Expertise
and insight
for the future

Saugat Awale, Jaya Kumar Bidari

Satellite Visualization in Web browsers

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

18 March 2020

Author Title Number of Pages Date	Saugat Awale, Jaya Kumar Bidari Satellite Visualization in Web browsers 45 pages 18.03.2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen

‘Satellite visualization’ refers to representing the real-time position of the satellite in its orbit with respect to earth’s surface as accurately as possible. Traditionally, the satellite’s graphical representation is executed in single-user desktop applications which make it difficult for multiple users in different workstations to view the same aerospace simulation of a satellite. This study researches the solution to the aforementioned problem.

In this work, a brief theoretical background regarding the physics of satellite propagation is discussed. After the discussion, each column of a TLE is defined. When the geometry of orbit of a satellite is well understood, a web-based application is proposed to visualize the same trajectory of a satellite in multiple workstations. The planning section describes the architecture of the proposed web application. It also lists the tools and technologies used for the project and explains the benefits of each tool. In the analysis part, error of time and coordinates is calculated. The error lies within the margin as defined by the satellite operators.

As a result, the satellite was successfully represented in a web browser and the same
Satellite telemetry could be viewed from multiple workstations. It is concluded that the challenge to view the same visualization in multiple workstations using a web browser is solved.

Keywords	TLE, Telemetry, SGP4, Cesium.js, Passes, CZML

Contents

List of Abbreviations

TLE Two-Line Element

API Application Programming Interface

OBC On-Board Computer

PCM Power Control Module

EU European Union

SAR Synthetic Aperture Radar

RADAR Radio Detection and Ranging

NASA National Aeronautics Space Administration

SGP4 Simplified Perturbations Model 4

STK System Tools Kit

RDP Real-Time Dynamic Processor

ISS International Space Station

ECI Earth Centered Inertial

ECF Earth Centered Fixed

1 Introduction

With the advent of modern technology in micro-electronics, there is a rise in the number of small satellite launches. Small satellites are defined as those satellites with a mass of 10 to 500 kilograms. These have become a competitor to large satellites due to its lower costs. Consequently numerous satellite-based companies have been established. Typically, the small satellites are launched by the organizations for remote sensing. Only a few percent of the satellites are launched to capture earth observation data [1] and ICEYE OY is one such start up organization.

Since this project is a joint effort by two authors the project's work is divided such that author Saugat Awale has conducted and prepared the report for Section 2 and Section 4.1, while Jaya Kumar Bidari has conducted and prepared the report for Section 3, Section 4.2 and Section 4.3 .

ICEYE OY is a satellite-based service provider which is preparing to launch its own satellites in lower earth orbit. A lower earth orbit is an orbit around the earth with an altitude of 2000 kilometres or less and an orbital period between 84 minutes and 127 minutes [2]. Operators in the ground segment team at ICEYE faced challenges in making aerospace simulation of the space flight available in multiple workstations and controlling the access to specific telemetries received from the satellite. This thesis has proposed a solution to the problem by making the simulation available in web browsers rather than traditional desktop applications.

Using a web application is beneficial to using a desktop application because it rids the user of dependence on software specifications which might also include the software's license. Additionally the user would have to manually

install the application software and configure the parameters accordingly to preview the same simulation, which would be costly and time consuming. On the contrary, using a web application will only require the user to have a functional network connection and an internet browser which is not a demanding requirement in the context of recent technological infrastructure.

Additionally, the operators will be able to control the display of the parameters. The libraries used in this thesis are completely open source which drastically decreases the cost of implementing the proposed solution. The web application is primarily based on Javascript which makes it relatively compatible to implement on the server side as well using node.js.

The aim of the web application is to facilitate the employees at ICEYE OY for efficient telemetry processing and satellite visualization. This thesis discusses how to create such a web application and also the challenges to implement such a solution. It also takes into account the security measures that need to be enforced and possible improvements which could be attributed for future developments.

2 Introducing orbits and Cesium.js Library

2.1 Orbit Theory

An orbit is defined as a curved trajectory which is caused by the virtue of gravitational force in an object. For instance, path of planet's trajectory around a star as shown in Figure 1 below;

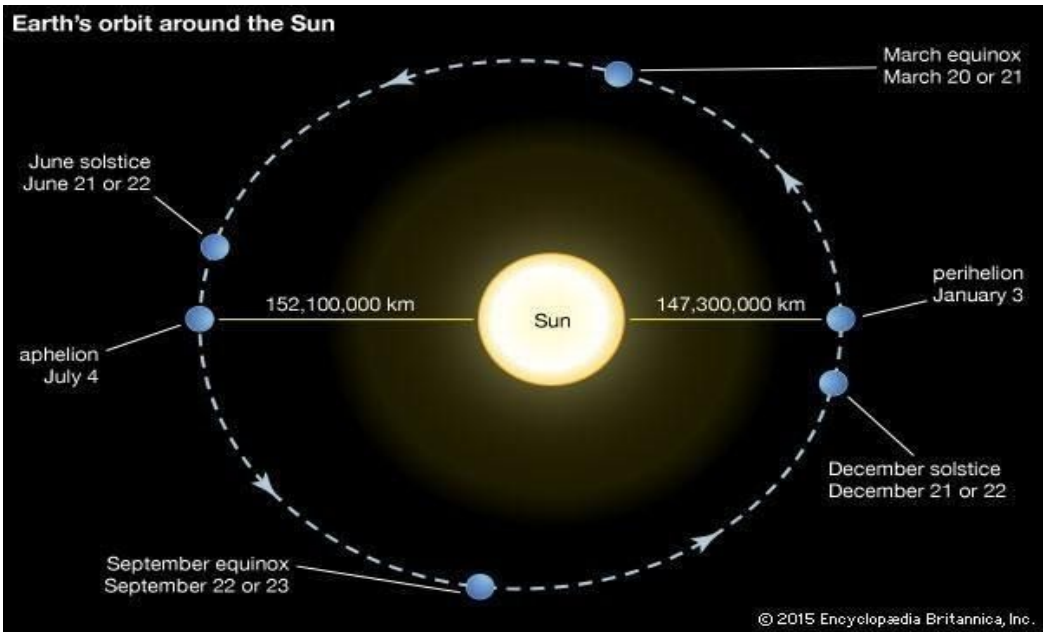


Figure 1. Earth's orbit around the sun. Reprinted from Encyclopedia Britannica [3].

The figure above shows that the orbit of a gravitationally bound object is not a perfect circle. Ideally, orbit refers to a regularly repeating trajectory which is known as Kepler's orbit. Kepler's orbit approximately follows an elliptical path and is described by Kepler's laws of planetary motion [4]. For simplicity purposes Einstein's relativistic effects on the body that is in the orbit will not be invoked.

A satellite also revolves around the earth in a keplerian orbit. Therefore, it is required to understand the geometry of the keplerian orbit to predict and visualize the position of the satellite. It should be noted that the motion of a Keplerian trajectory occurs in a three dimensional plane. To explain the motion in a three dimensional plane, two parameters are needed which are position and a vector velocity. Three components are required to describe each parameter. Hence we need a total of six elements to describe a Keplerian orbit.

These six elements are conveniently referred to as orbital elements. Below, Figure 2 visualizes a Keplerian orbit along with its labelled components.

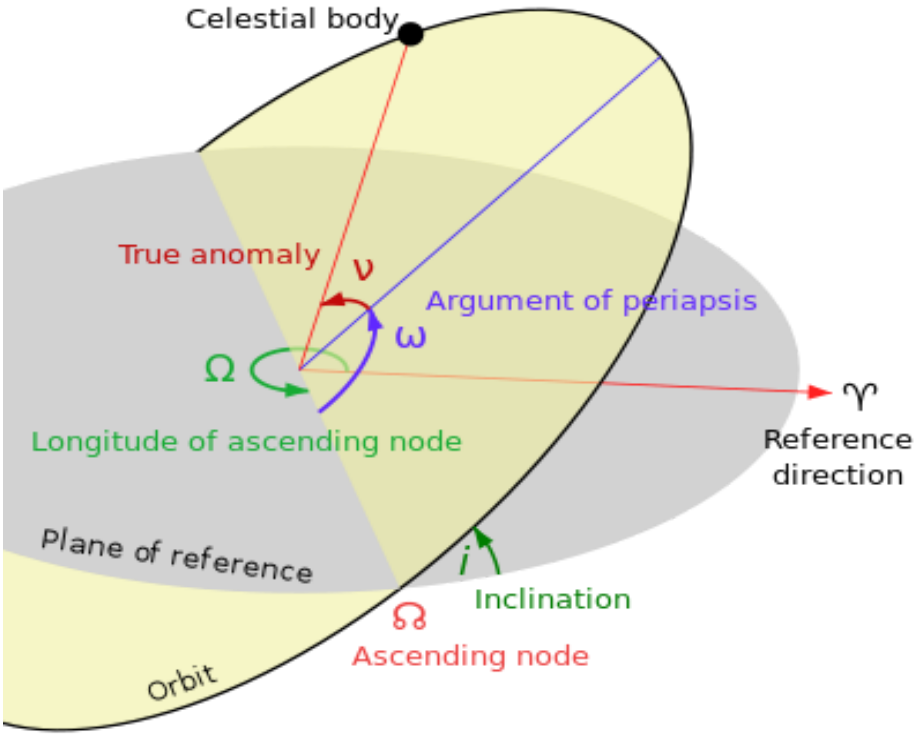


Figure 2. Keplerian Orbital Elements

As can be seen from Figure 2, a Keplerian orbit will have two sets of elements. Each set for two different bodies which depends on which body is chosen as a reference point. Following are the elements which define an orbit :

