Visualizing Orbital Debris Threats to Space Borne Objects

by

Siwei Zhao

Submitted in partial fulfillment of the Requirements for the degree of Bachelor of Computer Science, Honours

At

Dalhousie University
Halifax, Nova Scotia
June 2016

Table of Contents

Abstract

Space debris with sizes larger than one centimeter pose significant threats to spacecraft. The surveillance, tracking and early warning of space debris has become increasingly urgent owing to the explosive growth of the debris in orbit. Visualizing orbital debris threats to space borne objects is an important branch of geospatial visualization which is usually implemented in space situational awareness systems. A Space Situational Awareness (SSA) system is a geographic information system that are used by government agencies in many countries. It has the ability to view, understand and predict physical locations of natural and manmade objects in orbit around the Earth.

My system presents a new approach for SSA systems. It utilizes two-dimensional diagrams and a three-dimensional virtual earth in concert to represent the range of orbital debris threats with a strong interactive component. Users can realize the types and proportions of orbital debris threats by two-dimensional donut charts. Users can also view potential locations of threats by observing a geographical discrete global grid system on the virtual earth. By spinning the globe, users can observe the geographical location of threats directly. When compared with traditional methods of relying the combination of the points and curves, this system presents represents the regions of threats at in such a way as to facilitate high level monitoring over time.

My system builds upon D3.js and Cesium technology to combine two-dimensional visualization and three-dimensional visualizations to represent orbital debris threats. For two-dimensional visualizations I utilize the D3.js toolkit for drawing scatter plots and donut charts. D3.js is one of the most popular visualization frameworks which allows any data to be bound to the web DOM model. The Document Object Model (DOM) is a platform and language neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of web documents [23]. To put it simply, the DOM is a tree structure to represent data and elements on webpages. D3.js also has the advantage of high flexibility, cross-platform portability and efficient operating the large datasets. I also develop software within the Cesium geographic visualization framework and apply an Icosahedral Snyder Equal Area aperture 3 Hexagon partition method to implement a geographical discrete global grid system on a virtual earth. Cesium is an open-source JavaScript library for implementing dynamic information visualization on high quality 3D globes and maps.

Acknowledgment

Four-year college life comes to an end finally. I want to first acknowledge my teachers Ren Tao and Stephen Brooks. The successful completion of this thesis is inseparable from the teacher's careful guidance. Whenever I encounter difficulties that you will make every effort to help me. My teachers gave me a lot of help from selecting subject to writing review and many valuable suggestions for revision. Thanks for these two teachers care and guidance in the process of choosing topic and research. I give my sincere appreciation and great respect to them.

I would like to sincerely thank all my classmates and friends. Whenever I encounter difficulties that you will make every effort to help me. I want to thank my roommates for their companionship, support and encouragement, both in learning as well as life.

I would like this thesis dedicated to my favorite of all people, including my dear teacher, classmates and relatives. I also want to thank my parents for their care and support.

Chapter 1 Introduction

1.1 Background

There are more than 4,000 records of launches in the past fifty years since the liftoff of the first artificial satellite in 1957. It is accompanied by nearly 4,500 tons of orbital debris floating in orbit around the earth. Orbital debris, sometimes referred to as space trash, are useless human-made objects moving round the earth in orbit including various defunct satellites and the remains of rockets. Orbital debris also includes fragments formed by explosions and crashes of space borne objects. As of 2009, there are more than 19,000 orbital debris over 5 centimeters on record [24]. Orbital debris pose vital threats to spacecraft and satellites since even a minor crash can cause significant harm. Orbital debris with the size as small as a pill can devastate a major satellite. As such, the International Space Station adjusted its orbit and height many times in 2015, in the most active time, it adjusted orbit two times in a single week. There question therefore becomes how to avoid space debris and is an urgent technical problem for researchers.

Space Situational Awareness Visualization is a vital component of obstacle avoidance systems for satellites [1]. The main content involves converting data received into comprehensible visualization diagrams and representing the results of data in clear ways to help users understand the events and trends in the data.

