WebGL Massively Multiplayer Online Game

Student
Gianni Chen
giannic@seas.upenn.edu
University of Pennsylvania

Advisor
Professor Norm Badler
badler@seas.upenn.edu
University of Pennsylvania

Advisor
Aline Normoyle
alinen@seas.upenn.edu
University of Pennsylvania

Figure 1: *Screenshot in game*

Abstract

We will build a multiplayer online game using WebGL and web sockets. Players will connect to a server in their browsers and be able to view other players who are also connected. The goal of each player is to eliminate the others in a free for all format. The players are ranked through the prescribed calculations involving eliminations, deaths, and other properties. This is a simplified combination of current approaches which are either solely WebGL demonstrations or HTML5 and web socket games. One of our primary goals is to provide a stepping stone to build more sophisticated multiplayer online WebGL games.

Project Blog: http://webglmmo.blogspot.com

1 Introduction

Since the late 1990s, Massively multiplayer online games (MMOG) have been a popular alternative to other games as well as a pleasant escape from everyday life. This was made possible not only by the improvements to our internet systems, but also by the increased power of our local machines. More recently, many classic games have been making reimagined appearances as MMOGs in our browsers. Common ways to achieve the real-time aspect include Comet, a model which allows pushing of data to a browser without request, and sockets, which has a variety of implementations on the web for asynchronous input and output. We have also seen a movement of 3D graphics towards the web, starting from Flash to Canvas 3D and in 2011, the first stable release of the Web Graphics Library (WebGL). While WebGL allows for both 2D and 3D rendering, the *multiplayer* games we currently see are often in 2D. It is rare to see a multiplayer online game built in 3D using WebGL.

In the initial time frame of this project, we will implement a barebone game. The motivation is not to build the next World of Warcraft, but to set a base for those who would like to build full-feature multiplayer WebGL games in the future. Though the code base will be presented as a game instead of a boilerplate, parts of it should be adaptable to new environments for their respective purposes. This will be helpful to the games community as well as related researchers because it can be used as a rapid prototyping tool. There are parallels that can be drawn between this project and tools such as Twitter Bootstrap for their base functionalities in different fields.

Our project aims to combine the two very new technologies, WebGL and WebSocket, to build a very simple 3D MMOG. We will start by learning the technologies required for implementation. Once some familiarity is established, we will first implement an interactive 2D environment in WebGL for a local machine. Then, we will build upon that to make a 3D environment, bringing it a step closer to the desired look and feel. At this stage, we will begin to prototype some game features and dynamics. Then, we can begin experimenting with the WebSocket protocol on the 2D

prototype to support multiple players. Once we successfully have multiplayer input, we will generalize it to the 3D environment to achieve our desired outcome.

This project makes the following contributions:

- Explores a new alternative for gamers on the web
- Establishes a new front and back end combination for web games and developers who may be interested
- Provides base code for future WebGL multiplayer environments

1.1 Design Goals

There are two major audiences for this project: casual online gamers and web game developers. The target consumer is any person who has access to the web and has an interest in interacting with others through games. This is a broad audience, but we have seen in the past that many people are fascinated by interactions as simple as seeing people in another continent through a camera in the shape of a telescope on the street (Appendix). We want users to feel a competitive connection with others who are in the same virtual space. On the other end of the spectrum, this framework will target game developers, especially those interested in web technologies. Like Twitter Bootstrap provides website developers a quick way to host content, this project will allow game developers a quick way to build a functional game.

1.2 Project Proposed Features and Functionality

This project implements the following features and functionality:

- An interactive 3D environment accessible by anyone with a WebGL supported browser
- Support for multiple players to interact in the environment
- A way for players to navigate the environment
- A way for players to eliminate one another

  e.g. weapons, elements etc.
- A competitive way for players to avoid being eliminated

  e.g. jumping, shielding, etc.
- A presentable and deliverable open source base code

2 Related Work

Web games in the browser collectively have the largest user base. This reason has driven development for this particular medium for games to explode in the past two years. While the traditional tools used to build these were Java and Flash, more and more developers have chosen to move to HTML5 due to its advantages in efficiency and compatibility. More recently, with the release of WebGL, some developers have experimented with it in conjunction with HTML5 for its advanced rendering capabilities. It is difficult to find official papers on this subject in graphics conferences, but there have been some presentations and courses. One such example that I will be using as a tool for learning this technology is *Introduction to WebGL* [Jacob 2012]. Found across the web, we see a variety of projects from individual developers to larger organizations. We have described a few notable ones.