1) Eccentricity (e) - Eccentricity describes the elliptical shape of an orbit. It can be used to calculate for instance how elongated an ellipse is compared to a circle. Although eccentricity is not labelled in the figure it can be calculated by Eq. 1 [5];

$$e = (r_a - r_b) / (r_a + r_b) \quad (1)$$

Here r_a refers to the radius of apoapsis which is the farthest of the orbit from the center of the mass and rp refresh to the radius of periapsis which is the closest distance.

2) (a) – Semimajor axis determines the size of the orbit. It is the sum of length of the apoapsis and periapsis divided by two. It can be calculated mathematically from Eq. 2 [5] :

$$a = (r_a + r_b) / 2 \qquad (2)$$

3) Inclination (i) – Inclination is the vertical tilt of the ellipse with respect to the reference plane. Inclination is measured at ascending nodes.

4) Longitude of the ascending node (Ω) – It measures the horizontal orientation of ellipse's ascending node (the point where orbit passes upward with respect to the reference plane)

5) Argument of periapsis (*ω*) – It describes the orientation of the ellipse in the orbital plane of reference.

6) True anomaly (v) – True anomaly defines the position of the body in an orbit. It is measured as an angle between periapsis and the position of the body in orbit.

2.2 Defining TLE

A two-line element (TLE) is a standardized data format in which the orbital elements are encoded along with other additional data such as Satellite's Name, Satellite
Number. A TLE is associated with a specific point in time which is known as the

epoch. The epoch when we received the TLE is also one of the elements included in TLE. It can be understood that the TLE contains information which is vital for ground

operators such as the satellite's launch date. Following is an example TLE of a satellite.

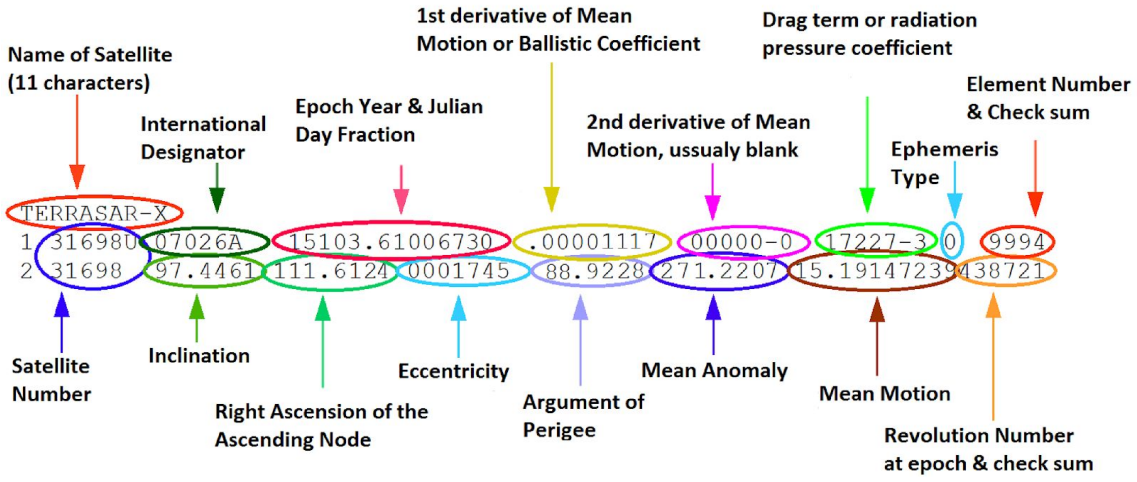


Figure 3. A sample TLE with labelled parts extracted from Spaceflight. [6]

The elements labelled in the above diagram will be briefly discussed:
Name of Satellite – TERRASAR-X is simply a name associated with the satellite.

International Designator – (07026A) The 07 refers 2007 as launch year. During 26th tally it was the 27th launch of the year and the alphabet ’A’ shows that it was the first object to be launched from the space shuttle.

Epoch Date and Julian Date Fraction – Julian day fraction is defined as the number of days passed in a year. For instance, the date above shows 15 as the epoch year (2015) and 103.61006730 as the Julian day fraction meaning approximately halfway between 103rd and 104th days after January 1, 2015. The exact date/time can be computed as follows:

103.61006730 days

103.61006730 days – 103 = 0.61006730 days

0.61006730 days x 24 hours/day = 14.6416152 hours

14.6416152 hours – 14 = 0.6416152 hours

0.6416152 hours x 60 minutes/hours = 38.496912 minutes

38.496912 minutes – 38 = 0.496912 minutes

0.496912 minutes x 60 seconds/minute = 29.81 seconds

103 mod 3(month number) = 14 (day of the month)

Hence the exact date/time it references is 2015/3/14 14:38:29.81.

First Derivative Mean Motion – First Derivative mean motion is the daily rate of change in number of revolutions the object completes per day divided by 2.

Second Derivative of Mean Motion – This is a second order drag term which is used in the SGP4 predictor to model terminal orbit decay. The unit of this measurement is revolutions/day^3.

Drag Term – This element is also known as BSTAR drag term and models the aerodynamic drag on a satellite which is caused by the sparsely present atmospheric molecules in the LEO. The unit of this term is m^{-1} .

Element Set Number and Check Sum – It is a running count of all the TLEs that have been generated for that particular object. The counter is increased with time and after it exceeds 999, it reverts to 1. The final number is for the checksum of line 1.

Satellite Number – It is the catalog number USSPACECOM has designated for the object. Presence of ”U” at the end is an indication that it is an unclassified

object.

Inclination (degrees) – It is the angle made by the equator and orbital plane which is referred to by ' i ' in Figure 2.

Right Ascension of The Ascending Node (degrees) – It is the angle between the vernal equinox and the point in which the orbit crosses the equatorial plane which is represented by 'Ω' in Figure 2.

Eccentricity – It is a constant value which defines the shape of the orbit. The value provided in the TLE will be a mean value of the eccentricity.

Argument of Perigee (degrees) – It is denoted by 'ω' in Figure 2. Perigee as labelled in Figure 2 is a general term while perigee refers to an orbit around the earth.

Mean Anomaly (degrees) – It is the angle measured from perigee to a circular orbit with radius equal to the semi-major axis.

Mean Motion – This element denotes the number of orbits per day that the object completes.

Revolution Number and Checksum – This refers to the orbit number at the epoch time when the TLE was received. The final digit is the checksum for line 2.

2.3 SGP4 algorithm for orbit propagation

The quantities mentioned above in section 2.2 can be used to predict an objects' position in 3 dimensional space at a given time. This way an orbit can be predicted there by propagating the object [7].

The TLE that is to be input in the SGP4 algorithm should be updated periodically to remove errors that will have accrued as time passes. Gravitational forces from other heavenly bodies such as the sun and moon perturbs the orbiting body. Frequently updating the TLE generates a more accurate orbital path each time it is updated. The source of the TLE that will be inputted in the SGP4 algorithm will be from celestrak's website (https://www.celestrak.com/NORAD/elements/).