Data visualization itself is primarily intended to aid as a graphical means for communicating clearly and effectively conveying information. Visualization of data helps to improve people's understanding of the data and it originated in 18th century, with bar charts and line charts first used in the work “The Commercial and Political Atlas” [25]. And in the early 19th century, William Player later published “Statistical Breviary”[26] which also introduced the pie chart. These three visual graphics are by far the most common even to this day. By the mid-19th century, data visualization was mainly used for military purposes, often representing the causes of death in the army, and the distribution of the army more generally.

Later, in the nineteen-eighties, R.M. Pickett and G.G. Grinstein presented new

iconographic displays for visualizing multidimensional data [27]. In 1990, the Society of the Man-Machine Interface, information visualization was published as a prototype technology. In 1995, IEEE Information Visualization was officially founded, which formally established it as an independent discipline. As the world now enters in the era of big data, data visualization massive amounts of data becomes an important issue.

1.2 Related Research

In this thesis I use D3.js [28] as a two-dimensional toolkit to develop aspects of my system. The full name is Data-Driven Documents which is a document driven system and a JavaScript function library. D3 can simplify complex steps for generating charts into streamlined functions by working directly with web data. Many domestic and foreign developers have created complex D3.js based applications. Prominent examples include The Wealth & Health of Nations that shows dynamic fluctuations in recent years, between 180 and 209 different countries about incomes and people's lives [2]. This visualization application implements dynamic effects by using a linear interpolation method of handling missing data.

For map visualization, some foreign companies choose Cesium as geographic visualization engine. NASA iSat is one such interactive tool for displaying information and research about satellites [3]. Cesium provides the computing and rendering of position information. Another example is the Cesium 4D Choropleth map [4]. In this example, the number of Taiwan's population density and population were visualized on a virtual globe.

Mikan Stamenkovich provides a 3D situational awareness display that helps reduce the risk of undersea groundings and collisions[5]. He uses Google Earth (GE) software as a foundation information system to display and operate with existing 2D hydrographic Vector Product Format (VPF).

Nathan Dennis MacCall present a method for using irregular polyhedrals to gain more control over the placement of the projective centers while maintaining the reduced distortion quality found in polyhedral projections [6]. His method uses irregular polyhedrals based on the method of projected Voronoi partitions of a sphere.

However, we focus on the combined application display of 2D data visualization and 3D cartographic visualization. Although data visualizations directly reflect the relationship

between the data, they do not in themselves reflect the orbital positions of geographic information. Map visualizations can only display the data of the current moment, but to not easily display data over time [7].

To solve this problem, this thesis provides a method creating a visual combination of the two kind of visualizations. My system implements this method by making D3 and Cesium work together and share data. D3 is responsible for recording differences compared with the current and historical data to help users predict space borne threats. In concert, Cesium is responsible for displaying the threats of space debris at the present time, allowing users to visually see the locations of the threats as they occurred. In this way, the space debris threat visualization application can provide two visual perspectives [8].

1.3 Content of Research

I explore the visualization of orbital debris threats to space borne objects in my thesis and create a Space Situational Awareness system based on creating a Geographical Discrete Global Grid System on virtual earth. In support of and preparation for this work I studied the following concepts:

(1) Visualization of human visual perception. The method arouses the user's attention in different situations and better reflect changes in the data visually.

(2) The visualization efficiency of interactive visualization design. Different interaction visualizations choices are particularly important in different situations. For example, a pie chart can better reflect the relationship between part data and overall data, but when the difference is too large or too small, visual efficiency will be very low. Similarly, line charts, bar charts, partition maps and trees each can introduce problems in certain situations. How to avoid such weaknesses and to select the most appropriate effective visualization one issue I studied carefully.

(3) The symbolic representation of data items. For example, commercial visualization software such as Many Eyes [29] (online visualization tool designed for IBM) has a variety of clear and significant icons.

(4) Geospatial layouts and navigation. The partition algorithm used in my thesis splits the

virtual surface of the earth and is generally related to research on discrete surface geometry and iterative subdivision methods in triangular, hexagonal, spherical segment grids.

(5) The mission-critical visualization systems. Mission-critical applications related to risk avoidance, emergency, people need to fully take into account the limitations data in an emergency. A central issue is how in the shortest possible time can one give the most appropriate information, and how to avoid human error.

(6) The cooperation between two-dimensional situational awareness system and three-dimensional situational awareness system. In my work I utilize D3.js creating visualization dashboard and Cesium generating virtual earth.

