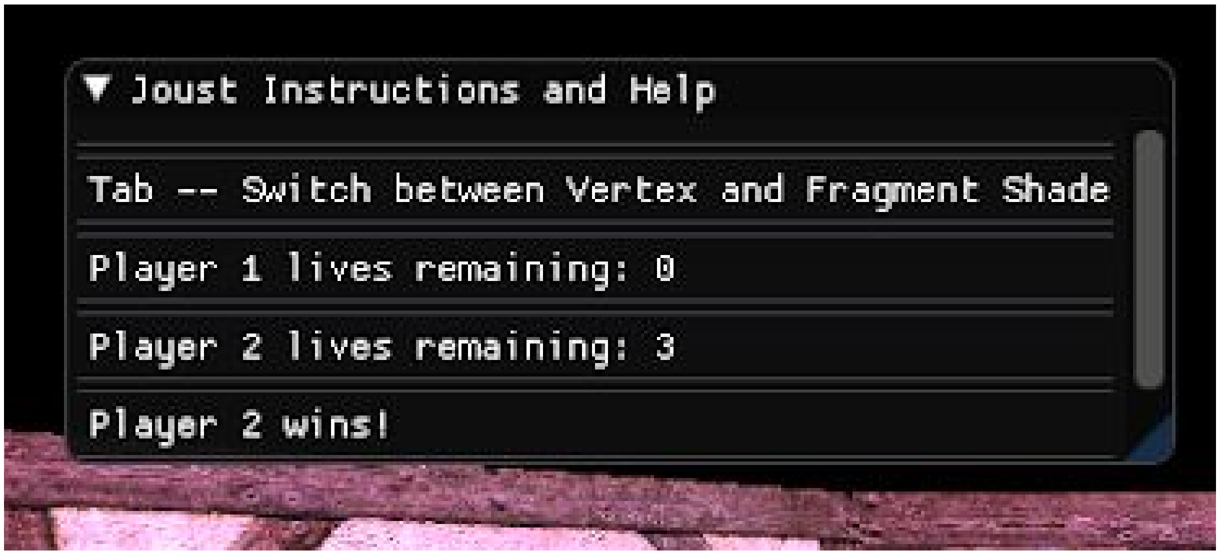


Figure 2. Shows each of the three maps. Players can freely transition between each map.

Lives and Win Text

Inside the game there is a menu which displays the remaining amount of lives each player has left. It first displays the amount of lives player one has and right below that it displays the remaining amount of lives player two has. Lives will not decrease for either player once either players' lives reaches zero. Once a player's lives reaches zero, the menu displays new text which states which player won the map. For example, if player one reaches zero lives remaining then new text will appear saying "Player 2 wins!". This can all be seen in Figure 3.


▼ Joust Instructions and Help

Tab -- Switch between Vertex and Fragment Shade

Player 1 lives remaining: 0

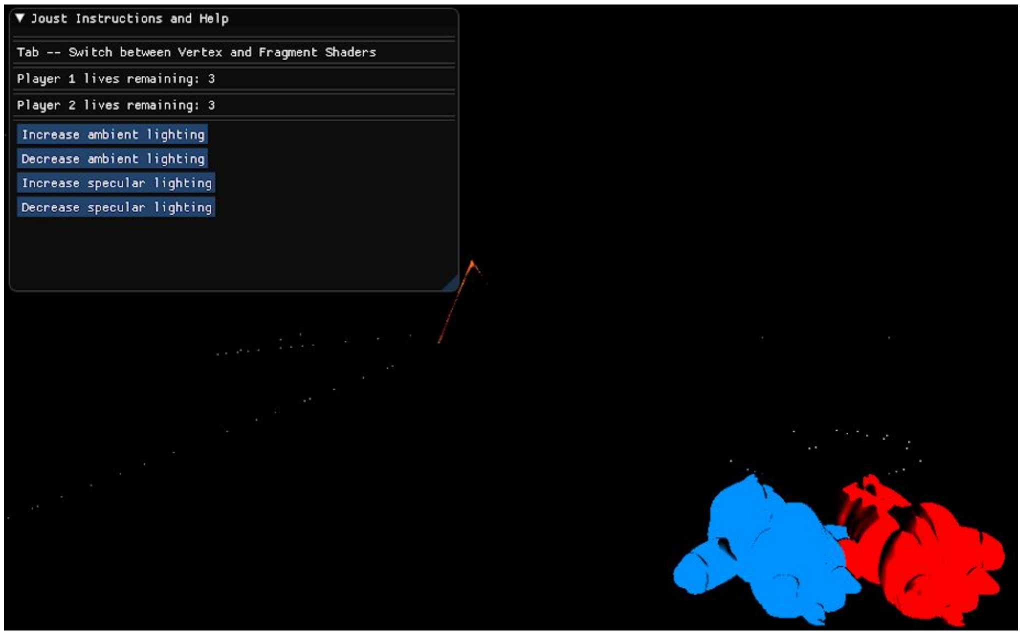
Player 2 lives remaining: 3

Player 2 wins!

Figure 3. Shows the in game menu which displays the remaining amount of lives each player has left. Also displays who won the match.

Lighting

Lighting can be changed between per fragment lighting and per vertex lighting during runtime by pressing the tab key. The program starts on per fragment lighting. Ambient lighting can be adjusted by clicking the increase and decrease ambient lighting menu buttons. Specular lighting can be adjusted by clicking the increase and decrease ambient lighting menu buttons. These menu options can be seen in Figure 4.


▼ Joust Instructions and Help
Increase ambient lighting
Decrease ambient lighting
Increase specular lighting
Decrease specular lighting

Tab -- Switch between Vertex and Fragment Shaders

Player 1 lives remaining: 3

Player 2 lives remaining: 3

Figure 4. Shows different lighting effects. The ambient and specular lighting can be changed using the buttons in the menu inside the game.

Technical Manual

Issues

As of December 17, 2018, there are no issues with our project. One issue we did come across while working on the project is that each player's character would not move when transitioning to a new map. The characters would correctly and appropriately spawn inside the first map but not in the second and third map. This was caused the characters being trapped in the first map by invisible walls. We assumed that the characters would be able to go through the invisible walls since we would manually be placing their positions within the code. To fix this issue we made the invisible walls box colliders instead of static plane colliders.

Changes

One change to our project is that we decided to make a configuration file. The configuration allows us to quickly load in a list of objects and also quickly changes certain properties for specific objects. The configuration file also holds information about shader file locations, camera starting orientation and position, gravity strength and direction, and jump height for each character. The configuration file has allowed us to test our code without having to recompile every time which has allowed us to save much on time.

Another change we made is that we originally intended for each player's character to wield a weapon. Each character was going to wield a lance. If a player's lance touched the opposing character's body, the opposing player would lose a life. If both characters bodies hit each other nothing would happen and the characters would simply bounce off each other. For the current version of our game we decided to take out the weapons and instead decided to have both players attempt to push each other's character off the map. We did this change since we did not have enough time implement weapons in such a way that they would constantly follow a character's body. While discussing possible project ideas, we anticipated that this would be a potential problem.

Things we would have done differently

Knowing what we now know one thing we would have done differently was not decide upon a project which relied so heavily on complex compound rigid bodies. Our original idea required for us use two seperate rigid bodies with one rigid body always following another rigid body. We attempted many different ideas in an attempt to solve this issue. One idea we attempted was to copy the orientation and position of one rigid body and copy it to another rigid body with an added distance offset. The problem with this method is that it caused to rigidbodies to constantly collide with each other. Another method we attempted was to use bullet constraints. The problem with this method was that the rigid bodies would move in unpredictable manners.