3D TRACKING SIMULATION OF ORBITING SATELLITES

A Project Report

Presented to

The Faculty

of the Department of Computer & Electrical Engineering/Computer Science

California State University, Bakersfield

In Partial Fulfillment

of the Requirements for

Senior Project

in

Computer Science

By

Doney Peters | Ivan Cisneros | Joey Hubbard | Thang Hin

May 2020

ABSTRACT

3D Tracking Simulation

of Orbiting Satellites

By

Doney Peters

Ivan Cisneros

Joey Hubbard

Thang Hin

While advances have been made in web applications that handle satellite tracking and imaging, little effort has been spent on creating highly functional and immersive simulations of the orbits of these satellites. The aim of this project is to create an efficient satellite tracking web application that will give users an accurate representation of the positions of satellites and allow users to see which satellite(s) are passing over at their present location in real-time. In addition, this web application will include a simulation using the Unity Real-Time Development Platform. Satellite tracking has only grown more complex over the years as more and more satellites are launched into space, with sources estimating at more than 1800 satellites currently in orbit. As such, current satellite tracking applications are inefficient and outdated. This project will rectify these deficiencies and will present users with a simple web-based application of the orbits taken by satellites.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

LIST OF TERMS

TLE	Two Line Element
NORAD	North American Aerospace Defense Command
SATCAT	Satellite Catalog (number)
SGP	Simplified General Perturbation
API	Application Programming Interface
JSON	JavaScript Notation Object Notation
ECI	Earth Centered Inertial
dll	Dynamic-Link Library
MVC	Model-View-Controller
.NET	Microsoft Web Application Framework
Azure	Microsoft Azure Cloud Computing Platform & Services

Chapter 1

Introduction

On October 4th, 1957, The Soviet Union launched the first ever satellite to achieve an elliptical low Earth orbit. Just 60 years later, the number of satellites in orbit is greater than one thousand. These satellites serve many purposes, from communication and Earth observation, to telescopes and space exploration. Satellites have become paramount to our global digital infrastructure. The challenge to keeping an object circling the globe has only grown more complex now that there are more satellites in space to avoid. The goal of our project is to create a useful and real-time satellite tracker that will display the position and path of all satellites in orbit.

Satellites are mostly tracked with computer programs using what are known as TLEs or Two Line Elements. TLEs are text files that contain two lines of characters and digits necessary to identify, and accurately track where a satellite may be at a specific time of day to a certain degree of accuracy. These files can be gathered from numerous repositories and websites, however, we will be retrieving our data from www.space-track.org, which is a government contractor managed satellite repository created by the United States Air Force. This website has an API (Application Programming Interface) our web application will be able to connect to in order to retrieve the TLE data of the satellites that will be tracked (admin@space-track.org, "Help Documentation"). Upon request, the space-tracker interface will return TLE data of the specified satellites in the format of a JSON(JavaScript Object Notation) file.

Once satellite data files have been retrieved from the Air Force repository, they will need to be parsed in order to be properly utilized. Part of the challenge of our project will be to convert this TLE position information into data that can be rendered into our simulation in a manner that will accurately represent where each satellite is located in relation to the Earth. We plan to write algorithms based on simplified perturbation models, which are mathematical models used to calculate space objects relative to the ECI(Earth Centered Inertial) coordinate system ("About Aerospace Coordinate Systems", 2019). Once the mathematical data for a celestial object has been properly parsed, its 3D simulated location is ready to be shown to the user of the web application.

The 3D simulation is going to be rendered by using the Unity engine, a popular 3rd party cross platform game engine created by Unity Technologies. The Unity engine will permit the project to be flexible since it allows the utilization and combination of three languages; JavaScript, boo, and C#. Not to mention, the Unity engine has a feature that allows exporting to webGL, a JavaScript API for rendering interactive 3D graphics on a web browser. In addition to that, Unity's documentation has a section on how to develop a WebGL platform using the Unity engine. Therefore, the development tools are all compatible since Unity provides support for a WebGL platform development.

However, this is not without constraint. There are several constraints in using Unity engine to develop a WebGL platform. An example of a constraint is JavaScript do not support multi-threading since JavaScript views the browser as a single thread. As a result, WebGL platform developed through the Unity engine can not take advantage of threads to speed up performance or utilize threads in scripted codes and managed dlls (Dynamic-Link Library).