1.4 Thesis Organization

The structure of the thesis will be introduced as follows：

Chapter 1 is introduces the research background, related research work, general research content and organization.

Chapter 2 introduces the key technology in the thesis, 2D visualization framework and 3D geographic visualization toolkits (D3.js and Cesium respectively) and the data storage format in my system.

Chapter 3 introduces data bindings and visual processes in D3.js. This chapter also presents dashboard design principles in warning systems.

Chapter 4 analyzes three kinds of spatial partitioning methods and details how to generate a Geographical Discrete Global Grid System in Cesium.

Chapter 5 introduces the cooperation between 2D and 3D situational awareness system as well as the final result.

Chapter 6 discusses system testing.

Chapter 7 is the conclusion and future work.

Chapter 2 Introduction of Key Technologies

2.1 Introducing D3.js

D3 (Data-Driven Documents) is a JavaScript library for manipulating web documents based on data and has been adapted for producing dynamic and interactive 2D data visualization in web browsers. D3 provides a variety of methods which simplify the generation of web based visualizations. D3 is essentially a JavaScript library, which means one can also use other JavaScript based libraries in tandem which can greatly reduce workload. D3 can simplify complex visualization steps into several particular functions such as transformations and animations of basic graphics. D3 is not itself a graphics rendering library, it depends on standard Web technologies such as HTML, SVG, CSS to draw visual elements. Users require a sufficient knowledge base of web technology before using D3.

One of the major points in favor of D3.js is that the developer can customize visualizations in an unconstrained fashion. This is not always possible in some other visualization libraries which only provide frequently used charts, such as line, spot, column, etc. D3 takes advantage of web technology (HTML, CSS, SVG) to manipulate every element on the page and bind data to those elements. For example, an SVG element can be created in D3, bound to data which might affect the size of the element and then manipulated with a force-directed graph which changes the locations and states interactively. Even complicated visualizations that incorporate world maps can be created by reading JSON data (discussed in section 2.3). So the first notable advantage of D3 is flexibility.

Second, D3 only requires a web browser to achieve basic and advanced visual effects without external dependencies and environments. This makes D3 project lighter and more readily shared and deployed.

Third, D3 has extensive interaction which inherits form existing implemented functions of JavaScript within browsers. This ensures D3 provides a quick response of click and drag operations to achieve real-time response between the user and browser.

2.2 Introduction of Cesium

Cesium is a JavaScript library for creating 3D globes and 2D maps in web browsers without requiring an extension or plugin[30]. It was established in 2011, and the goal of the developer is to advance the state of art in 3D web based Digital Earth interaction. As an open-source project, Cesium has succeeded in this aim. It has become widely known for 3D geospatial visualization and is the platform of choice across many fields, such as aerospace, defense, smart cities, real estate, sports, tourism, and traditional GIS. Cesium has been hosted at Github [31], and anyone can learn or develop their own application with Cesium. Compared with other 3D visualization engines, Cesium has the following advantages:

First, Cesium has powerful time-dynamic geographical visualization function modules. For example, CZML is a very important concept in Cesium which is a kind of JSON format string. CZML contains point, line, model, graphic elements and indicates how these elements change over time.

Second, Cesium has high-performance with regard to rendering graphics. It uses WebGL which is based on OpenGL ES 2.0 on the underlying hardware. This ensures the cross-platform and cross-grower aspects of Cesium and efficient rendering of 3D computer graphics.

Third, Cesium provides three different views, 2D, 2.5D and 3D which are convenient for constructing geographical discrete global grid systems in general.

Fourth, as noted, Cesium is open source and has been hosted at Github, anyone can download its toolkit and apply in their own applications.

2.3 Introduction of JSON

When designing an application that communicates with a remote computer, a data format and exchange protocol must be selected. JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write and for machines to parse, especially in concert with D3.js and Cesium. It is a text format that is completely language independent and familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal format

for our system.

The codes above represent the data structure of scatter plots storing data inside JSON document, which hold data using key/value pairs. JSON is able to create multiple key/value pairs together, as shown in this example:

```
{
"element":[
    {"name":"Tx234460d","X":"10","Y":"10",
 "R":"5","color":"#006400",},
        {"name":"Tx234461d","X":"10","Y":"30",
"R":"5","color":"#32CD32",},
        {"name":"Tx234462d","X":"10","Y":"50",
"R":"5","color":"#98FB98",},
        {"name":"Tx234463d","X":"10","Y":"70",
"R":"5","color":"#FFD700",},
        {"name":"Tx234464d","X":"10","Y":"90",
"R":"5","color":"#FF8C00",}
    ]
}
```

In this case, each element in scatter plots contains the information of name, position and color.

