GSoC 2016 Proposal: Enabling Cesium for Liquid Galaxy

Liquid Galaxy Project

Personal Details

- Name: Abhishek V. Potnis
- Email: abhishekvpotnis@gmail.com
- IRC Nick: abhishekp - #liquid-galaxy on Freenode
- Github Username: abhishekvp
- Telephone: +91-9892197628
- Skype Username: abhishekp91
- Country of residence: India
- Timezone: IST - (UTC + 5:30)
- Primary language: English

Abstract

Enabling other applications for Liquid Galaxy would greatly benefit the open source community, thereby nurturing the open source culture and encouraging inter-community bonding. This project aims at enabling Cesium - an open source virtual globe for Liquid Galaxy. The Liquid Galaxy project started off by making use of Google Earth for the panoramic system. The idea of this project is to enable Cesium to run across the multiple displays, providing an immersive and a riveting experience to the users. This project focuses on endowing Cesium with features such as Camera Synchronization, Content Synchronization across the displays and Space Navigation Camera Control.

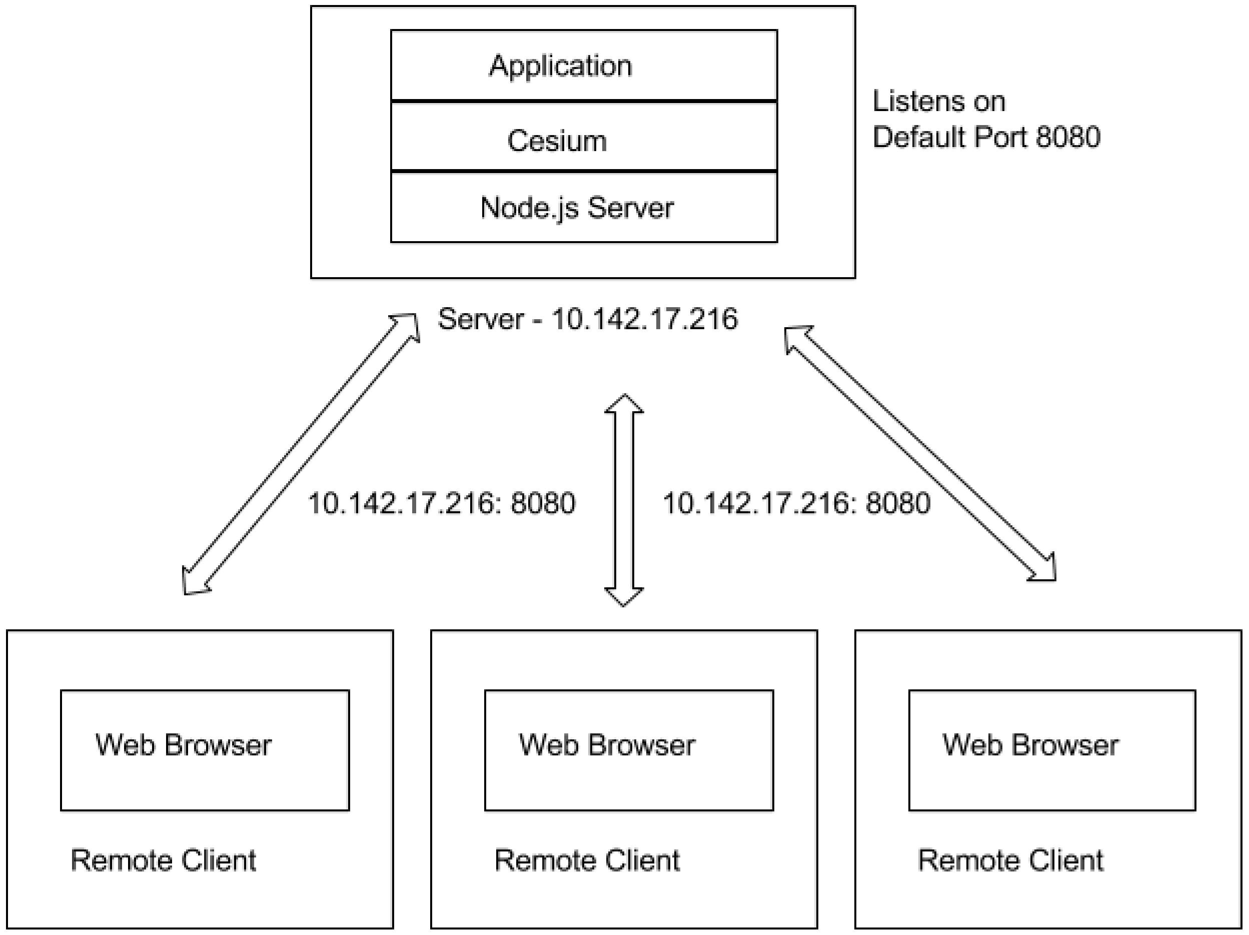
Implementation Outline

The Liquid Galaxy Project strives towards development of tremendously mesmeric panoramic systems. This project is geared towards bringing the compelling and immersive experience across multiple displays to the Cesium virtual globe. The primary features this project would focus on to endow Cesium with are as follows:

- Synchronization
  - Camera Synchronization between Master Cesium and Slave Cesiums
  - Content Synchronization between Master Cesium and Slave Cesiums
- Space Navigator Camera Control

Cesium

Cesium is a javaScript library to render 3D globes and maps using WebGL. A web server is required to host the Cesium files. This proposal assumes that the Node.js Web Server is used for hosting. The architecture of a typical Cesium based application is shown in the figure below.



Architecture of a Typical Cesium based Application

The Node.js Server hosts the Cesium files with the application. The remote clients can access the application from the web browser by pointing it to the URL of the application. In the above architecture the URL being 10.142.17.216:8080.