2.1 Khronos Group SIGGRAPH

The Khronos Group consists of media centric companies such as NVIDIA, Electronic Arts, and Google, among others. They focus on creating "open standard, royalty-free APIs" which allow graphics developers to create content for interested users or consumers. At SIGGRAPH, they have a variety of presentations on these standards and libraries, including tutorials which give an overview of their current capabilities. For example, at SIGGRAPH 2012, they presented *Graphics Programming for the Web: WebGL* [Russel and Mo 2012].

2.2 Chrome Experiments

Chrome Experiments, hosted by Google, is a collection of some of the most creative web experiments using cutting edge web technologies such as HTML5 and WebGL. They demonstrate just how powerful these tools are and provide a point of references for state-of-the-art web technologies. Many developers have built their applications with this collection as a driving force for improvement. Many of these submissions have are open source and provide a useful means of learning the standards, conventions, and techniques in developing with these tools. For instance, in the projects tagged multiplayer, DCubic has implemented a space where multiple users can connect and create cubes in space of various colors and animation features [Perez-Fadon 2012].

2.3 BrowserQuest

BrowserQuest is a MMORPG adventure game built by Mozilla Foundation [allergic 2013] and Little Workshop. Little Workshop is a duo, Guillaume Lecollinet and Frank Lecollinet, based in Paris, France [Lecollinet and Lecollinet 2012]. It is implemented using HTML5 Canvas and WebSockets. It is now an open source project hosted on Github. It was originally meant to be a demo of how these technologies can be used to implement a game, explaining why the in-game population has declined since. It hosts several load-balanced servers, with multiple environment in each one. Upon connection, the player can interact with other players in the environment they were connected to. Another notable contribution of BrowserQuest is its compatibility with mobile devices (iOS, Android, Firefox).

2.4 HexGL

HexGL is a HTML5 and WebGL game single player racing game built by Thibaut Despoulain, a senior Computer Engineering student at Université de technologie Belfort-Montbéliard. This too, had optimizations to bring the game to mobile. This impressive demo was one of the first noted to use three.js, a library built on top of WebGL. For that reason, it was later added as an entry to Chrome

Experiments [Despoulain 2012].

3 Project Proposal

This project aims to create a functional 3-dimensional Massively Multiplayer Online Game. Its core components will be implemented with WebGL and the WebSockets protocol. We will first build a 3D environment. This will likely be a simple terrain, but is a means for interaction between multiple players. This space will be populated by users connecting from remote locations. Each user may be represented by something as simple as geometry moving in perspective space on keyboard input.

The key here is that the users can interact with each other with minimal delay. To demonstrate this, the game will be a standard deathmatch, where any given player tries to eliminate as many other players as possible in order to climb the rankings. Upon elimination, a player will respawn in a random location. The means of attack for a player will be via a "shockwave" that radially emanates from the player itself in the horizontal plane. Other players can avoid elimination by jumping, that is changing their vertical height temporarily. This is a simple framework that will set a starting point for more sophisticated renderings of future MMOGs built with the same stack.

3.1 Anticipated Approach

The environment for this game will be in a finite space. We will define the terrain first with a simple rectangle, then import a terrain defined by an object file with changes in elevation at different points. This terrain will be the minimum y value any player can be at. That is, it will be a rigid collision object.

Suppose for now our players are represented by spheres. Each player has 2 possible actions: to attack or jump. To attack, a circle, representing a shockwave, concentric with the player in the horizontal plane will be generated. Over a short period of time, this circle will expand in radius and decay in power until it reaches 0, at which point it will disappear. To jump, the player will follow a simple $f = mg$ update of the player given some initial velocity in the y direction and a constant horizontal velocity at the point of take off [Kankaanpaa 2004]. For any player, if their model intersects with another player's shockwave, their HP will decrease. Once a player's HP falls below 0, they are pronounced eliminated and will respawn after a delay in another location.