2.4 Setting up a Local Server by Node.js

In order to run Cesium applications, it requires a local web server to host our files. I use Node.js for this system. Cesium has no server requirements since it is completely client side. This means any web server that can host static content can also host Cesium. Setting up a web server with Node.js is simple and only takes 3 steps:

(1) Install Node.js from their website and use the default install settings.

(2) Open a command shell in the Cesium root directory and download the required modules by executing npm install. This will create a node_modules directory in the root directory.

(3) Finally, start web server by executing node server.js in the root directory.

(4) The result is shown in Figure 2.1.

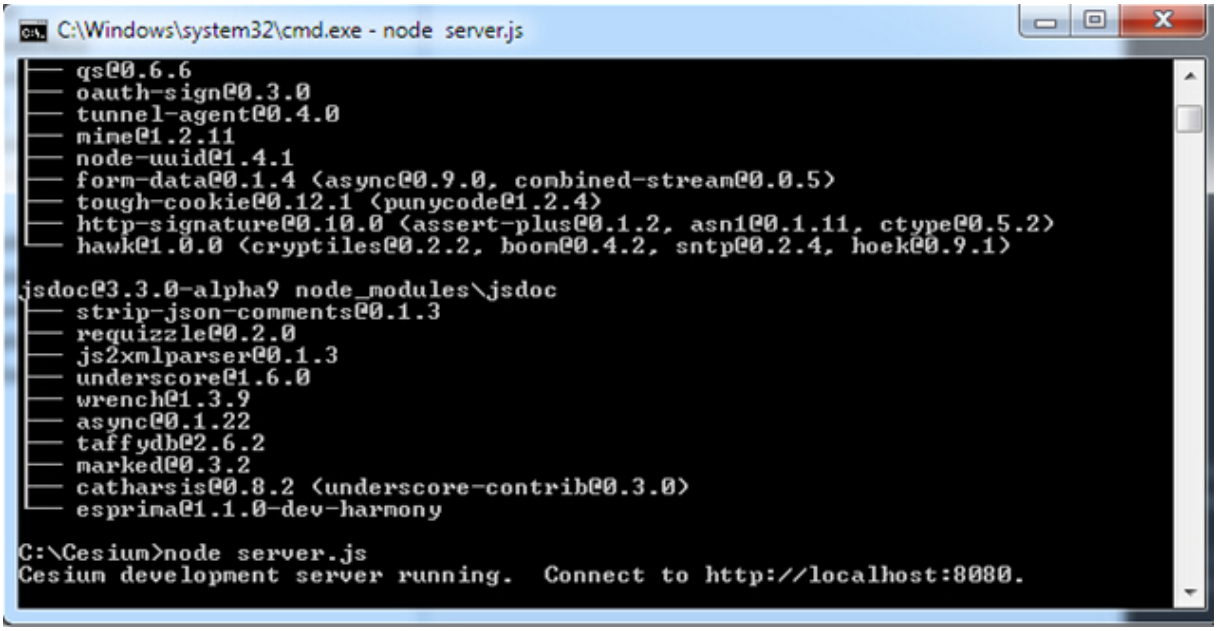

Figure 2.1: Result of setting up local server for Cesium

Chapter 3 2D Space Situational Awareness

One of the crucial part of my space situational awareness system is 2D data visualization. As the information of numerous space borne objects is complicated and unintelligible when viewed as a whole, it is hard to anticipate a crisis and plan maneuvers. Human users are not always proficient at interpreting large amounts of data directly. But the human visual system is adept at reading diagrams and discovering important patterns, assuming the visualization is suitably designed. The essence of data visualization is encoding data into visual forms, reading graphics and the decoding information by others. The goal of data visualization is not simply drawing diagrams, but also revealing information within the data. For example, in my space situational awareness system, the purposes of the two-dimensional charts is visualizing the types and levels of orbital debris threats.

Next, I will describe the use of D3.js for visualization of the space data.