Synchronization

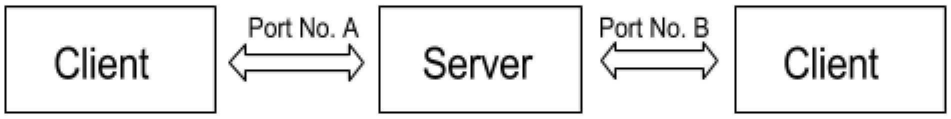
To ensure a riveting and immersive experience, it is essential for all the displays to be completely synchronized. Liquid Galaxy applications typically use a Master-Slave Architecture - with the Slaves orienting themselves and syncing with reference to the Master. Cesium is a virtual globe that is rendered on the client web browser from a NodeJS Server hosting the application. So synchronization between the Master and the Slave Cesium would essentially involve communication between the client web browsers. The web browser running on the Master would require to communicate to the web browsers running on the Slaves about the camera position and the content.

Browser to browser communication may be achieved using one of the techniques such as Web Sockets (Requires Server since connection is established between a server and s client), WebRTC (Requires Servers for Signalling, STUN and TURN) and others. This proposal describes a Proof of Concept Prototype for Camera Synchronization that makes use of WebSockets.

WebSockets

WebSockets help establish persistent connections between a server and a client. Both the server and the client can send data anytime, making it a full duplex bidirectional mode of communication.

Adapting WebSockets to our requirement of browser to browser communication would involve a server. So a client - browser would send a message to a server, a Node.js server in our case, and the server would forward the message to another client - browser.



A Typical Browser to Browser Interaction using WebSockets

WebRTC

WebRTC stands for Web Real Time Communication. WebRTC is an API definition catering to demands of Video, Audio and Data sharing between browsers. Our requirement of browser to browser communication can be fulfilled using WebRTC's RTCDataChannel. WebRTC has implemented the following API's:

- MediaStream - Get access to data streams, user media - Camera, Microphone
- RTCPeerConnection - Audio/Video Calling, Bandwidth Management and Encryption
- RTCDataChannel - Peer to Peer Communication of data