An http request will be sent to celestrak's API which will return us the required TLE.
For instance sending an http request to the url `https://www.celestrak.com/NORAD/elements/stations.txt` will return us with the list of TLE of the stations as follows:

`ISS (ZARYA)
1 25544U 98067A 19309.66249666 .00001325 00000-0 30962-4 0 99922 25544
51.6443 19.8013 0005962 231.8878 286.8452 15.50255982197202
ALTAIR PATHFINDER
1 42711U 98067LS 19309.32703528 .00174426 23814-4 34695-3 0 9993
2 42711 51.6274 278.7840 0004197 76.2284 283.9192 15.99498793141260
NSIGHT
1 42726U 98067MF 19309.38522670 .00036732 00000-0 21355-3 0 9996
2 42726 51.6332 311.2943 0005399 20.7093 339.4126 15.78099292139826
KESTREL EYE IIM (KE2M)
1 42982U 98067NE 19309.43626898 .00009475 00000-0 10036-3 0 9991
2 42982 51.6374 345.5676 0003905 325.0231 35.0507 15.64900840115781
ASTERIA

1 43020U 98067NH 19309.47317236 .00036656 00000-0 21951-3 0 9991
2 43020 51.6400 322.7110 0002078 7.5405 352.5627 15.77449346111868
DELLINGR (RBLE)
1 43021U 98067NJ 19309.31771099 .00009235 00000-0 96177-4 0 9992
2 43021 51.6385 345.5786 0001537 303.2352 56.8496 15.6537085011154
1KUNS-PF
1 43466U 98067NP 19309.37279651 .00028424 00000-0 21530-3 0 9999
2 43466 51.6363 342.8686 0002361 17.7379 342.3701 15.72145598 84843
UBAKUSAT
1 43467U 98067NQ 19309.32862040 .00014374 00000-0 14213-3 0 9994
2 43467 51.6369 351.6066 0001037 280.9572 79.1307 15.66146860 84739
BATSU-CS1 (IRAZU)
1 43468U 98067NR 19308.56163801 .00053029 00000-0 30277-3 0 9997
2 43468 51.6369 339.6250 0003418 27.9080 332.2104 15.78347821184809
CUBERRT
1 43546U 98067NU 19309.44363948 .00007633 00000-0 74145-4 0 9999
2 43546 51.6379 350.1260 0003818 318.0657 42.0046 15.67448452 74892
TEMPEST-D
1 43547U 98067NV 19309.44025997 .00008295 00000-0 10893-3 0 9993
2 43547 51.6390 5.1317 0005023 293.2596 66.7868 15.59464998 74374

RAINCUBE
1 43548U 98067NW 19308.67793543 .00014864 00000-0 15476-3 0 9998
2 43548 51.6368 0.7450 0005032 313.2404 46.8171 15.64800187 74815`

ISS's tle will be parsed and inputted in the SGP4 algorithm which will output the coordinates of the orbiting body with future timestamps.

2.4 Introducing CesiumJS

To be able to display propagated orbit in a web browser, the algorithm will need to be coded in JavaScript. Prior to that a visualization tool is required which will

render 3d models, globes and maps. This project will use CesiumJS for the aforementioned purpose.

CesiumJS is an open source JavaScript library which can be used to create 3D globes and maps. It can also be used for light weight aerospace simulations and smart cities plannings. For intents of this thesis, CesiumJS will be used only to display the orbital path of the satellite. This is the link to the CesiumJS project's javascript library https://cesium.com/index.html.

Firstly, the environment is to be set up as per CesiumJS's documentation. https://cesium.com/docs/ .

Below is the source code for the cesium Application:

```
<!DOCTYPE html>

<html lang="en">
<head>
  <meta charset="utf-8">
  <script
src="https://cesium.com/downloads/cesiumjs/releases/1.66/Build/C
esium/Cesium.js"></script>
  <link
href="https://cesium.com/downloads/cesiumjs/releases/1.66/Build/
Cesium/Widgets/widgets.css" rel="stylesheet">
</head>
<body>
  <div id="cesiumContainer" style="width: 700px;
height:400px"></div>  <script>
    Cesium.Ion.defaultAccessToken = 'your_access_token';
    var viewer = new Cesium.Viewer('cesiumContainer');
  </script>
</body>
</html>
```

A brief breakdown of the code:

This line includes the cesium.js library in the head section

```
<script
src="https://cesium.com/downloads/cesiumjs/releases/1.66/Build/Cesium/Cesium.js"></script>
<link
href="https://cesium.com/downloads/cesiumjs/releases/1.66/Build/Cesium/Widgets/widgets.css" rel="stylesheet">
```

This line creates an HTML element to hold the cesium widget

```
<div id="cesiumContainer"></div>
```

Cesium.js is heavily dependent on Bing imagery [11]. Hence, access tokens from ion accounts is required to access bing's images. The access token is to be provided to the cesium application as follows. A cesium ion account is to be created to receive the token.

```
Cesium.Ion.defaultAccessToken = 'your_access_token';
```

Finally, the top level Cesium widget is created with a variable name `Viewer` which uses the HTML element defined above.

```
var viewer = new Cesium.Viewer('cesiumContainer');
```

Before running the application locally all the npm packages need to be install with the command `npm install`

When all of the packages are installed, it should be run in the local server with the command `npm start`

After 'Getting Started' section is completed as per CesiumJS's documentation the following view will be visible in the web browser:

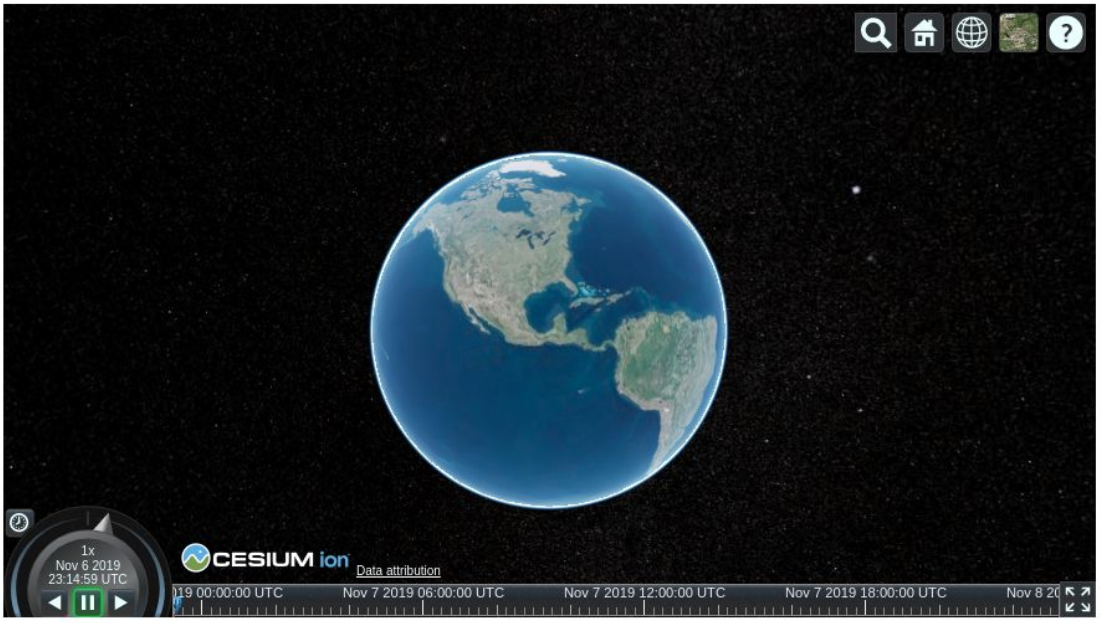


Figure 4. Screenshot of a basic cesium application

Primary models that are available in the view are an earth, a timeline and a clock. The satellite's model will be shown orbiting the earth in this cesium view.

3 Designing the web application

The web application will be a simple JavaScript file which will be loaded via nginx web server. The application will be available to clients via an http request. The javascript file will be responsible for loading the cesium modules and calculating satellite coordinates using the sgp4 algorithm.