3.1 Data Binding in D3.js

There are many tasks in mapping data to graphs such as changes in ranges, calculations of interpolation and the computing of layouts. The core principle of D3.js is matching data with visual elements, which is referred to in D3 as data binding. Specifically, D3 abstracts the data visualization into a binding between data and visual elements. A data item is made to correspond to a graphic element and a value corresponds to an attribute of an element. D3 encapsulates this potentially complicated process of binding and lets developers focus on the visual aspects. After data binding, each visual element can be modified and monitored by manipulating the corresponding data. In D3 there are two functions responsible for binding data: data () and datum ().

Function data () binds each value in array with every visual element separately and sets up binding rules. The Function data() can also deal with the situation in which the array length is inconsistent with the number of elements. When the array length is greater than number of elements, D3 affords surplus elements placeholders for binding data. When array length less than the number of elements, D3 has functions pointing to surplus elements for deleting data.

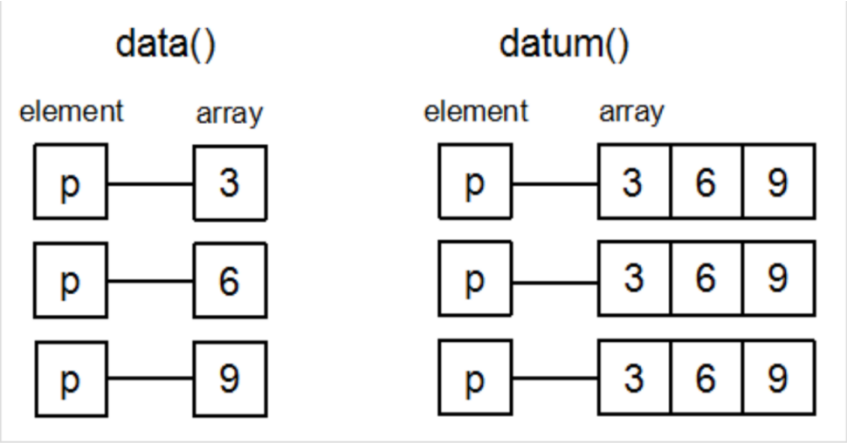

Figure 3.1: Data () and Datum () function diagrams

For example, there are three elements in the body of HTML, as following codes shown.

<body>

GreenCircle

 YellowCircle

 RedCircle

< /body >

GreenCircle, YellowCircle and RedCircle are three elements in HTML which belong to the same class p. Figure 3.1 shows how to use function data() and datum() binding each value in the array to every element in class p. Assuming that the binding array is [3,6,9], If data() function is applied then the first element of p, GreenCircle is binded to 3, the second element element, YellowCircle is binded to 6 and the third element, RedCircle is binded to 9. Unlike data() function, datum() function bind array itself to each element in p which means GreenCircle, YellowCircle and Red Circle bind to the same array [3,6,9].

Data binding is frequently used in my system. For example, when donut charts and scatter plots are created, I use the function data() to bind similar data together. For example, geographical locations of the same debris at various moments should be gathered in one class.

Data is thereby centralized to deal with properties such as changing size and opacity of circle, manipulating proportion of every partition in donut chart (see figures 3.4 and 3.5).

Scatter Plots are one of the simplest dashboards to implement, while D3 does not provide built-in methods to create multiple DOM elements. The solution creates a new instance for every circle. The code is shown below:

```
svg.append("circle")
    .attr("cx", d.x)
    .attr("cy", d.y)
    .attr("r", 2.5);
```

Function append () adds a circle element into canvas at location (d.x, d.y) and with a radius of 2.5. However, D3 provides an entirely new approach to creating multiple DOM elements that can then be used for visualization. D3's design encourages developers to represent what they need but not how to implement visualizations. To clarify this process, let's consider a small example:

```
var circle = svg.selectAll("circle")
                .data(data);
circle.exit().remove();
circle.enter().append("circle")
      .attr("r", 2.5);
circle.attr("cx", function(d) { return d.x; })
      .attr("cy", function(d) { return d.y; });
```

The first line of code (svg.selectAll("circle")) returns an empty set because there is no element yet within the SVG canvas. Second, binding this empty set with data(data) produces three possible statuses: enter, update and exit. If the amount of data is smaller than the current set of visual elements, which are the circles on the page, the D3's exit().remove() will remove extra elements. If the amount of data is larger than the elements, the function enter() one the following line will add new elements into SVG. After applying the enter() and exit() functions, the amount of data and elements are always consistent, no more and no less. The final step is updating the position of circle based on the coming data. The last two lines of

code (circle.attr("cx", function(d) { return d.x; }).attr("cy", function(d) { return d.y; });) implement this operation.