Aside from the simulation, the web application itself will be written using the Microsoft .NET Core framework. The server side of the application will be written using the C# language and organized using MVC (Model-View-Controller) architectural pattern. The client side of the program will be written using a variety of standard web languages, such as HTML, CSS, JavaScript, in addition to utilizing Microsoft Razor pages for some data retrieval from the server.

Another feature is the utilization of the MongoDB program, which is an open source database management system, in order to store TLE data. This is an important component to the functionality of this project as it will help in the organization and accessibility of large amounts of data. As there are hundreds of satellites orbiting the Earth, it is necessary to be able to store and access information, such as JSON objects from the Space-Track API, through a reliable and accessible DBMS.

Many challenges will be faced while working on this project. The main set of challenges that are expected include proper distance simulation scaling in Unity, accurate mathematical calculations when acquiring satellite coordinates using the TLE data, and project complexity time constraints. Additionally, one last potential problem that will be faced is the learning curves when working with new frameworks and a new database management system. Learning the ins and outs of how these function and how they can be utilized to reach the solution desired.

In view of the fact that there has been little effort in the creation of highly functional and immersive simulations of the orbits of satellites, the concern of this project is specifically aimed towards addressing such concerns and creating a multi-featured, intuitive, user experience.

Chapter 2

Project Architecture

The high level layout for this project can be broken down into four major components: the database storing the two line element files needed to calculate the coordinates of the celestial objects, the server side module that requests and receives specified TLE data from space-track.org and parses the file, the server side module calculates the actual celestial Cartesian coordinates, and, finally, the front end encompassing the Unity simulation experience in addition to satellite object information. A high level diagram depicting the project is shown below:

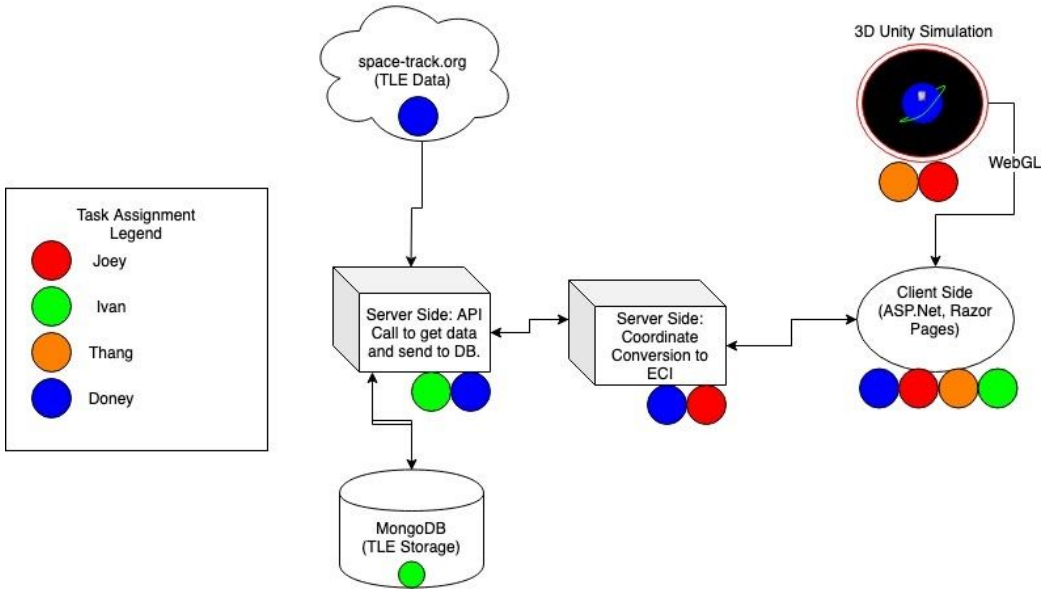

Figure 1. Flowchart of the main components of our project.