This will be implemented by keeping lists of players and shockwaves. At each timestep in the game, each player will be checked against the list of shockwaves for intersections. For each intersection, their HP will fall a predefined constant amount. At the end of each time step, players who died will have their deaths incremented by 1 and players who eliminated others will have their kills incremented by the number of players they eliminated. These statistics will be used to rank players.

The multiplayer portion will be implemented through handling WebSockets using the socket.io API. We will use node.js to host a server to which multiple computers can connect to. Socket.io then allows us to pass data back and forth between the server and each client. This way, when one client has a state change, everyone else will get the same update.

3.2 Target Platforms

I will be using the standard web front-end technologies, HTML, CSS, and Javascript. Building on top of this, I will be using WebGL and related libraries such as three.js. For socket handling, I will be using socket.io. Finally, I will likely be using node.js to host the server. The user will be able to access this online game through any browser and machine that supports WebGL (e.g. Chrome, Firefox, etc.).

3.3 Evaluation Criteria

This MMOG will be evaluated under these criteria:

- Is this method a step forward from traditional methods of creating online games?
- Does the game show that WebGL and WebSockets is a feasible future direction in online gaming?
- How well are players synchronized in the world with the rest of the population?
- Does the game have potential to be adapted and morphed into more sophisticated games?
- How easy is it to get started building a larger game off of this one?

4 Research Timeline

4.1 Project Milestone Report (Alpha Version)

- Complete all background reading
- Experiment with simple prototypes of 2D games combined with WebSockets
- Experiment with 3D WebGL renderings of simple geometry
- Architect the modular pieces of the code and how their contracts with other parts

4.2 Project Final Deliverables

- Fully functional 3D MMOG built in WebGL with WebSockets
- Hosted game so users have access from anywhere on the internet
- Documentation that makes it easy for a user to build off of this game

4.3 Project Future Tasks

- Optimize connections and renderings to scale the application for more connections
- Integrate more physically based calculations into player interactions

    e.g. pushback from another player's attack

- Assign object models to players to make for a better demonstration
- Apply more sophisticated renderings of animations

    e.g. particle systems, lights

- Create more detailed terrains so they can be full maps with obstacles

---

5 Method

- **Server**: The server was set up initially to start serving some simple files, initially static html with canvas. This way, it would be smoother to later incorporate full dynamic scenes without restructuring the way the files refer to each other.
- **2D Client**: The initial approach to both learning the technology stack and developing the game was to make a pure client side 2D scene. This is meant to be a simple onboarding as well as a setup for the 2D Network experiments. This consisted of a simple scene drawing multiple shapes on a flat canvas.
- **2D Network**: Once the 2D scene was made, the network component was implemented as a first step to proof of concept that WebGL and WebSockets are not allergic to one another. At this point, the framework only passes the location of each client to the server, and the server to other clients. This is simply an x, y pair. This pseudo-code is a simplified concept of the server and client code.

```
// server code
client.on(move, new_location):
    client_list.broadcast(new_location)

// client code
client.on(keypress):
    move(keypress);
    send_to_server(new_location)
```

- **3D Client**: Almost simultaneously with the 2D Network, the 3D Client was being worked on. This, like the 2D client, is largely an experiment, but unlike the 2D client, will eventually become part of the final product. This began with drawing and transforming simple shapes in WebGL. Once that was working, I began to incorporate components of three.js to accelerate the drawing and rendering of a real game scene.
- **3D Network**: Arguably one of the most difficult parts of this project was to make sure the 3D scenes between all the clients are synchronized between all the clients and the server as well. A few components went into making this happen.

  First and foremost, keeping a consistent heartbeat via a timer in all the clients and server was important. This way, we can limit the number of actions of each client to a small finite number on each beat. For example, if a client holds down a key, the result would then not depend on the keyboard repeat rate of the client, but on the speed of the heartbeat on the server and client.