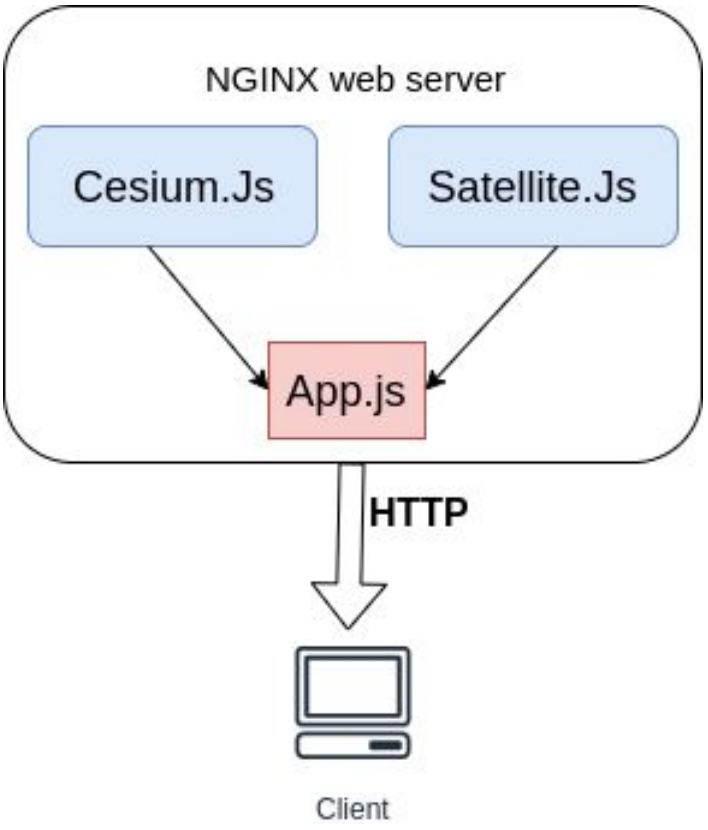


Figure 5. Architecture of web application

3.1 List of software tools used with their function

1) Cesium.js for creating 3D map environment

2) Nginx acts as a gateway between the application and the user

3) React.js handles the front-end operation of the application such as providing logging in feature

4) Node.js is the environment where the cesium application will be run

3.2 Architecture of the web application

As we can see in figure 5, a javascript file named app.js imports objects from Cesium library and Satellite library. App.js then creates a satellite model in cesium.js and assigns it an orbit which is calculated by satellite.js library. Nginx

will act as a proxy web server which redirects all the http requests to the port where the cesium application will be hosted. Nginx can also be used to perform SSL termination and balancing load generated by the users.[8]
ReactJs handles the UI section when users login.[9] NodeJS is the runtime environment[10] where CesiumJS application will be running which will also encrypt the information entered by the users and received by the application.[11]

4 Satellite Visualization in Cesium JS

CesiumJS contains its own models which are to be imported and displayed on the users' screen. Cesium already includes some gltf models. The goal of the project is to display a satellite model along with its orbit.

4.1 Representing a model in Cesium.js

There are two ways of representing a model in Cesium.JS. Either by importing models from the cesium library or by defining a model using czml format, which is a format which is exclusively created for Cesium.JS. Both of the methods will be used in this project to display the models. Cesium.JS's directory structure is to be understood before importing the cesium models. A sample terrain model will be displayed

4.1.1 Navigating Cesium.js Library

Following files and folders can be found in the application folder:
The app directory contains the following sub directories

Source/ : Includes application code and data.
ThirdParty/ : Third Party libraries which are required by CesiumJS to run
LICENSE.md : Consist of terms of services for the application

index.html : The main page which loads the app code and imports all the models

server.js : It is entry point file of the application which defines the configuration, dependencies and routes of the project

4.1.2 Displaying a sample model in Cesium.js

Among many other models present in Cesium.js, Terrain is one of the in-built models of Cesium.js library. Terrain models for instance include visualization of Mountain peaks, valleys and also terrains for water effects such as oceans, lakes and rivers. A server provides terrain data for Cesium engine.It requests and renders tiles as required on the basis of current camera position. In order to add terrain data, `CesiumTerrainProvider` module will be used. It will specify a url and some configuration options and then the specification will be assigned to the provider named `viewer.terrainProvider`.

In this example Cesium World Terrain tileset will be used and Cesium ion hosts it. “My Assets” includes it by default. In this case, `createWorldTerrain` helper function can be used to create the Cesium World Terrain tileset.

```
var viewer = new Cesium.Viewer('cesiumContainer', {
    terrainProvider: Cesium.createWorldTerrain()
});

// Load Cesium World Terrain
viewer.terrainProvider = Cesium.createWorldTerrain({
    requestWaterMask : true, // required for water effects
    requestVertexNormals : true // required for terrain lighting
});
```

`requestVertexNormals` and `requestWaterMask` are configuration options.They are optional and they tell Cesium to request extra data for lighting and water effects. They are set to false by default.

Once the terrains are included, one additional line is needed to make sure objects behind the terrain are correctly obstructed. The front-most objects will only be visible.

```
// Enable depth testing so things behind the terrain disappear.
viewer.scene.globe.depthTestAgainstTerrain = true;
```

Now, Terrain and animated water are displayed.
Below is the completed example of code for displaying a sample model of terrain in Cesium.js

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <script
src="https://cesium.com/downloads/cesiumjs/releases/1.66/Build/C
esium/Cesium.js"></script>
  <link
href="https://cesium.com/downloads/cesiumjs/releases/1.66/Build/
Cesium/Widgets/widgets.css" rel="stylesheet">
</head>
<body>
  <div id="cesiumContainer" style="width: 700px;
height:400px"></div>
```

```
  <script>
    Cesium.Ion.defaultAccessToken = 'your_access_token';
    var viewer = new Cesium.Viewer('cesiumContainer', {
        terrainProvider: Cesium.createWorldTerrain()
    });

    viewer.terrainProvider = Cesium.createWorldTerrain({
    requestWaterMask : true, // required for water effects
    requestVertexNormals : true // required for terrain lighting
    });
    viewer.scene.globe.depthTestAgainstTerrain = true;
  </script>
</body>
</html>
```

4.1.3 Introducing czml format in Ceisum.js

CZML format is a tyme-dynamic JSON format used to display graphical scenes.

The fact that it is in JSON format makes it convenient to use in a web browser. The format describes graphical scenes which are dynamically updated with time. This CZML format can be used to describe cesium models such as models , billboards, points, lines and other graphical primitives .It also specifies their dynamicity. The advantage of using CZML formats over cesium's library api is that the CZML format is data-driven. It allows the cesium viewer to display a rich scene and it does not have to be achieved programmatically.[12]

CZML format in cesium.js is comparable to KML format in Google. KML is also a data format which describes scenes in its respective environment. Some of the distinguishing characteristics of CZML are as follows:
1) It is written in JSON format.

2) CZML precisely describes the attributes of models, that are changed over time. For instance, a dot could be black for an interval of time and red for another specific interval of time. Additionally, czml format also gives the client for interpolating the position. For instance, if the position of a satellite is specified at two different instances of times, the czml format allows the client to represent the satellite in between the intervals of time. The interpolation is done using a library in-built interpolation algorithm. Furthermore, any property used to describe the satellite model can potentially be made time dynamic as will be the case in this project.

3) The czml models can be represented even when the czml document is partially available.

4) CZML formats can be easily parsed.

CZML Structure

Since, CZML is simply a subset of JSON, consequently a valid CZML document is also an authentic JSON document. Specifically, a valid CZML document consists of a single Javascript array where every object is denoted as a CZML packet. A CZML packet is a javascript object describing the graphical properties of a single object's graphical properties. For instance an aircraft. Below is an example of a sample of a CZML document which consists of two packets.

```
[
    // packet one
    {
        "id": "GroundControlStation"
        "position": { "cartographicDegrees": [-75.5, 40.0, 0.0] },
        "point": {
            "color": { "rgba": [0, 0, 255, 255] },
        }
    },
    // packet two
    {
        "id": "PredatorUAV",
        // ...
    }
]
```

The packets consist of `id` property which as the name suggests, identifies the object which it is describing. If an `id` is not specified, the cesium library will generate a unique identifier.

In addition to the `id`, Packets consist of more than one property which defines the actual graphical item in question. For instance, in the code above, the packet with id `GroundControlStation` object is specified a fixed position with latitude 40 degree, height as 0 and longitude of -75.5 degree. With the aforementioned graphical properties, a point is created which is blue in color. The outcome of this czml will be such that there will be a blue dot on the map of the earth at the specified location and it will be referred to with id `GroundControlStation`.[12]

CZML Structure - Intervals

Generally, a CZML's property is an array. Every array object is a JSON object literal which specifies the graphical properties for various time intervals.

```
{
    "id": "myObject",
    "someProperty": [
        {
            "interval": "2012-04-30T12:00:00Z/13:00:00Z",
            "number": 5
        },
        {
            "interval": "2012-04-30T13:00:00Z/14:00:00Z",
            "number": 6
        },
    ]
}
```

In the code sample above, `someProperty` represents any arbitrary property over two intervals. The first interval defines the `number` property from 12:00:00 to 13:00:00 UTC where the value of `someProperty` is five. The second interval defines the `number` property from 13:00:00 to 14:00:00 UTC where the value of `someProperty` is six.

The idea is to replace `someProperty` with `position` of the satellite and give coordinates of the satellite for each interval. The position of the satellite for in-between intervals is calculated using interpolation algorithms.

CZML Structure - Interpolating cartesian position

The CZML's interval property can also be specified succinctly by providing an epoch. Here, `epoch` refers to the beginning of an interval.