Perhaps the key component of the entire project are the celestial two line element files. As stated previously, these files contain a comprehensive list of all active satellite TLE data. Ironically, a single TLE entry inside this file contains three lines of data. These lines follow a strict format as defined by NORAD SATCAT and NASA. The first line, "line 0", contains a twenty-four character satellite name. The specifications of the next two lines are given in table 1 and table 2. The following figure shows the TLE data for the International Space Station.

```
ISS (ZARYA)
1 25544U 98067A   19350.13922549 -.00000461  00000-0  00000+0 0  9994
2 25544  51.6422 179.2595 0007472  40.7843  56.7759 15.50118763203489
```

Figure 2. An example of a Two-Line Element

Table 1. Two-Line Element Set Format Definition, Line 1

Field	*Column*	*Description*
1.1	01	Line Number of Element Data
1.2	03-07	Satellite Number
1.3	08	Classification
1.4	10-11	International Designator (Last two digits of launch year)
1.5	12-14	International Designator (Launch number of the year)
1.6	15-17	International Designator (Piece of the launch)
1.7	19-20	Epoch Year (Last two digits of year)
1.8	21-32	Epoch (Day of the year and fractional portion of the day)
1.9	34-43	First Time Derivative of the Mean Motion
1.10	45-52	Second Time Derivative of Mean Motion (decimal point assumed)
1.11	54-61	BSTAR drag term (decimal point assumed)
1.12	63	Ephemeris type
1.13	65-68	Element number
1.14	69	Checksum (Modulo 10) (Letters, blanks, periods, plus signs = 0; minus signs = 1)

Table 2. Two-Line Element Set Format Definition, Line 2

Field	*Column*	*Description*
2.1	01	Line Number of Element Data
2.2	03-07	Satellite Number
2.3	09-16	Inclination [Degrees]
2.4	18-25	Right Ascension of the Ascending Node [Degrees]
2.5	27-33	Eccentricity (decimal point assumed)
2.6	35-42	Argument of Perigee [Degrees]
2.7	44-51	Mean Anomaly [Degrees]
2.8	53-63	Mean Motion [Revs per day]
2.9	64-68	Revolution number at epoch [Revs]
2.10	69	Checksum (Modulo 10)

Table 1-2. TLE format for line 1 and line 2.

It should be apparent by the tables 1-2 that there is a great deal of information provided within these packets of TLE data. For our project, the data inside the TLE will serve two functions:

1) data relevant to the position of the satellite will be used in the SGP calculations
2) data that may be of interest to users will be made viewable inside a display within the simulator

Once the TLE files are inside the Mongo database, it must be converted from text into numeric values. This process involves parsing the text, or splitting line 1 and line 2 of the TLE precisely into the substrings based upon the defined formats from NORAD SATCAT in Tables

1-2. For the purpose of calculating the satellite position, the necessary fields are the eccentricity, inclination, right ascension of the ascending node, the argument of perigee, and mean anomaly. These values are fed as input parameters into the SGP4 perturbation model, which calculates the position of satellites based on the TLE data.

The client side of the project will encompass many different frameworks to give the user the most enjoyable experience when interacting with the application. First off, the overall design aesthetic will be designed using the Twitter Bootstrap user interface library. When the overall look of the web page was discussed, it was decided that a minimalistic, material design would provide the best experience for the user. The Bootstrap framework greatly minimizes the complexity of this task with pre-designed navigation bars, lists, tables, and text fonts. The client user interface will consist of only a single web page containing the simulation in addition to a list of satellites currently being displayed and their corresponding coordinates.

Razor pages, part of the Microsoft .NET Core framework will be used to pass calculated satellite coordinates from the server side of the application to the client side. Razor pages allows a developer to write C# code into a web page and then at compile time, it is converted into HTML and Javascript. The retrieved coordinates are then displayed via text for the user to see. Additionally, the names of the satellite whose coordinates are calculated will be sent from the server to the client using the same Razor pages ViewData object technique. The names are displayed with their corresponding coordinates. However, a drawback exists to exclusively using the ViewData objects to request data from the server. Anytime a new request is made, the entire page must refresh in order for the web page to properly display the results of the request made by

the user. This issue can be overcome by using AJAX (Asynchronous Javascript and XML) calls to the server from the client.

The most important portion of the client interface, the satellite simulation, will be written using the Unity framework to render the Earth and all of the simulated celestial objects. The coordinates of each satellite will be passed from the ViewData object into a javascript variable which Unity will use as a reference point to determine where to render the object relative to the Earth. From this point, Unity will also need to know the position and velocity vectors of the object as well, which the server will be able to provide using a simple AJAX request. From these pieces of information, Unity will successfully render the object and properly simulate its orbit and speed.