  Second was to limit the amount of information and frequency of data transfers. Minimizing the size of the packet as well as only broadcasting to clients only when necessary is important. Also, the implementation of dead reckoning decreasing the amount of packets being sent as well.

  Finally, though the heartbeat seems to solve the problem of timing, it was important to make sure the game states in the clients and the servers were consistent at all times. For example, in the demo game, a player should not be able to hit another player if not both of them are synchronized in location and velocity. This was achieved by mainly server-side checks to make sure all clients are where they think they are. If not, the server would then update them appropriately.

- **Gameplay**: Finally, while implementing the 3D Network and Scene, the gameplay component was crucial in both the testing and final product. In order to make sure the clients can interact correctly, much of the gameplay needed to be implemented along the way. There is still plenty to implement before this demo can be a full consumer quality game, but the current installations provide a robust proof of concept that this technology stack is a feasible, powerful, and generalizable approach to a large network of developers, gamers, and researchers.

6 Results

The game framework I made was straightforward to expand to a demo game. It has implemented a 3D environment that allows different clients to connect to the same space online, and interact with one another. The sample game implements this in the form of a spaceship game where the players try to eliminate each other by shooting bullets. However, the code base allows this to be easily modified to any form of clients, interactions, or environments.

The documentation allows for easy onboarding of a developer who can then write simple utility functions to define the types of assets that should be loaded, the interactions they would like between clients, the environment that would like, and many other configurations.

7 Conclusions

I was able to achieve a solid proof of concept for a technology stack that uses WebGL, WebSockets, and Node.js to host a multiplayer online game framework as well as make a demo game out of it. I also achieve a personal goal of learning all of these technologies as well as the concepts behind making a game. This experience has allowed me to learn yet another stack of skills that I can refer back to in academics and in the industry. I hope this will be a useful contribution to both the online developers and the research community.

8 Future Work

Throughout the process of making this game framework, I became increasingly interested in the concepts and technologies behind the making of a game. This long list of future work is a side effect of my excitement.

- **Entity Interpolation (Valve)**

  Latency Compensation: A method to ease the lag that is experienced by clients is to store past data and show it to the clients as present data. As a direct result, present data will be stored as future data and presented as past data in the next frame. This is one large component of entity interpolation which makes the viewing of the game more smooth and, mimicking a low latency connection.

- **Collisions between players**: For the best effects, this would require a full implementation of rigid body collisions and physics behind it. The ideal effect is that ships would be damaged and physically move away from each other with jerk and acceleration on collision. Given a full physically accurate simulation, the force and angle of approach would also affect the result such that some collisions would result in damage and some would result in a nosedive.

- **Dead (Ded) Reckoning with acceleration (currently only velocity)**: Currently, the implementation uses a "naive" form of dead reckoning with velocity only. Major improvements are already seen, but it would be even more effective and correct if, in conjunction with latency compensation, acceleration was built into the dead reckoning implementation.

- **Network and Client side optimizations**: The data that is being sent through the network between server and client is not at its slimmest at the moment. Then, the same transformations and changes of states are being applied at both ends. This is necessary, but the functions that implement it are not optimized. Also, included in this, there should be server side checks for data consitency across clients.

- **Expose game states to clients**: One of the more straightforward tasks would be to expose the stats to each client. This would require a generic interface that would be flexible enough to keep track of different stats. In the case of the demo game, an example stat could be the number of other players a player has eliminated.

- **Environment obstacles**: Currently, there are clouds rendered on the client sides that make the scene more realistic and interactive. However, these clouds does not affect a player's ability to navigate. It would be more interesting if there were obstacles or effects that would affect the speed of a player or the spaces they can occupy.

- **More variety on models**: The HexGL game model is being used in the demo game made on this framework. To build more customization into the system, there can be different models, colors, etc.

- **Higher quality graphics**: Overall, the rendering of this project was not realistic. It was not the focus of this project. However, to push this project to the next level, it would make sense to make more realistic textures, materials, etc.