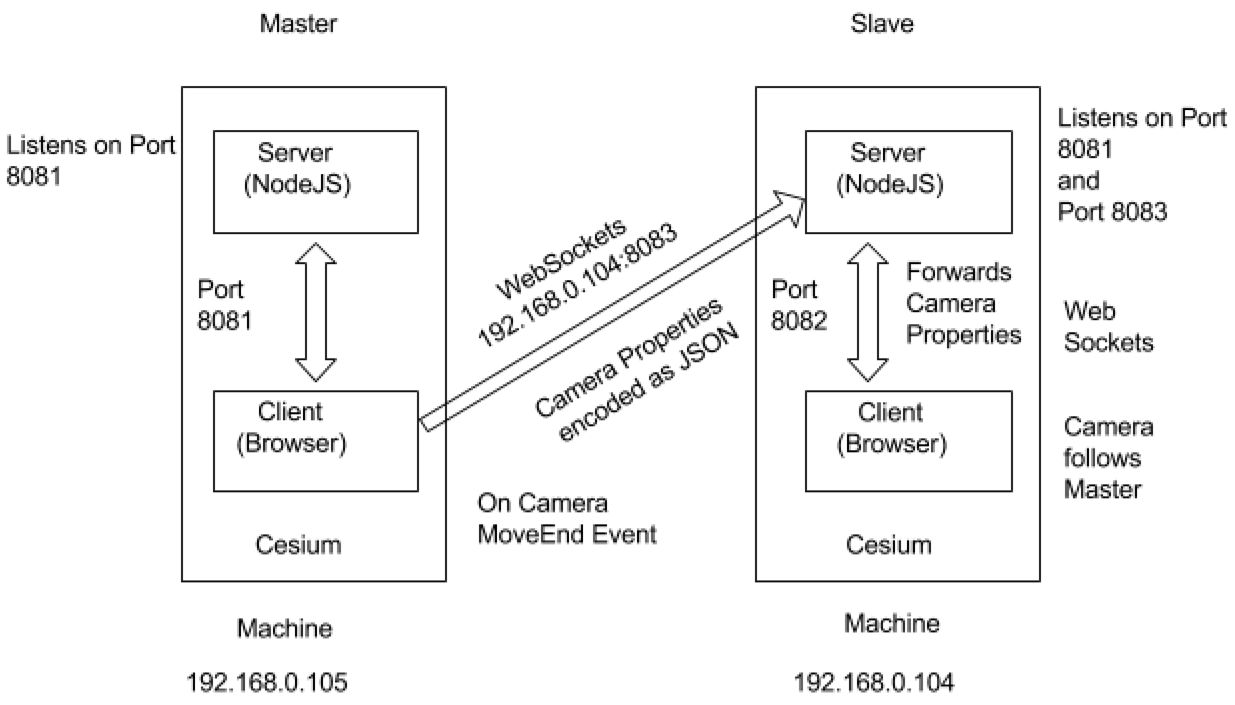
With most browsers now supporting WebRTC, it demands to be explored to fulfill our requirement.

Camera Synchronization

A Proof of Concept Prototype has been developed to demonstrate camera synchronization between the Master and the Slave Cesiums.

POC Prototype Architecture 1

This architecture assumes that the Cesium is up and running on both the Server and the Client. It makes use of WebSockets to achieve the Browser to Browser Interaction.



System Architecture Overview

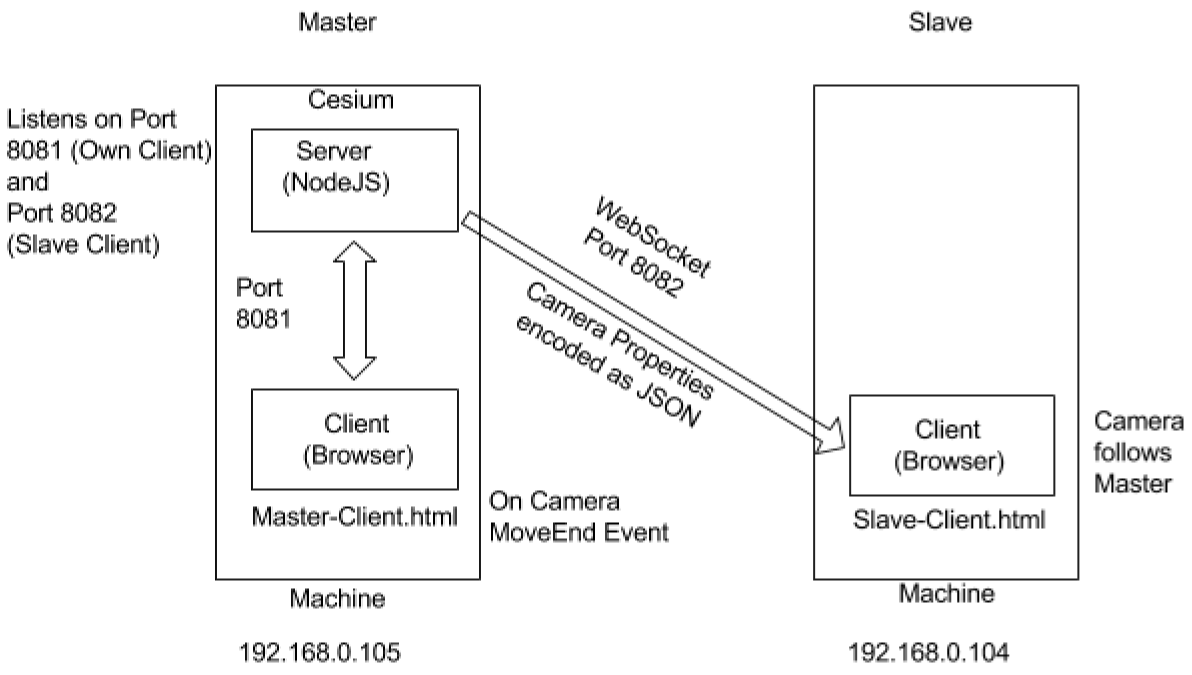
The Master and the Slave machines have Cesium up and running. An event listener at the Client on the Master, listens for CameraMoveEnd Event. On detection, it marshals the Camera properties namely Cartographic Coordinates - Latitude, Longitude and Height along with Heading, Pitch and Roll into JSON for easier processing. This JSON message is sent using a Websocket to the NodeJS Server on the Slave Machine. The Server on the Slave listens on two ports - one for incoming messages from the Master client and one from/to its own Client. On receiving the message, the Slave Server simply forwards the JSON to its own client(Slave Client) through a Web Socket. The Slave Client un-marshals and parses the JSON and has its Camera flyto the location extracted from the parsed JSON, thus following the Camera on Master machine.