Before being injected into the browser, the Unity project must also be given constants of gravitational pull of the earth, in addition to the moon, which affects how celestial objects orbit the planet. Once all of the renderings and gravitational constants are loaded into the Unity project, it is exported as a WebGL file, which includes three files, the HTML code, a directory containing all the javascript necessary to render the objects in the browser, and the CSS styling to properly show display the unity frame around the simulation. The HTML files are injected into the main web page of the client and the other two directories are included in the project file system. The user will see the simulation displayed in the center of the page with options to go to full screen to give a more focused experience.

Another major component of this project is the database for which the TLE data will be stored in and accessed from the client-side. MongoDB's document model serves as a great appliance into our project as it stores data in JSON-like documents, which the TLE data gathered

from Space-Track is formatted in. In order to fully utilize MongoDB into the project, it was decided that a cloud-based database would serve as a better option because it handles most, if not all of the complexity that comes with creating a database (Host, management, securing, etc.). This service is provided using MongoDB Atlas, which uses Microsoft's Azure cloud service that handles the server side of the database management system. Atlas handles much of the concerns as we progressed further into the project, such as its scalability, hosting, and any latency issues we face. Outside of this, it works very similar with traditional database management systems. Atlas also requires a sharded cluster to be set up before creating the database, which is made up of shards that contain subsets of sharded data and mongos that act as query routers that provide the interface between the client application and the cluster. It allows for scale reads and writes through several nodes. This is setup on MongoDB's website. Below is a diagram that illustrates this process:

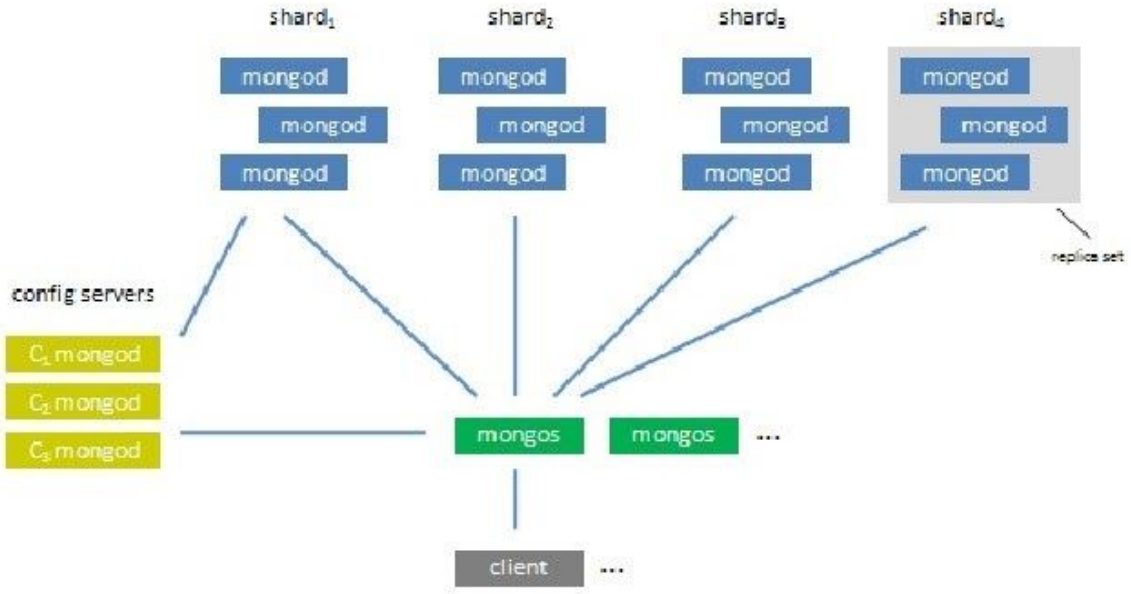

Figure 3. An architectural overview of MongoDB Atlas.

Once completed, the team utilized another feature of MongoDB – MongoDB Compass. Compass is a graphical user interface that allows the visualization and exploration of data stored in a database. These features include the implementation of data validation, adding/removing/updating data, and evaluating how the database is performing. With these tools, the team was able to create the database. Each member was issued admin privileges to the database, with the leading member of the database portion to oversee the management of the database. A connection-string was created that will allow each member of the team to establish a connection to the database. This is needed for the server side of the project to connect to the database in order to access the information stored. Once done, we pulled the TLE data from the repository in space-track.org and stored it into the database. To do so, we excavated and retrieved its TLE data, stored into JSON files, and added it to the database. As shown in Figure 1, the TLE data pulled from space-track.org will be called from the server side and sent to the database, where it will be stored so that it can be displayed on the client side through the simulation in Unity.