```
{
    // ...
    "someInterpolatableProperty": {
        "epoch": "2012-04-30T12:00Z",
        "cartesian": [
            0.0, 1.0, 2.0, 3.0,
            60.0, 4.0, 5.0, 6.0,
            120.0, 7.0, 8.0, 9.0
        ]
    }
}
```

In the code above we are specifying that the position at 0 second after the time epoch "2012-04-30T12:00" is 1.0, 2.0 and 3.0 . 60 seconds later, the position of the body will be 4.0, 5.0 and 6.0 . This way the position of the satellite can be time-tagged and specified at various intervals. As is observed in the code above, the position of the satellite in between the intervals is not specified. It is not optimal and pragmatic to specify the model's position at every small interval hence, cesium's in-built interpolation algorithm will be used so that the position of the body can be calculated in between the intervals.

```
{
    // ...
    "someInterpolatableProperty": {
        "epoch": "2012-04-30T12:00Z",
        "cartesian": [
            0.0, 1.0, 2.0, 3.0,
            60.0, 4.0, 5.0, 6.0,
            120.0, 7.0, 8.0, 9.0
        ],
        "interpolationAlgorithm": "LAGRANGE",
        "interpolationDegree": 5
    },
}
```

In the example code above, `Lagrange` interpolation is specified in the `interpolationAlgorithm` key.

If the `interpolationDegree` is not explicitly stated, linear interpolation will be used by default.

4.1.4 Displaying a satellite in czml format

In the root folder, a subfolder named `Apps` is found. The `Apps` folder contains a collection of sample models in czml inside the `SampleData` folder. It contains a file named ‘simpl.czml’ which is an authentic czml document. That particular czml document has defined various cesium models in czml format such as `document`, `billboard`, `lines`For the `satellite` model the simple.czml has specified a cesium-native data:image code description in the `image` property.

The image property is defined in the billboard model as follows:

```
"billboard":{
  "eyeOffset":{
    "cartesian":[
      0,0,0
    ]
  },
  "horizontalOrigin":"CENTER",
  "image":"
                   YAAAAf8/9hAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8Y
                   QUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAADJSURBVDhPnZHR
                   DcMgEEMZjVEYpaNklIzSEfLfD4qNnXAJSFWfhO7w2Zc0Tf9
                   QG2rXrEzSUeZLOGm47WoH95x3Hl3jEgilvDgsOQUTqsNl68
                   ezEwn1vae6lceSEEYvvWNT/Rxc4CXQNGadho1NXoJ+9iaqc
                   2xi2xbt23PJCDIB6TQjOC6Bho/sDy3fBQT8PrVhibU7yBFc
                   EPaRxOoeTwbwByCOYf9VGp1BYI1BA+EeHhmfzKbBoJEQwn1
                   yzUZtyspIQUha85MpkNIXB7GizqDEECsAAAAASUVORK5CYII=",
  "pixelOffset":{
    "cartesian2":[
      0,0
    ]
  },
  "scale":1.5,
  "show":true,
  "verticalOrigin":"CENTER"
},
"label":{
  "fillColor":{
    "rgba":[
      0,255,0,255
    ]
  },
  "font":"11pt Lucida Console",
  "horizontalOrigin":"LEFT",
  "outlineColor":{
    "rgba":[
      0,0,0,255
    ]
  },
  "outlineWidth":2,
  "pixelOffset":{
```


In spite of defining the model as `billboard`, the graphical representation of the `billboard` can be manipulated. In the czml sample above it is defined as a base64 format. The graphics formed from the base64 code looks similar to that of a satellite. For the `billboard` model that represents the satellite to be time-dynamic, it needs to be assigned cartesian coordinates within specific time intervals and interpolated with an algorithm.[12]

4.2 Propagating the satellite model in cesium.

The satellite's orbit needs to be estimated with a specific set of well defined

parameters. Such a calculation is referred to as satellite orbit determination (OD). The set of such parameters are received from the GPS of the satellite. The satellite's GPS calculates the physical quantities and the transmits to the ground station when the next connection takes place with the satellite. After receiving the parameters and observing it in the past, the future orbit path is to be predicted using various trajectory physics. Such a computation will be referred to as Orbit propagation (OP). OP will approximate the future orbit path of the satellite.

The reason for the orbit propagating to be an approximation is because the satellite's motion is influenced by various external forces such as atmospheric drag, tids, thrusters and other foreign body's gravitational forces. These various external factors cause the satellites' orbit to be perturbed. As time passes, the approximations' errors start accumulating and eventually the satellites' propagated orbit becomes useless for practical purposes.[13]

Plenty of algorithms are available that propagate the orbit of the satellite and provide discrete position of an orbit within a specific time interval.

This project will use the SGP4 orbit propagation algorithm.SGP4 algorithm is one of five mathematical models (SDP4, SGP, SGP4, SGP8 and SDP8) which are used to calculate orbital state of a satellite. The primary reason for choosing SGP4 is because it can be efficiently modelled with TLE produced by NORAD and NASA. This makes the use of the SGP4 algorithm popular and the open source library of SGP4 implementation is readily available.

Furthermore, the SGP4 algorithm outputs an error of roughly 1 km at epoch. The error eventually grows approximately approximately 1-3 km per day. Hence, the idea is to refresh the TLE every 24 hours.[14]

Considering that a Low Earth Orbit satellite travels at a linear velocity of

approximately 7.5km/day the error output of SGP4 models is considered for this project's purposes. Originally the SGP4 algorithm was published in a sample code in FORTRAN IV in 1988. However, the SGP4 algorithm has been rewritten and published in other languages so far. Since, the cesium application is run in a javascript runtime environment a javascript implementation of the mode is to be used by our cesium application.

4.2.1 Implementing SGP4 algorithm

An open source project named satellite.js is to be imported which has a modelled SGP4 algorithm in its library. This project is specifically designed to make a satellite propagate using TLE which are available on the web. It is a javascript library, hence it consists of javascript functions which provides the necessary SGP4 calculation. In addition to those functions it also provides transformation functions for coordinates of the satellite. The availability of such functions has an added value for clients and users because the coordinates of the satellite can be displayed in any format that the client requires.

The library encapsulates the SGP4 algorithm which can be accessed by importing the library in our main app.js file. Before importing the library, it is to be installed using a package manager. For this project npm will be used. The user is to navigate to the root directory and install the project using npm with the following command:

```
npm install satellite.js
```

This command installs the satellite library in the dist folder. The name of the file is `satellite.min.js`. It is to be loaded using a script file from a html file. It can be included in html file with the command:

```
<script src="path/to/dist/satellite.min.js"></script>
```

As the satellite.js library is loaded all its functions can now be imported. It has a

`satellite` object available in global scope. Hence, it can be readily as follows:

```
var positionAndVelocity = satellite.sgp4(satrec, time);
```

Sample code snippets for propagating the satellite is shown below:
Firstly the tle is to be loaded in the javascript file.

```
// Sample TLE
var tleLine1 = '1 25544U 98067A   19156.50900463  .00003075  00000-0  59442-4 0  9992',
    tleLine2 = '2 25544  51.6433  59.2583 0008217  16.4489 347.6017 15.51174618173442';
```

The loaded tles are to be passed as parameters when instantiating a satrec object.

```
// Initialize a satellite record
var satrec = satellite.twoline2satrec(tleLine1, tleLine2);
```

After the satellite object is instantiated it can be propagated for the calculation of its position and velocity. The function for propagating the satellite is named `sgp4` and it takes two arguments: the `satrec` object and time.

```
//  Propagate satellite using time since epoch (in minutes).
var positionAndVelocity = satellite.sgp4(satrec, timeSinceTleEpochMinutes);

//  Or you can use a JavaScript Date
var positionAndVelocity = satellite.propagate(satrec, new Date());
```

The position and velocity of the satellite for a given time is then assigned to the variable `positionAndVelocity`. That variable is returned a key-value pair of ECI coordinates. ECI is a coordinate system where earth is located in the center and is considered to be moving in the inertial plane. The ECI coordinates can be extracted from the variable as follows.

```
// The position_velocity result is a key-value pair of ECI coordinates.
// These are the base results from which all other coordinates are derived.
var positionEci = positionAndVelocity.position,
    velocityEci = positionAndVelocity.velocity;
```

Since GMST is considered a standard time reference, there is also the option to transform the coordinates to gmst.

```
// You will need GMST for some of the coordinate transforms.
// http://en.wikipedia.org/wiki/Sidereal_time#Definition
var gmst = satellite.gstime(new Date());
```

Additionally, the library allows you to transform the coordinates from ECI to ECF.

Physical quantities associated with aerodynamics such as `dopplerFactor` and `lookAngles`.