Code Snippets: https://goo.gl/x9LZOG

The drawback of this architecture is that it requires even the slaves to have Cesium installed and running. This would affect scalability, when multiple slaves are connected to the master.

POC Prototype Architecture 2

This architecture assumes that only the Master has Cesium installed and running, while the Slaves simply access the the Master.

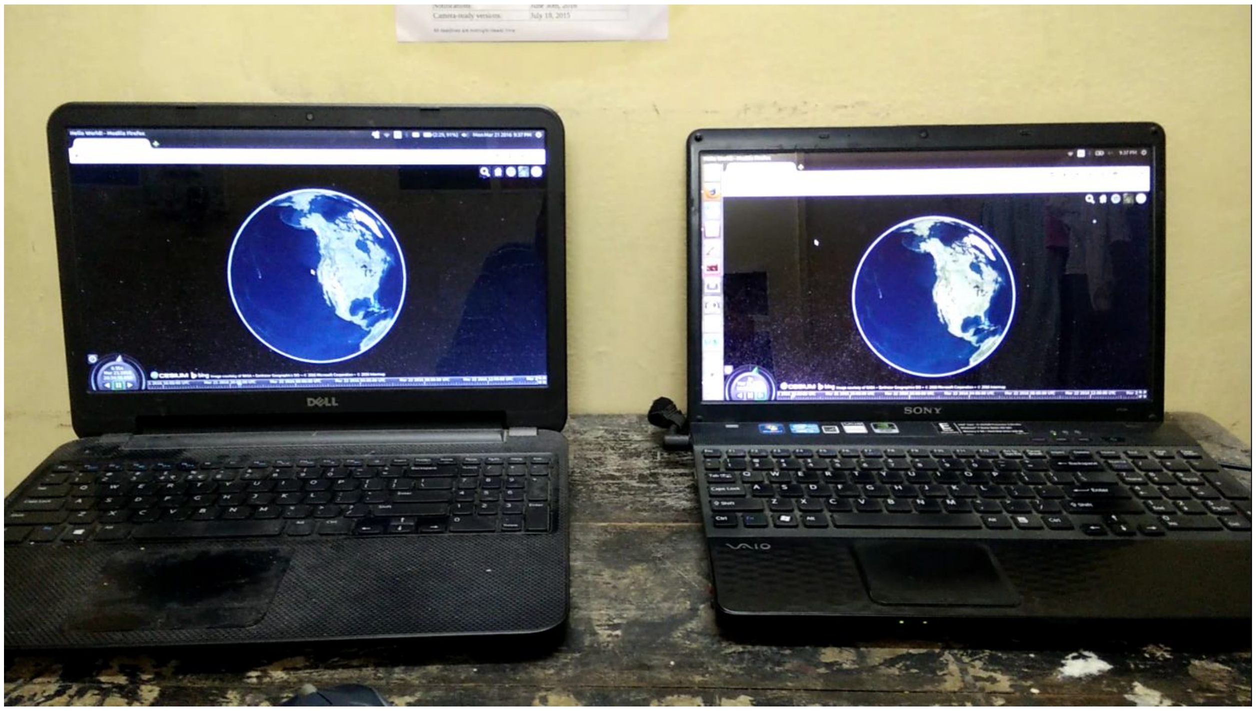


System Architecture Overview

The Master machine has Cesium up and running. An event listener at the Client on the Master, listens for CameraMoveEnd Event. On detection, it marshals the Camera properties namely Cartographic Coordinates - Latitude, Longitude and Height along with Heading, Pitch and Roll into JSON for easier processing. This JSON message is sent using a Websocket to the NodeJS Server on the Master Machine. The Master Server listens on two ports - one for incoming messages from its own (Master) Client and one from the Slave Client. On receiving the message, the Master Server simply forwards the JSON to the Slave Client through the Web Socket. The Slave Client un-marshals and parses the JSON and has its Camera fly to the location extracted from the parsed JSON, thus following the Camera on Master machine.

This architecture aids in scalability, since Slaves can simply connect to the Master without having to setup Cesium.

Code Snippets: https://goo.gl/2Pd6L9



Snap of the Setup

Video Demonstrating the Prototype: https://goo.gl/YDalID

The time-lag between the Slave Cesium to follow the Master in this prototype is noticeable and needs to be addressed. The prototype relies on the CameraMoveEnd event of Cesium to detect the motion of the Master Camera, and send over the new Camera properties to the Slave. The prototype demonstrates the slave following the master Camera, such that the Slave has the exact same view as that of the Master. Cesium's Camera methods such as |lookLeft()| and |lookRight()| can be used to give a panoramic view while syncing the Master and the Slaves.

The communication channels such as WebSockets, WebRTC and others need to explored, validated and analysed for response time and efficiency during the course of this project.

Content Synchronization

Along with the Camera, the Content too must be synchronized between the master and slave Cesiums. The changes from the master need to mirrored onto the slaves. WebSockets can be used as the communication channel between the master and the slaves. Other communication channels such as the RTCDataChannel of the WebRTC can be explored for efficiency.

An instance of ‘content’ in Cesium is termed as an |Entity|. A group of entities can be stored as an |EntityCollection|. The entities rendered on the globe form the content. Cesium’s base widget |viewer| has the |entities| as its data member. |entities| is essentially a |EntityCollection|.

Approach

