# Mouse Accuracy: The Game

# Computer Graphics

Abhijith Settipalli

Andrew Morris

Dustin Tsui

Index

Index

1. Index
2. Summary
3. Game Design
4. Technical
   * 1. Three JS
        1. Shadows
        2. Frustrum
        3. Object Imports
        4. Phong Shader
        5. Raycasting
        6. Shader Highlights
        7. Antialiasing
        8. Cube Maps
5. Level design
6. Improvements

Summary

We came upon the original idea of a paint program. Originally it was based to be painting off of an object in 3d space, where we could rotate the object around and paint willingly. While we were working on the project, we were unsatisfied in the outcome. The project was not very interesting nor was its scope large enough to encompass much of the material we had been covered in class. As such, we slowly shifted the idea towards a more game oriented. We had two-thirds of the members having experienced game knowledge, so we decided that it would be much more entertaining of a project. Having desire to make a game from the start, we reassessed our project and had to quickly start on our game.

We knew from the start, ray casting was going to be an integral part of our project. Since we use the mouse exclusively, it was only fitting. We quickly came up the idea to making an abstract mouse clicking game. Other things that we were going to research and implement, such as importing objects, and using a phong shader, were divvied up and tasked to each group member.

Along the way, we ran into some unseen circumstances, like certain limitations with Three JS and not having nearly as enough time to solve issues. The biggest setback was unable to completely integrate objects imported into the scene. While one of our intentions was to make levels with imported objects, it was scrapped because of tricky ray casting errors. Determined that it was too costly to try to solve such issue, we changed the multitude of bridge objects into basic geometry. However, we were able to fully implement object importing after a plethora of trial and errors.

Game Design

**Objective**

The objective of the game is to reach to the blue block starting from the red block. This is accomplished by traveling on the bridges made by the moving geometry. The geometry could be any type. Access to Three JS enables shapes to be rendered well without sacrificing performance.

**Gameplay**

The gameplay involves with the player moving the mouse. They will need to time and quickly react to opportunities given to them by the game. The player must keep their mouse steady across on the geometries and not to stray from the path. Doing such will accomplish the goal.

**Mindset**

Patience is a key element in this game. Moving when the player is unprepared will likely result in a failure. Knowing when and how the pattern moves will be something for the players to study. Steady hands are a requirement, the game can be very unforgiving, and sometimes the objects are not as large as they seem. In its essence, this game is about mouse accuracy, reaction speed, and mouse speed.

Technical

**Three JS**

We picked Three JS because we discovered how useful it was to use a library. Three JS has many convenient features that ease how much source code we need to write. It also came with a lot of free and open documentation. This was very nice for whenever we wanted to implement something that was complex. One of the features we used heavily was the import feature. We had many tests of imported objects of different types. In the end, we settled for the obj importer instead of the many others numerous importers because of the performance issue. It may not be Three JS’s fault, but the importers were very greedy with memory and would drop the frames to an absurd low. We simply couldn’t have a below ten fps mouse accuracy game, so we settled for obj.

Three JS has a lot of support with various features. Like being able to create scenes very quickly, something that would’ve taken much longer to create. Things like changing the camera angles, and movement is much simpler and more intuitive. Shadows are the full implementation of webgl’s capabilities. It is a very powerful tool that reduced the amount of menial labor required to start a project in webgl.

**Shadows**

**Frustrum**

**Object Imports**

As mentioned before, object importing was originally a key part of the project. However this was scrapped as we came to an unforeseeable obstacle. As for our importer, we used an obj importer. This is akin to one of the homework we had. An obj file contains definitions packaged into one file. Each line contains information about a type of line or vertices that could be created. In particular, our tree obj has over 430 vertices and- needed many more vertex normals to create the tree object onto the scene. Our particular tree, handled texture mapping as well so this was very nice.

The object import is also important because it was also the handler for the texture import. We had covered textures in class, but sadly we were unable to have any hands on experience. It would have been interesting to implement textures that were more advanced like the environment mapping. It may be added with more time since Three JS handles mapping on an object fairly well.

**Phong Shader**

**Raycasting**

In class, we went over ray casting quite extensively. In our project, we have a single ray cast from the mouse onto the screen. This is especially important, because this determines whether the player is still on an object. This ray cast goes through all the objects and returns an array if it passes through anything.

**Shader Highlights**

Shader highlights appear whenever the player is playing the game and is glancing over an object that is usable as a bridge. This was accomplished by mimicking the object’s shape and having a slightly larger object surrounding it. It renders a very accurate highlight that more easily pops out to the player. This helps emphasize which object the player is currently on. This recognition ability is key to player accuracy.

**Antialiasing**

**Cube Maps**

Level Design

Our levels were made to slowly ramp up the difficulty. One of the things we wanted to do was having each level display a different graphic technique we had learned. This kept things a bit fresh with every level with our final level looking very different than the others. Overall look and feel of the game was originally intended to be a very abstract conceptual game. But in the final level, the game has more realism to it. Certainty the earlier levels were made to be extremely abstract similar to many of mouse moving games. Our goal was to experiment with as many different things as we could in the short amount of time with the Three JS system. It was certainty a very fun experience to have since we were trying to implement with various things at a time.

We tried to keep the game elements to a simple system. We wanted a more experimental feel of the graphics systems. This lead to the very abstract and one type of mechanic game system we have for each level. We had intentions to create complex levels, but we were very focused on creating various little things to the graphics system and ran out of time to develop a super difficult final level.