```
// You can get ECF, Geodetic, Look Angles, and Doppler Factor.
var positionEcf   = satellite.eciToEcf(positionEci, gmst),
    observerEcf   = satellite.geodeticToEcf(observerGd),
    positionGd    = satellite.eciToGeodetic(positionEci, gmst),
    lookAngles    = satellite.ecfToLookAngles(observerGd, positionEcf),
    dopplerFactor = satellite.dopplerFactor(observerCoordsEcf, positionEcf, velocityEcf);
```

The satellite has now been propagated. X, Y and Z coordinates need to be extracted from the `positionEci` variable.

```
// The coordinates are all stored in key-value pairs.
// ECI and ECF are accessed by `x`, `y`, `z` properties.
var satelliteX = positionEci.x,
    satelliteY = positionEci.y,
    satelliteZ = positionEci.z;
```

The `lookAngles` variable itself contains various properties which are different types of look angles.

```
// Look Angles may be accessed by `azimuth`, `elevation`, `range_sat` properties.
var azimuth   = lookAngles.azimuth,
    elevation = lookAngles.elevation,
    rangeSat  = lookAngles.rangeSat;
```

The user is also able to access geodetic coordinates such as longitude, latitude and height with the following code.

```
// Geodetic coords are accessed via `longitude`, `latitude`, `height`.
var longitude = positionGd.longitude,
    latitude  = positionGd.latitude,
    height    = positionGd.height;
```

In case of situations where the user requires the geodetic coordinates in degrees instead of radians, there is an in-built function for conversions.

```
//  Convert the RADIANS to DEGREES for pretty printing (appends "N", "S", "E", "W", etc).
var longitudeStr = satellite.degreesLong(longitude),
    latitudeStr  = satellite.degreesLat(latitude);
```

This library enables the user to generate satellite coordinates for a fixed interval of time. The idea now is to feed the TLE to the `satrec` function as parameter, generate the coordinates of the satellite, include the coordinates in czml format and then make the satellite orbit around the globe presented in the cesium application.

4.2.2 Representing a time-dynamic satellite using czml

In order to represent the satellite in the cesium app with respect to time, its coordinates would need to be calculated using satellite.js as time passes from epoch. This is achieved simply by invoking the `satellite.sgp4` function over a loop with index step 1 and the index is passed as a second argument. Such a loop will return the position and velocity of the satellite at every minute of the interval. A sample code snippet is produced below.

```
var satrec = satellite.twoline2satrec(tleLine1, tleLine2);
for (var i = 0; i < 2882; i++) {
  var positionAndVelocity = satellite.sgp4(satrec, i);

  var positionEci = positionAndVelocity.position;
  var velocityEci = positionAndVelocity.velocity;
```

```
var satrec = satellite.twoline2satrec(tleLine1, tleLine2);
for (var i = 0; i < 2882; i++) {
  var positionAndVelocity = satellite.sgp4(satrec, i);

  var positionEci = positionAndVelocity.position;
  var velocityEci = positionAndVelocity.velocity;

  var tempDate = new Date(
    Date.UTC(
      parseInt(tleEpoch.substr(0, 4)),
      parseInt(tleEpoch.substr(5, 2)) - 1,
      parseInt(tleEpoch.substr(8, 2)),
      parseInt(tleEpoch.substr(11, 2)),
      parseInt(tleEpoch.substr(14, 2)),
      parseInt(tleEpoch.substr(17, 2)),
      i * 60000
    )
  );

  var gmst = satellite.gstimeFromDate(tempDate);

  // You can get ECF, Geodetic, Look Angles, and Doppler Factor.
  var positionEcf = satellite.eciToEcf(positionEci, gmst),
    //observerEcf   = satellite.geodeticToEcf(observerGd),
    positionGd = satellite.eciToGeodetic(positionEci, gmst);

  var satelliteX = positionEcf.x,
    satelliteY = positionEcf.y,
    satelliteZ = positionEcf.z;

  coords.push(
    j * 60,satelliteX * 1000,satelliteY * 1000,satelliteZ * 1000
  );
  j++;
}
```

In the code above satellite coordinates are calculated for approximately 48 hours. `tempDate` variable extracts the date information stored in the TLE and adds to it a UTC timestamp. This timestamp removes the inconsistency that is

created when the clients' local machine is located in different timezones. `coords` variable is an array where the coordinates of the satellite are stored. Variables `gmst` stores the Greenwich sidereal time format which is the format preferred to receive the position of the orbiting body.

As the name suggests `satellite X/Y/Z` are the X coordinate, Y coordinate and Z coordinate of the satellite which is extracted from the ECF coordinate system outputted by the function. `j` is a variable counter which counts the increment of the time. It is a multiplier of `60` denoting that the interval of each coordinate is a minute. All of the generated coordinates are pushed and stored into an arraylist with variable named `coords` along with the time passed since epoch which is represented by the variable `j`.

If the coords array is loaded by the czml billboard it would only display the position of the satellite in a discrete manner as the position is not of the satellite is not stated within the intervals of one minute. As mentioned previously, position of the satellite is an interpolatable property. An additional property that mentions the CZML document to interpolate the satellites' property would automatically fill in the coordinate points.

The `coords` variable would need to be specified in the `cartesian` section of the `position` property. As stated in the introduction of CZML format, since CZML is a valid JSON document the assignment of `coords` array to CZML's key does not throw an error.

Sample code below displays an example.

```
"billboard":{
  "eyeOffset":{
    "cartesian":[
      0,0,0
    ]
  },
  "horizontalOrigin":"CENTER",
  "image":""
          "Af8/9hAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADs"
          "MAAA7DAcdvqGQAAADJSURBVDhPnZHRDcMgEEMZjVEYpaNklIzSEfLfD4qNnX"
          "AJSFWfhO7w2Zc0Tf9QG2rXrEzSUeZLOGm47WoH95x3Hl3jEgilvDgsOQUTqs"
          "Nl68ezEwn1vae6lceSEEYvvWNT/Rxc4CXQNGadho1NXoJ+9iaqc2xi2xbt23"
          "PJCDIB6TQjOC6Bho/sDy3fBQT8PrVhibU7yBFcEPaRxOoeTwbwByCOYf9VGp"
          "1BYI1BA+EeHhmfzKbBoJEQwn1yzUZtyspIQUha85MpkNIXB7GizqDEECsAAA"
          "AASUVORK5CYII=",
  "pixelOffset":{
    "cartesian2":[
      0,0
    ]
  },
  "scale":1.5,
  "show":true,
  "verticalOrigin":"CENTER"
}
"position":{
  "interpolationAlgorithm":"LAGRANGE",
  "interpolationDegree":5,
  "epoch":epoch,
  "cartesian":coords
},
```

`Lagrange` algorithm is chosen for interpolation and the interpolation degree specified is 5. The `epoch` key refers to the beginning time of the event and the date time is assumed to consist of UTC timestamp to remove inconsistencies in clients located in different timezones.

`cartesian` key of the `position` is assigned with value as `coords` array. As per the calculation. `coords` array contains the position of the satellite with only the intervals of minutes. `interpolationAlgorithm` specification in the CZML document calculates the position of the satellite in between those satellites and displays a graphical movement of the satellite.

The CZML documents presented in the sample code are static and are hard

coded by the user. This method of displaying the satellites to the user is inefficient and impractical. It completes the task of actually propagating the satellite and displaying it in the cesium app. However, to make the application more scalable and dynamic, the information regarding the TLEs and position of the satellite are to be assigned by the code, inside the application itself rather than manually entering the TLEs.

For this purpose a mini library is to be introduced which automates the task of loading TLEs, propagating the satellite with the TLEs that has been inputted, computes the coordinates of the satellite with respect to time along with necessary coordinate transformations and finally bundles the information in a valid CZML document which can then be loaded by the cesium app. It is that particular CZML document which, when loaded displays the satellite along with its orbit and also shows the actual graphical revolution of the satellite around the earth's globe.

The library's primary objective is to output a CZML document, considering its functional purpose, the library is a sort of a `czml-writer`. Hence, a file named `czml-write.js` is to be created. Inside the Apps folder in the root directory a sub folder named `js` is created which shall contain all the functional javascript code including `app.js`

`czml-writer.js` is not a purely dynamic library. It contains static components to it as well. Such as the list of months with their index and helper functions used for padding the digits of time displayed. It could also potentially include any additional information that the client desires to be displayed on the cesium application. Most common are the position of the ground station on the globe, a sunlight terminator that contrasts the sunlight from shadow and pass lines which are simply a straight line drawn from satellite to the ground station during the interval when there is a connectivity with the satellite. The time interval when

the satellite is within the radius of connectivity from ground station is conveniently referred to as a `pass` because the satellite passes over the ground station.

The sample code below shows a few static components of `czml-writer.js`

```
var month = {
   "Jan":"01",
   "Feb":"02",
   "Mar":"03",
   "Apr":"04",
   "May":"05",
   "Jun":"06",
   "Jul":"07",
   "Aug":"08",
   "Sep":"09",
   "Oct":"10",
   "Nov":"11",
   "Dec":"12"
};
```

A `month` constant with indices is required by the library for efficient two way conversion from numbers to months. This benefits the function of assigning a date to the cesium clock and also aids the users to easily understand the context of time in terms of month.