An instance of |EntityCollection| can be sent using WebSockets or other communication channel to the Slaves from the Master. The Slaves on receiving the instance can simply add the |entities| to their respective |viewer|’s. An event listener on the Master can listen to the |CollectionChanged| event. The |collectionChangedEventCallback| returns an |Entities| Array that is the an Array of |Entities| that have been updated, named as |changed|. This |changed| entity array can be sent from the Master to the Slaves. Each of the Slaves can simply traverse the |Entities| Array and |add()| the |Entity| to their respective |viewer|’s |EntityCollection|. The technicalities involved in sending the |Entities| Array or the |EntityCollection| instance may depend on the communication channel to be used and can be explored during the course of the project.

An alternative approach to this can be using Cesium’s GeoJsonDataSource. The Master may send the entities as GeoJSON or TopoJSON. Each of the Slaves may then |load()| the entities onto their respective |viewer|’s. A variation of this approach may also involve the the Slaves using |loadJson()|.

The above mentioned approaches and others if any, need to explored, analysed and validated for rendering quality, response time and efficiency, and the best approach may be selected during the course of the project.

Space Navigator Camera Control

Liquid Galaxy systems typically use a Gamepad for navigation and Camera Control. The idea here is to use a GamePad to interact with Cesium for navigation and control.

Approach

The HTML5 Gamepad API can be used to query events from the Gamepad. Cesium's |ScreenSpaceEventHandler| can be used to create custom event handlers for Camera Control. The custom event handlers listening on the events from the Gamepad using the Gamepad API can be used for navigation and control, thus providing the user with an immersive experience using the Gamepad. Cesium's tutorial on using the Camera can be used as a reference to tie the Gamepad events to the navigation of the space camera. This approach would need to be explored, analysed and validated for response time and efficiency during the course of this project.

Timeline

My semester ends on May 3, so I will be able to start working on the project from May 4. I have structured the timeline, such that I will complete most of the project coding before July 20, before my next semester begins. The timeline is as follows:

- Week 1 (May 4 - May 11)
  - Be clear about the objectives of the project by discussing them with my mentor
  - Research and discuss with my mentor, the communication channels to be explored for the project and the criterion for deciding the communication channel.
  - Explore, Analyse and Compare the Communication Channels:
    - WebSockets
    - RTCDataChannel - WebRTC
    - Others
  - Familiarize myself more and play with Cesium

            - Material, Primitive, Camera, Scene, EntityCollection, ScreenSpaceController, etc.