3.2 Workflow of D3.js

The workflow in D3 requires the unification and recombination of different graphic elements and data. The general process is as follows. First, HTML file is the trunk of D3 containing the logic and implementation of code. Other files such as CSS and JSON files store the styles of web pages and the value of elements. Next, all the components in D3 should be selected and declared before visualized on the page. Even if no element is selected in this operation, D3 will create empty collections. The two main components are data and SVG graphic elements. D3 reads json, csv, tsv format files directly and then binds the read data to SVG element properties of the webpage body. The result of binding is a completely new HTML interface. Finally, all that remains is the work of the browser to render the specified graphics and represent data characteristics. In this part, I will discuss this process with an example in complete detail. The flowchart of process is represented in Figure 3.2.

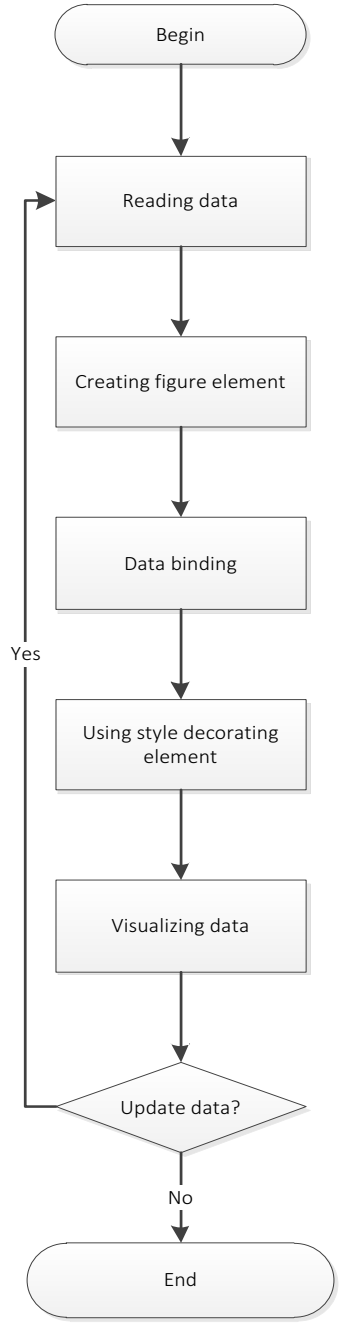

Figure 3.2: Workflow of D3.js

Data is the base of visualization in D3, so in the first step it should read in the necessary data. D3 supports multiple formats of data such as JSON and xml. In my system, I mainly utilize JSON documents to store tags, geographic locations and threat levels of space borne objects. I use the JSON.parse() function to read JSON documents which will be discussed in chapter 5 in further detail.

After obtaining the data, D3 creates graphic elements by using the append() and select()

functions. The select function usually implements data creation and data binding at the same time. That is, the elements beyond those necessary for data() will be erased by the exit() function, and insufficient elements will be added by the enter() function. So far, each element corresponds to its own data.

The third step is decorating elements. D3 normally uses CSS to control the format of the visualization, controlling aspects such as SVG margin and the color of an element. CSS stands for Cascading Style Sheets. It provides pixel-level control of positions to elements on webpages. CSS is a cornerstone technology used by most websites to create visually engaging and stylistically consistent webpages. In my application, I mainly use CSS to control styles of axes, as in the codes shown below:

```
.axis line{
        fill: none;
        stroke: black;
        shape-rendering: crispEdges;
}.
```

The shape-rendering in code above control the style of axes. D3 provides various styles to choose from including optimizeSpeed, geometricPrecision, crispEdges and inherit.

For updating data, D3 repeats this process until the visualization meets its end condition. As the data binding mechanism, graphical elements are changed accordingly in updating data to achieve real-time visualization. Specifically, circle elements and slices in scatter plots and donut charts change their sizes and positions when update button is clicked. Scatter plots generate new spots based on the information in JSON file. Donut charts distribute new proportions for each slice to correspond with updated data.