9 Appendix

A list of resources that I used, but did not quite belong in references.

- Multiplayer Chrome Experiments

  http://www.chromeexperiments.com/tag/multiplayer

- WebSocket Wikipedia

  http://en.wikipedia.org/wiki/WebSocket

- Comet Applications Wikipedia

  http://en.wikipedia.org/wiki/Comet_(programming)

- Telescope between New York and London

  http://www.cnn.com/2008/WORLD/europe/05/22/scope.project

- SketchFab

  https://www.sketchfab.com

- Learning WebGL

  http://learningwebgl.com

- Skyboxes

  learningthreejs.com/blog/2011/08/15/lets-do-a-sky/

- Three.js Revisions

  https://github.com/mrdoob/three.js/wiki/Migration

- Pointer Lock

  http://www.html5rocks.com/en/tutorials/pointerlock/intro/

  https://developer.mozilla.org/en-US/docs/API/Pointer_Lock_API

- Express JS

  http://expressjs.com/guide.html

  http://codetype.wordpress.com/2012/07/10/some-node-js-and-express-js-beginner-help/

- Server Hosting and Node JS

  http://cuppster.com/2011/05/12/diy-node-js-server-on-amazon-ec2/

  http://www.bennadel.com/blog/2321-How-I-Got-Node-js-Running-On-A-Linux-Micro-Instance-Using-Amazon-EC2.htm

-
-
-

References

ALLERGIC, JSWISHER, K. E. A., 2013. Mozilla developer network: Webgl. https://developer.mozilla.org/en-US/docs/WebGL.

DESPOULAIN, T., 2012. Hexgl. http://hexgl.bkcore.com.

I. FETTE, A. M., 2011. Websockets. http://www.websocket.org/aboutwebsocket.html, December.

JACOB, B., 2012. Fitc spotlight javascript: Introduction to webgl. http://people.mozilla.org/ bjacob/webgl-spotlight-js-2012/.

KANKAANPAA, H., 2004. Doing gravity right @ONLINE. http://www.niksula.hut.fi/ hkankaan/Homepages/gravity.html.

LECOLLINET, G., AND LECOLLINET, F., 2012. Browserquest. http://www.littleworkshop.fr/browserquest.html.

MARRIN, C., 2011. Webgl specification. https://www.khronos.org/registry/webgl/specs/1.0/.

PEREZ-FADON, D., 2012. Dcubic. http://www.dcubic.net/html5/.

RUSSEL, K., AND MO, Z., 2012. Graphics programming for the web: Webgl. ACM SIGGRAPH 2012 Course Slides, August.

VANIK, B., 2012. Acm siggraph presentation: Multiplayer javascript/webgl voxel world game. https://github.com/benvanik/blk-game.

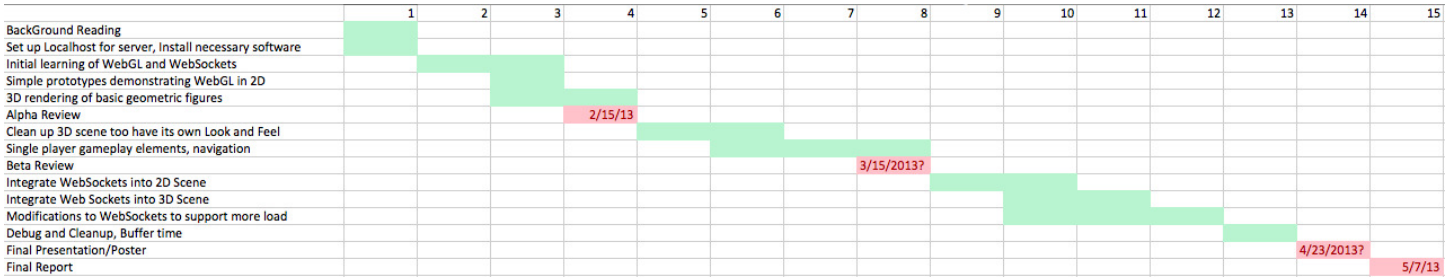

Figure 2: *Gantt Chart Tentative Schedule*