- Week 2 - 3 (May 11 - May 25)
    - Prototype, test and validate communication channels, considering the performance constraints using the already developed POC prototype for Camera Synchronization
    - Discuss with my mentor and finalize a communication channel to be used for the project
    - Start working on a prototype using the finalized communication channel with Camera Synchronization
        - Syncing and Aligning the Slave displays with Master display using:
            - |lookLeft()|
            - |lookRight()|
        - Establish a relation between the display screen size and the amount in radians to be used by the above two methods

- Week 4 - 5 (May 25 - June 8)
    - Code and test the Camera Synchronization module of the project
    - Discuss with my mentor and finalize the criterion for validating the Camera Synchronization.
        - Response Time
        - Rendering Quality
        - Time Lag
    - Explore the various approaches for Content Synchronization
        - Sending |EntityCollection| instance to Slaves
        - Sending content as JSON to Slaves
            - GeoJSON or TopoJSON using GeoJSONDataSource
            - Using |loadJSON()|

- Week 6 - 7 (June 8 - June 22)
    - Prototype, test and validate the approaches for Content Synchronization
    - Discuss with my mentor about the criterion for deciding the Content Synchronization.

        - Response Time
        - Rendering Quality
        - Time Lag
    - Finalize an approach for Content Synchronization
    - Code and test the Content Synchronization module of the project

- Week 8 - 9 (June 22 - July 6)
    - Complete the implementation of the Camera and Content Synchronization
    - Exhaustively test both the features together across displays of various sizes
    - Discuss the results of the tests with my mentor
    - Explore the HTML5 GamePad API

- Week 10 - 11 (July 6 - July 20)
    - Explore the ScreenSpaceCameraController and ScreenSpaceEventHandler
    - Work out the events to be detected from the Gamepad and their respective reflections/responses in Cesium with reference to the Camera
    - Start working on implementation of the Navigation and Control using the GamePad API and the ScreenSpaceEventHandler

- Week 12 - 13 (July 20 - August 3)
    - Complete the Navigation and Control using the GamePad feature of the project
    - Test the feature using a Gamepad
    - Integrate the code with Camera and Content Synchronization module
    - Exhaustively test the integrated code
    - Discuss the results of the tests with my mentor

- Week 14 - 15 (August 3 - August 17)
    - Work on documenting the code.
    - Work on documenting the research and the results of experimenting the prototype for future research.

- Week 16 -17 (August 17 - August 24)
    - Buffer Week to contain backlogs if any.

Availability and Commitment for the Project:

My semester ends on May 3, so I will be free during the summer till July 20, when my next semester starts. I will easily be able to devote about 49 hours per week (7 hours per day), till July 20. In July, after my semester starts, I will be able to devote about 32 hours per week (4 hours per day on weekdays and 6 hours per day on weekends).

Open Source Experience

I have been code contributing to Mozilla for quite some time now, by fixing bugs and improving the Mozilla Developer Network(MDN) Documentation. I have open sourced some of projects on Github. I successfully completed a Google Summer of Code Project for the Cesium Community last year, titled as NASA's Data Curtains from Space mentored by Mike McGann and Ryan Boller at NASA. I have delivered talks on Open Source and Contributing to Mozilla at numerous Mozilla events during my tenure as a Mozilla Representative. My name has been included in the "about:credits" section of the Firefox web browser for my contributions. My name has also been included on the Mozilla Monument outside Mozilla's space at San Francisco for my contributions. I was also invited to attend the Mozilla Summit 2013 at Santa Clara, San Francisco, USA; with the entire travel and stay being sponsored by Mozilla.

Academic Experience

I have completed my Bachelors in Computer Engineering from University of Mumbai. I am currently pursuing my M.Tech. - Ph.D. Dual Degree in Geoinformatics from Indian Institute of Technology Bombay, with my area of research being Stream Reasoning over Geospatial Semantic Sensor Web.

Why Me

Being a contributor to the Mozilla project for almost 4 years now, I am familiar with the open source culture. I am experienced in working with Cesium, having successfully completed a Google Summer of Code project for the Cesium Community last year. I strongly believe that my passion, inquisitiveness and my experience would help me do justice to this project and see it to its timely completion.