3.3 Dashboard Design in the Warning System

Two-dimensional dashboards may include a wide range of charts, from bubble charts, stack graphs to tag clouds. Just about every type of chart can be constructed with D3. However, there is no uniform way to visual all data and visualization dashboards need to be adapted to diverse datasets for different situations. Selecting the appropriate type of chart for

the user to meet their requirements is also necessary. Web based two-dimensional dashboards follow the following three design principles even though they may have unique styles [32].

(1) Display the Most Useful Data with No Scrolling

Segmenting charts into multiple pages may impact the performance of visualizations significantly. This requires some effort to filter and integrate data and it becomes particularly important in situational awareness systems. Under sudden threats, the judgment of human operators may decline. In this situation, by scanning the contents of a dashboard, users should be given the most important details in an order that make sense. Only then can dashboards assist users to analyze the threats and find solutions in a timely manner.

(2) Use the Right Dashboard Structure for the Proper Context

Every type of dashboard structure has a unique focus and features, so one or more charts should be selected according to the actual conditions and user requirements. For example, a bubble chart is a type of chart that displays three dimensions of data. Its strength lies in flexible structures without restrictions by axis and the third dimension reflects the value of elements. Therefore, bubble charts are suited to visual data which need comparison without analyzing trends over time. Tag clouds are another chart which is composed of text data. Tags are usually single words and the importance of each tag is shown with font size or color. Therefore, a tag cloud is good for word based visualizations especially when conveying summaries or key points. These two examples further illustrate that the characteristics of context determine dashboard structure.

(3) Highlight Important Data

When users look at a dashboard or graph, their eyes should be drawn to the information that is most important. This means a lot more than data reduction and integration. Important data should be made to stand out. Changing the opacity and color is the simplest way to highlight data.

3.4 Implementation

Previously, I mentioned bubble charts and tag cloud, but they are not suitable for space situation awareness systems. This system must be able to store and integrate historical information, analyze the trends over time and represent severe threats to the user. Based on

these characteristics, I choose scatter plots and donut chart for the 2D dashboard components in my system.

3.4.1 Scatter Plots

Scatter plots are similar to bubble charts for representing three dimensions of data. However, it is anchored by the base coordinate axis and the data is rearranged accordingly. In my system, scatter plots present the changing of threats in various space borne objects which are represented with color and tags and record the history of recent information. The size of circles represents the scale of the threat. Every time data updates, a new list of circles will be generated and the previous elements move forward to the left. The data and attributes of the same space debris are grouped together to modify color and position at the same time. The chart updates data at intervals and users can comprehend the trends of threats at a glance.

Figure 3.3: Scatter Plots in my system

3.4.2 Donut Chart

I also incorporated donut charts in my space situational awareness system. Compared with other charts I discussed earlier, donut charts focus on the proportion of different threats at the same instant. The threats in the system are influenced by distances between spacecraft

and orbital debris. The smaller the distance between them, the more the threat will stand out on the chart which is the proportion of each slice. On account of the data binding mechanism of D3, every slice has real-time changes based on bound data source and corresponding labels will move along the slice as well The text labels are allocated to each slice in the same parent class when they are generated, so that they change their positions with corresponding slices. Figure 3.4 shows the donut chart created in my space situational awareness system.

Figure 3.4 Donut chart in my system.

The code below details how to create polylines in donut charts. The first step is binding the data. Each polyline is assigned a specific value. When data is updated, the position of polyline has a corresponding change. The statement in my code that implements data binding above is data(pie(data), key). Pie(data) which means the value or threat of each slice and key means the label of each slice. So this statement binds each slice with its value and text.

The second step is creating the animation. The following code polyline.transition().duration(1000) defines the duration of animation which is 1000 milliseconds. The D3.interpolate() function controls the interpolation of the animation including the calculation of updated positions. In particular, this function calculates the positions of inter arcs and outer arcs automatically based on the interpolations during this time as the statements var d2 = interpolate(t) and var pos = outerArc.centroid(d2) represent. Finally a statement returns these position values.