```
function pad(n){
    return n<10 ? '0'+n : n;
}

function timeConverter(difference){
    var hours = (difference)/3600|0;
    var minutes = (difference/60)%60|0;
    var seconds = (difference%60)|0;

    pad(hours)
    pad(minutes)
    pad(seconds)

    return(hours+":"+minutes+':'+seconds);
}
```

Sample code above are helper functions which parse and return information in appropriate formats. `pad` function as the name suggests pads zeroes in digits that are less than 10 in magnitude. `timeConverter` function is one if the function which in turn uses the `pad` function to add 0s in its output. `timeConverter` function returns time in the format HH:MM:SS. The `difference` argument denotes an integer which is `number of seconds` and the function converts it into a different time format.

```
this.czmlObj =
[{
    "id":"document",
    "name":"SimpleExample",
    "description":"A simple example",
    "version":"1.0",
    "clock":{
      "interval":"",
      "currentTime":""
    }
},
{
    "id":"Facility/KSAT",
    "name":"KSAT",
    "billboard":{
      "eyeOffset":{
        "cartesian":[
          0,0,0
        ]
      },
      "horizontalOrigin":"CENTER",
      "image":""
              "hEUgAAABAAAAAQCAYAAAAf8/9hAAAAAXNSR0IArs"
              "4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADs"
              "MAAA7DAcdvqGQAAACvSURBVDhPrZDRDcMgDAU9Gq"
              "N0lIzijw6SUbJJygUeNQgSqepJTyHG91LVVpwDdf"
              "xM3T9TSl1EXZvDwii471fivK73cBFFQNTT/d2KoG"
              "pfGOpSIkhUpgUMxq9DFEsWv4IXhlyCnhBFnZcFEE"
              "uYqbiUlNwWgMTdrZ3JbQFoEVG53rd8ztG9aPJMnB"
              "UQf/VFraBJeWnLS0RfjbKyLJA8FkT5seDYS1Qwyv"
              "8t0B/5C2ZmH2/eTGNNBgMmAAAAAElFTkSuQmCC",
      "pixelOffset":{
        "cartesian2":[
          0,0
        ]
      },
      "scale":1.5,
      "show":true,
      "verticalOrigin":"CENTER"
    },
```

The sample object presented in the code above with the name `this.czmlObj` is the main CZML document which is to be parsed and formatted. This is the object that will be returned in the end. The code above displays two CZML models, `document` and `billboard`. The `document` model contains a `clock` key which denotes the clock graphics present in the bottom left of the cesium application. The clock model is included in the CZML document so that the clock can be assigned to a time interval when the satellite is actually propagated.

`The billboard` model is introduced to display a ground station on the globe. `image` property of the model specified CZML native code to display graphics similar to that of a ground station. The coordinates for the billboard model is specified as follows:

```
"position":{
  "cartographicDegrees":[
    15.389123,78.230127,459
  ]
}
```

As can be observed in the sample code above, the czml object may contain an arbitrary number of czml objects including clocks, billboard, satellit models, terminator lines or even any other aerodynamic cesium models. For efficient retrieval of such models present in the CZML document a function is to be introduced that retrieves the object by its identifier.

```
this.findObjectById =
function(id){
    for(var element of this.czmlObj){
        if(id===element["id"]){
            return element;
        }

    }
};
```

Additionally a function is to be introduced that adds a CZML model into the document.

```
this.addObject =
function(obj){
    this.czmlObj.push(obj["obj"]);
};
```

This `addObject` function is used for example when a satellite model is created and assigned to a variable and it is dynamically added to the CZML document by calling this function with the said variable as a parameter. As can be observed in the code, the function will simply `push` the variable in `this.czmlObj` variable which in itself is an array of JSON objects.

There are cases when, instead of adding new CZML objects, already created objects are to be manipulated and assigned new properties and features. For the purpose of displaying the satellites within an appropriate timeline, the clock of the cesium application would need to be assigned the epoch of the time. The clock can be accessed with the helper function `this.findObjectById`.

```
244     this.assignDate =
245     function(){
246
247         var czmlDocument = this.findObjectById("document");
248
249         var intervalStartDate = new Date();
250         intervalStartDate.setHours(intervalStartDate.getHours()-6);
251         timeLineInterval = intervalStartDate.getUTCFullYear()+"-"+
252                            pad(intervalStartDate.getUTCMonth()+1)+
253                            "-"+pad(intervalStartDate.getUTCDate())+
254                            "T"+pad(intervalStartDate.getUTCHours())+
255                            ":"+pad(intervalStartDate.getUTCMinutes())+
256                            ":"+pad(intervalStartDate.getUTCSeconds())+
257                            "Z/"+intervalStartDate.getUTCFullYear()+"-"
258                            +pad(intervalStartDate.getUTCMonth()+1)+"-"
259                            +pad(intervalStartDate.getUTCDate()+1)+"T"
260                            +pad(intervalStartDate.getUTCHours())+":"
261                            +pad(intervalStartDate.getUTCMinutes())+":"
262                            +pad(intervalStartDate.getUTCSeconds())+"Z";
263         czmlDocument["clock"]["interval"] = timeLineInterval;
```

```
264
265         now = new Date();
266         now = now.getUTCFullYear()+"-"+pad(now.getUTCMonth()+1)+
267                 "-"+pad(now.getUTCDate())+"T"+pad(now.getUTCHours())+
268                 ":"+pad(now.getUTCMinutes())+
269                 ":"+pad(now.getUTCSeconds())+"Z";
270         czmlDocument["clock"]["currentTime"] = now;
271
272     };
```

`this.assignDate` is a function which is to be invoked during the initialization of the cesium application. The reason to execute this particular function is so that the time interval that is displayed to the users using the cesium application is correctly adjusted within a timeframe. As can be observed in line 250 of the code sample above, the clock time interval begins from 6 hours before the current time when the application is instantiated. For example, if the user opens the cesium application on date 05.06.2020 and time 11:23:00, the clock's starting time will be dated 05.06.2020 and time 05:23:00.

Finally, the code displayed in section 4.2.2 which does the actual propagation, is to be included in the czml-writer.js file. After the `propagate` function is invoked with the necessary arguments, the position of the satellites is stored in an array named `coords` as is shown in the code. Hence, the next step would be to include the `coords` array to the czml document which can be achieved as follows:

```
}
this.obj["position"]["cartesian"] = coords;
```

Additionally, it is to be noted that the cartesian coordinates displayed above has been interpolated by the lagrange's algorithm with interpolation degree 5. The czml-writer.js is now completed and a mini library for creating czml documents is introduced to the project.

4.3 Displaying additional information on cesium application

There could potentially come situations when the clients will want to actually view additional information in the cesium application. Such additional information may be satellites' position in the geodetic coordinate system, the timing of the passes and eclipses or satellites' clocks. Furthermore, the clients could also want to view the markers of the globe such as prime meridian and equator.

The cesium application creates a layer of canvas on the html page. For the clients to be able to see any additional information on the cesium application, any additional that is not introduced via cesium's library application would have to overlap on the canvas. This project displays the additional information by creating an HTML `div` panel and laying it on top of the cesium application. Firstly a`div` tag is introduced in the index.html page with the id `overlay`. `Overlay` style is to be defined in a css file named `styles.css` .

```
<div id="overlay">

</div>
```

`overlay` css id is to be defined as follows:

```
#overlay{
  font-size: 1.11em;
  top:26px;
  left:26px;
  position:absolute;
  z-index:1;
}
```

The `z-index` property with 1 makes all the contents defined inside the div tag appear on the top of the cesium application. In the sample code above the position of the `div` html tag is `absolute` which means the div will always appear in the same location of the page regardless of the size of the window. Therefore, the size of the div does not change dynamically.

4.3.1 Displaying geodetic coordinates of the satellites

Geodetic coordinate system refers to the coordinate system in which position is specified by geodetic latitude, geodetic longitude and (in the three-dimensional case) ellipsoidal height. The coordinates are stored in the `coords` array of CZML documents in the cartesian coordinate system. However, it can be conveniently converted to a geodetic coordinate system with the in built helper functions that are in built in Cesium application. Such functions are introduced in section 4.2.1 .

An html div panel is required to display the coordinate numbers to the clients. The div panel is to be created inside the `div` html tag so that it is overlayed on top of the cesium canvas.