This concludes the components that make up the design of this project, as well as the first half of our project report. More will be added as we progress further into our project.

Chapter 3

Implementation

The implementation of this project began with the web application itself and the third party library, One_SGP4, to help us understand how to convert our TLE files into coordinate systems that can be interpreted by the Unity simulation that was to be built. Using the aforementioned library, we were able to feed in a single TLE file as a string, step through how the process of calculating and converting the numbers were performed, and, finally, see if the output actually matched with actual current locations of the satellite we were observing. From this point, we were able to swap out components starting with the conversion to unit and velocity vectors. The conversion process is displayed in the flowchart below:

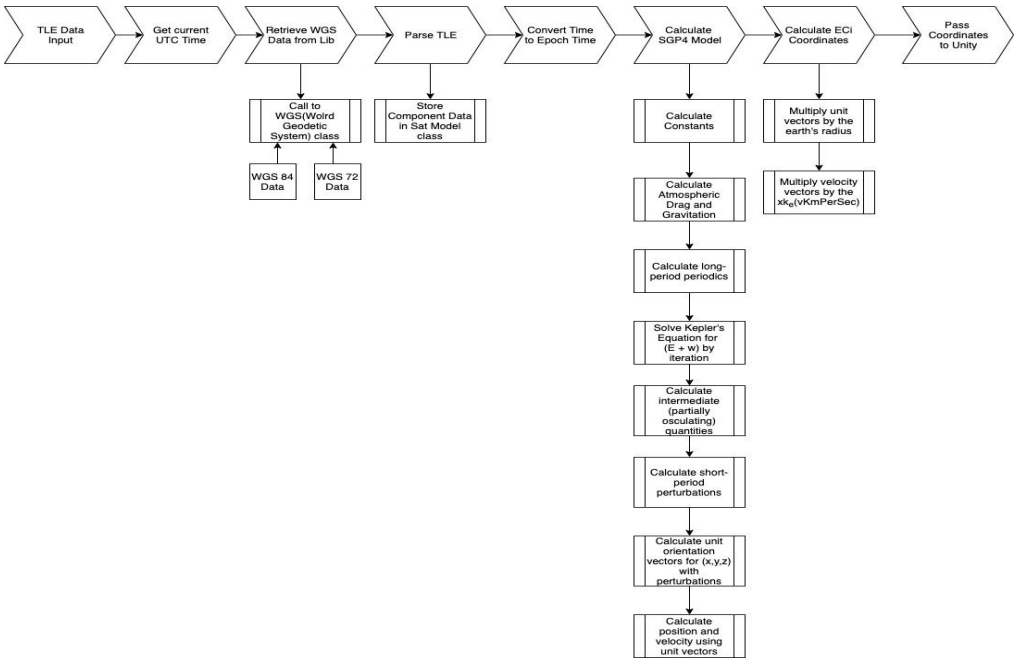

Figure 4. Flowchart of converting TLE to unit/velocity vectors.

This task was performed methodically and followed a similar structure to the library, in order to easily integrate the custom modules that were built with the third party conversion methods that would turn the vectors into coordinates. The decision to keep the third party conversion methods was due to the high complexity of the conversion methods, due to the time constraints and the other tasks that needed to be completed.

On the client side of the application, the javascript language was used to implement the fetching of the coordinates from the servers and displaying them for the users to experience. Additionally, Javascript was also used to pass the requested coordinates to the WebGL simulation using the unityInstance.sendMessage() method. Once the user clicks on and navigates to the simulation page, a razor page method is used to call the server to retrieve the names of all of the satellites stored in the database. Once all the names have been successfully loaded, the user is able to click on the name and activate the aforementioned Javascript methods that request and retrieve the calculations of the selected satellite. Upon a successful response from the server the coordinates are then displayed and updated every second underneath the satellite's name to show the real time latitude, longitude, and elevation of the celestial object. A snippet of the the data being displayed is shown below:

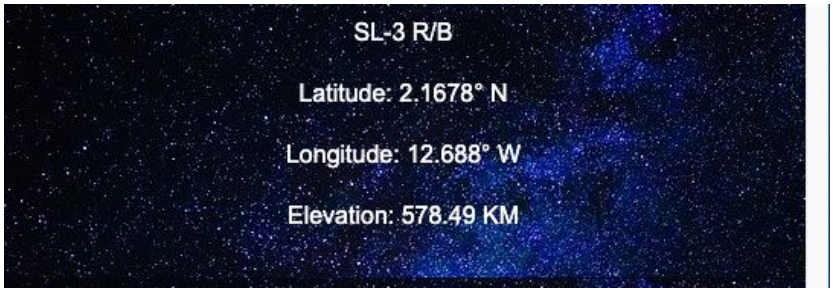

Figure 5. Depiction of satellite coordinates being displayed.

Then the unityInstance.sendMessage('GameObject','Method','JsonObject') method will send a JsonObject to the unity simulation's window, specifically to the associated GameObject along with the Method that is invoke by the unityInstance.sendMessage invocation. In this case the unityInstance.sendMessage('CreateSatellite','testSate','JsonObject') will invoke the GameObject CreateSatellite and the function testSate will be executed with the parameter JsonObject being passed into it. Afterward, the testSate function will be-serialized the JsonObject into usable variables.

The variable will then be utilized to compute spherical coordinates since the simulation uses spherical coordinate planes. The formula is x = Earth_radius + elevation * sin(latitude) * cos(longitude), y = Earth_radius + elevation * sin(latitude) * sin(longitude), z = Earth_radius + elevation * cos(latitude). The x, y, and z will create a Vector3 variable that will be utilized as a position for the satellite to be created on.

Lastly, the TLE data is stored into a cloud database in the form of JSON-like documents gathered from Space-Track. MongoDB Atlas served as a great appliance to achieve this, as it also uses JSON-like documents for its document storage. It allows for the handling of large data, which is essential to make our website work as there are thousands of satellites with thousands more of TLE information. And so, they are stored into the MongoDB database where they can be queried. When the query is made, the server will convert the TLE positional data, using the SPG4 perturbation model, into coordinates relative to Earth. It will then be called from the client side and be displayed through the simulation in Unity.

Chapter 4

Results

After the implementation phase, we were able to integrate all of the components of our project together and create a website that included a simulation that took in accurate coordinates based on the updated TLE data at space-track.org. Below are screenshots that display elements of our website:

Figure 6. Project homepage.

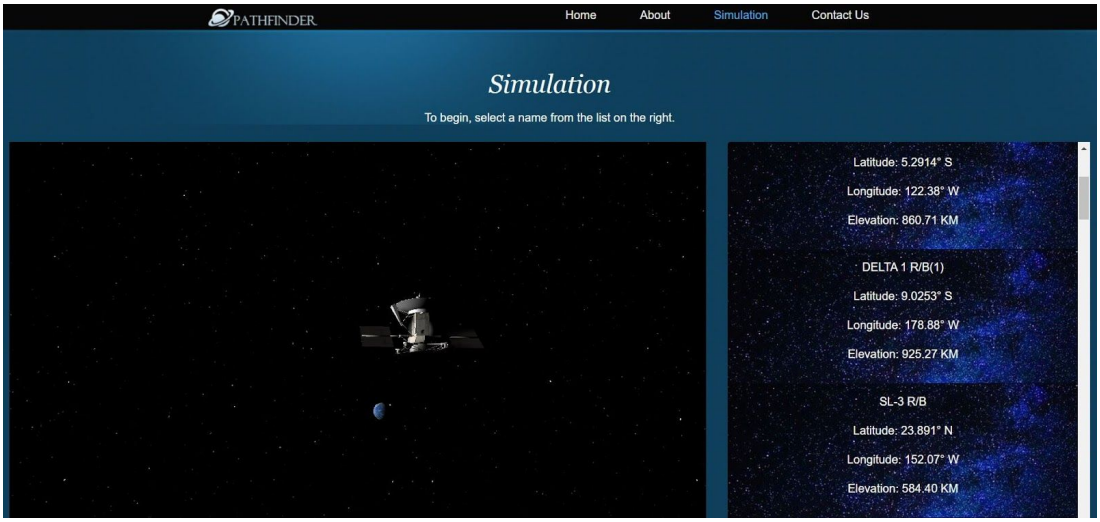

Figure 7. Unity simulation of satellite orbiting Earth, with its coordinates displayed.