```
<div id="overlay">
  <div class="panel panel-info">
  </div>
</div>
```

A panel heading is to be included and an id is assigned so that it can be accessible via javascript.

```
<div class="panel-heading">
  <span id="position">Position</span>
</div>
```

The sample code above displays the html panels in the cesium application but it will be blank since there are no contents filled inside it. The contents that are to be displayed are the latitude, longitude and height of the satellite at current time.

This can be achieved by periodically retrieving the position of the satellite model that will be displayed in the cesium application. Then an in-built helper function is called which returns the cartesian coordinates of the model at a specific point in time by taking in the timestamp as argument. Additional helper functions are available which transform the cartesian coordinates to cartographic and then

the numbers obtained are to be dynamically updated on the `div` panels that were previously created.

```
function updatePosition(){
  var satEntity = czmlEntities.getById(satelliteId);
  var satellitePosition = satEntity.position
                          .getValue(viewerClock.currentTime);
  var positionInRadian = Cesium.Cartographic
                          .fromCartesian(satellitePosition);


  latitude.text((positionInRadian.latitude*57.2957795131)
                .toFixed(9));
  longitude.text((positionInRadian.longitude*57.2957795131)
                .toFixed(9));
  altitude.text(((positionInRadian.height)/1000).toFixed(3)+
                ' kiloMetres');
}
```

The sample code above displays an 'updatePosition` function which extracts the current position of the satellite at any given time. The `czmlEntities` variable is a collection of all the CZML objects present in the czml document. The cartesian coordinates returned by the helper function are present in the radian units. A simple conversion is done from radian to degrees by multiplying the number with 57.29577 which is equivalent to 180/π .

`latitude`, `longitude` and `altitude` variables are simple html objects which have been accessed via javascript. Initially they are to be defined in the HTML document.

```
<div class="panel-body">
  Latitude = <span id="latitude"></span><br>
  Longitude = <span id="longitude"></span><br>
  Altitude = <span id="altitude"></span><br>
</div>
```

This block of code is to be wrapped inside the div with `overlay`. Since the span tags are each given an `id`, these can be accessed in `app.js` file as follows.

```
var latitude = $('#overlay #latitude');
var longitude = $('#overlay #longitude');
var altitude = $('#overlay #altitude');
```

The final step would be to actually invoke the function periodically in an interval. This can be accomplished with javascript's setInterval function.

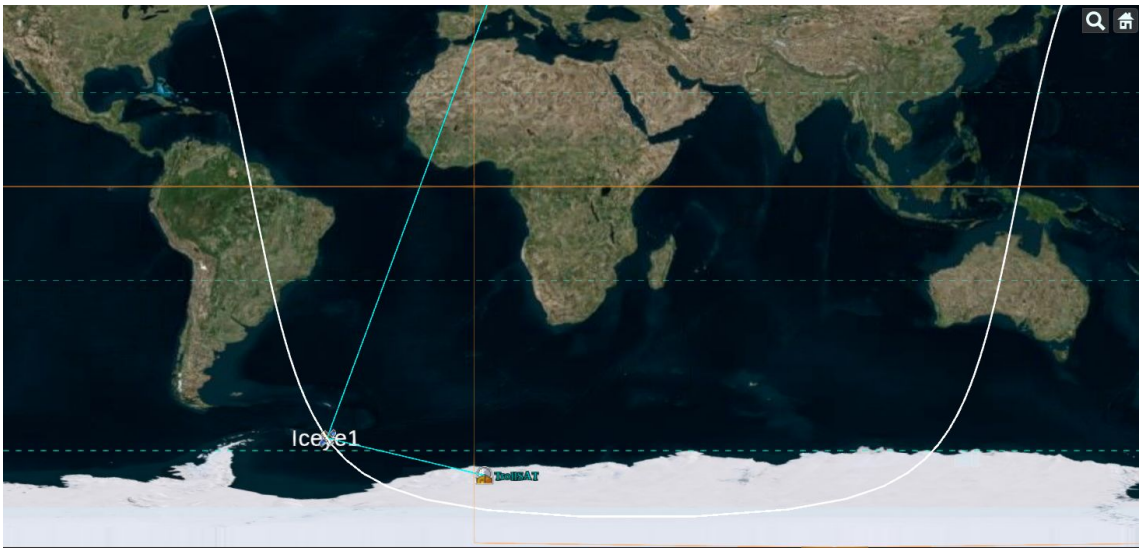
```
var tid = setInterval(updatePosition,1000);
```

Conclusion

At this stage of the project, It is expected that the satellite is propagated in SGP4 algorithm using satellite.js library and the predicted orbit is displayed in the cesium application. Additionally, there should appear a panel on the upper left corner of the screen which updates the coordinates of the satellite on each interval of every second.

The application is primarily written in javascript so it can be conveniently be built and run in a javascript runtime environment and loaded via any web browser. While deploying it in production, it is hosted in nginx server and the port number where the default cesium application is hosted can be forwarded to the necessary endpoint as per clients' request. This way, applications will be accessible when the users type in the specified url in their browser given that the application is hosted in the network that the users are logged into.

There are other various graphics configurations that the clients can utilize to edit the graphics feature of the application as per their needs. For instance, the map can be displayed on 2d plane



The users can also speed up and slow down the clock widget of the cesium application

Since all of the tools used for this project are open source libraries, the clients can modify underlying libraries to suit their needs.

This project has brought together various open source dependencies, and the interaction of their functionalities has been constructed to display the satellites' orbit trajectory in a web browser. This project renders the use of desktop applications to visualize the satellites' orbit as obsolete.

References

1. Small Satellites-A tool for earth observation. Konecny G. Invited Paper.[online].

   URL: https://www.researchgate.net/publication/229028414_Small_satellites-A_tool_for_Earth_observation
   Accessed: 10 October 2017.

2. Inter-Agency Space Debris Coordination Committee. IADC Space Debris Mitigation Guidelines;15 October 2002.[online].

   URL: https://www.unoosa.org/documents/pdf/spacelaw/sd/IADC-2002-01-IADC-Space_Debris-Guidelines-Revision1.pdf
   Accessed 10 October 2017.

3. Britannica Encyclopedia online.

   URL: https://www.britannica.com/science/orbit-astronomy
   Accessed : 12 October 2017

4. Kuhn TS. The Copernical Revolution. Harvard University Press; 1995;

5. Green, Robin M. Spherical Astronomy. Cambridge University Press ; 1985.

6. Spaceflight.nasa.gov. NASA, Definition of Two-line Element Set Coordinate System [online].

URL:https://spaceflight.nasa.gov/realdata/sightings/SSapplications/Post/JavaSSOP/SSOP_Help/tle_def.html357fa2c

7. Models for propagation of NORAD Element Sets. Spacetrack report no.3, Felix R. Hoots, Ronald L. Roehrich. December 1980 [online].

URL: http://celestrak.com/NORAD/documentation/spacetrk.pdf
Accessed: 19th December 2018

8. NGINX docs, NGINX project (2019). [online] available at https://docs.nginx.com/nginx/ [Accessed 20 December 2019]

9. Reactjs.org, Facebook (2020). [Online] Available at

https://reactjs.org/docs/getting-started.html [Accessed 2 January 2020]

10. Node.js docs, Node.js (2019). [Online] Available at https://nodejs.org/en/docs/

[Accessed 4 January 2020]

11. Cesium.js Docs, Cesium.js Open Source 3D mapping. [Online] Available at https://cesium.com/docs/cesiumjs-ref-doc/ [Accessed 29 May 2019]

12. CZML Guide, Cesium Language (CZML) Guide. [Online] Available at https://github.com/AnalyticalGraphicsInc/czml-writer/wiki/CZML-Guide [Accessed 1st February 2020]

13. Orbit Propagation and Determination Of Low Earth Orbit Satellites. [Online] Available at https://www.hindawi.com/journals/ijap/2014/903026/ [Accessed 2nd March 2020]

14.Revisiting Spacetrack Report. Vallado, David A.; Paul Crawford; Richard Hujsak; T. S. Kelso (August 2006). [Online] Available at http://celestrak.com/publications/AIAA/2006-6753/AIAA-2006-6753-Rev2.pdf [Available at 10 October 2019]

15. Satellits.js, A javascript library for satellite propagation using TLEs available on the web. [Online] Available at https://github.com/shashwatak/satellite-js [Available at 2 January 2020]