The client side uses basic HTML 5 components, as well as JQuery and Bootstrap libraries that provides users with a clean and simple interface. One of the driving factors for our project was to make it user-friendly, as compared to other websites that have also done the same. Additionally, the Unity simulation was integrated into the webpage and converted to Javascript through a WebGL plugin.

With the server side, we were able to make it retrieve the TLE data from space-track.org through an API request. The TLEs are stored into the MongoDB Atlas database, where it will be passed to the client side of the web page. This results in the users being able to view the latitude, longitude, and elevation coordinates of a satellite and see it in real time through the Unity simulation. Users are able to select which satellite they want to see, and the information regarding its coordinates to determine the location relative to Earth.

Using Unity, we were able to successfully create a simulation of the satellites orbiting Earth. Each of the coordinates were passed through JSON objects into Javascript variables,

which Unity used as a reference point to determine where to render the object relative to Earth. The server then provided Unity the position and velocity vectors of the objects through AJAX requests. From this, we were able to render the object and properly simulate its orbit and speed.

Lastly, we were able to utilize MongoDB's document model to store the TLE data gathered from Space-Track as they were stored in JSON-like documents. MongoDB Atlas allowed the server side to access that information that was queried in the database. When a query is made, the server converts the TLE positional data into a series of latitude, longitude, and elevation coordinates that use the SPG4 model. The TLE data pulled from Space-Track is called from the server side and sent to the cloud database, where it is stored so that it can be displayed on the client side through the Unity simulation.

Overall, the results of our project were a success. We were able to create a website that allows for users to see the current position of satellites in real time. Each group member was able to contribute different aspects of the project that coalesced into a functioning web tracking application. Elements of JavaScript, Unity, DBMS, and others were utilized in the creation of the website. We were also able to market our project through the submission of our proposal to several organizations, and are eager to continue working on it in the hopes that it can be utilized in real world applications. Project Pathfinder gives users an immersive experience by allowing them to select satellites from our database to view their coordinates, and navigate through space within our Unity WebGL simulator.

Chapter 5

Conclusion

After a year of working on Project Pathfinder, a great deal of experience has been gained in creating web applications that implement Unity WebGL and the physics behind pinpointing a satellite's location. Each of us were responsible for complex modules within the project; having strong connectivity between each module was an imperative, otherwise, the overall experience for the user would suffer. We believe that the end result of our project is a fun and immersive application that gives people the opportunity to see satellites up close. Most people do not realize the scope at which they rely on satellites for their daily lives and they take this technology for granted. We hope that this project gives users a newfound or deeper love of space and space technology.

Though this is the conclusion, Project Pathfinder is not yet finished! We hope to incorporate in the future additional features such as a mobile app version, the ability to predict when a satellite will pass overhead, unique satellite textures, toggling sunlight effects, and time-scaling. We really enjoyed working on Project Pathfinder for the past year and we want to continue our work on it to see how far this path will take us.

References

- Vallado, D., & Crawford, P. (2008). SGP4 Orbit Determination. AIAA/AAS Astrodynamics Specialist Conference and Exhibit. doi: 10.2514/6.2008-6770
- “About Aerospace Coordinate Systems”. (2019). Retrieved from https://www.mathworks.com/help/aeroblks/about-aerospace-coordinate-systems.html
- "Orbital Propagation: Part I". (1994, September). Retrieved from https://www.celestrak.com/columns/v01n01/
- "Orbital Propagation: Part II". (1995, March). Retrieved from https://www.celestrak.com/columns/v01n04/
- Bate, R. R., Mueller, D. D., & White, J. E. (2015). *Fundamentals of astrodynamics*. New York: Dover Publications.
- Vitagliano, A. (n.d.). Retrieved from https://web.archive.org/web/20070907013516/http://main.chemistry.unina.it/~alvitagl/solex/MarsDist.html
- (n.d.). Retrieved from https://spaceflight.nasa.gov/realdata/sightings/SSapplications/Post/JavaSSOP/orbit/ISS/SVPOST.html
- admin@space-track.org, S. A. I. C. (n.d.). Help Documentation. Retrieved from https://www.space-track.org/documentation#api

- Technologies, U. (n.d.). Building and running a WebGL project. Retrieved from https://docs.unity3d.com/Manual/webgl-building.html