The final step is organizing graphical elements and removing isolated elements. The statement polyline.exit().remove(); shows this process, and exit() and remove() functions delete the elements without bound data.

```
Var    polyline = svg.select(".lines")
                    .selectAll("polyline")
                    .data(pie(data), key);
polyline.enter()
        .append("polyline");

polyline.transition()
          .duration(1000)
          .attrTween("points",
                      function(d){
                                  this._current = this._current || d;
                                  var interpolate = d3.interpolate(this._current, d);
                                  this._current = interpolate(0);
                                  return function(t) {
                                          var d2 = interpolate(t);
                                          var pos = outerArc.centroid(d2);
                                          pos[0] = radius * 0.95 * (midAngle(d2) < Math.PI ? 1 : -1);
                                          return [arc.centroid(d2), outerArc.centroid(d2), pos];
                                          };
                                 });
polyline.exit()
          .remove();
```

Compared with scatter plots, donut charts reflect the priority of current threats more directly and help users to find the most urgent issues. In addition, since donut charts only change the proportion of each slice and do not generate new slices in an update, it clearly reflects the transformation of each threat over time. People are more focused on the primary threat in an emergency so due to reasons discussed above I choose donut chart in my situational awareness system.

Chapter 4 3D Space Situation Awareness System

A 2D situational awareness system can be implemented by using D3.js which was discuss in the previous chapter. However, the ultimate goal of my system is the combination of 2D situational awareness and 3D situational awareness. For my 3D situational awareness component, I decided to implement it with the Cesium framework.

4.1 Geographical Discrete Global Grid System

Cesium is a technical visualization tool that can help us create 3D globes, appropriate for visualizing the real-time position and threats of Space Borne Objects in my 3D situation awareness system. Traditional approaches represent the earth as a lattice of points by connecting latitude and longitude, for example the 2.5×2.5, 5×5, and 10×10 NASA Earth Radiation Budget Experiment (BRBE) grids described in Brooks(1981) [33]. However, the most serious problem of this method is the unequal distribution of regions[9]. Areas close to the poles are less precise than the regions nearing the equator. Applying Geographical Discrete Global Grid Systems (DGGS) is a feasible alternative method which has been be widely used in commercial and academic systems such as optimum path determination systems (Stefanakis and Kavouras 1995) [10] and global climate modeling systems [11].

Geographical Discrete Global Grid Systems (DGGS) exhaustively partition the globe into equal, hierarchical cells. Each cell can change texture, color and transparency independently with a unique identifier or indexing. Cesium provides an interface to convert 2D and 3D views so that any manipulation on canvas can reflect on the 3D globe immediately. Many systems implement 3D visualizations in this way. For example the Hex Plent Technical Demo (by Joel Davis) [9] map divisional grid textures on globes.

In general, the three forms of discrete global grids are square grids, hexagonal grids and triangular grids[12]. Square grids are most widely used due to their simplicity, easy processing and computational efficiency in small to medium span ranges[13]. However, many war simulation games and geographic information visualization tools use hexagonal grids instead

of square grids because squares share an edge with four neighbors but also touch another four neighbors at just one point. This weakness complicates movement along grids because diagonal movements are hard to weight properly with integer movement values[14]. This problem can be solved easily in hexagonal grids because the distances between neighboring hexagons are always equal[15]. Other advantages of hexagonal grids are the high space utilization and small perimeter-to-area ratio which makes visualization more efficient[16]. For these reasons, I chose hexagonal grids for the discrete global grids.

4.1.1 Basic Concept of a Hexagon

Hexagons are 6-sided polygons with the same length edges. According to the orientations, hexagons can be divided into two typical forms horizontal hexagons (pointy topped) and vertical hexagons (flat topped). In the system, I choose horizontal hexagons grid which is conventionally used by most researchers.

4.1.2 Size and Spacing

In a geographical DGGS system, a hexagon cannot appear by itself. They must form a group and interlace with each other. In the horizontal hexagons, as figure 4.1 shows, the height of a hexagon is double in size. The size means the distance between central point and each vertex. The vertical distance between two neighboring hexagons is three-fourths times the height. The width of a hexagon is sqrt(3) times size. The horizontal distance between adjacent hexes is width.

The key to generating a hexagon grid is determining the position of central points in each hexagon of the grid. Because by using the basic concept of hexagon above, it is feasible to calculate the position of each vertex in the hexagon and thereby draw the boundaries.

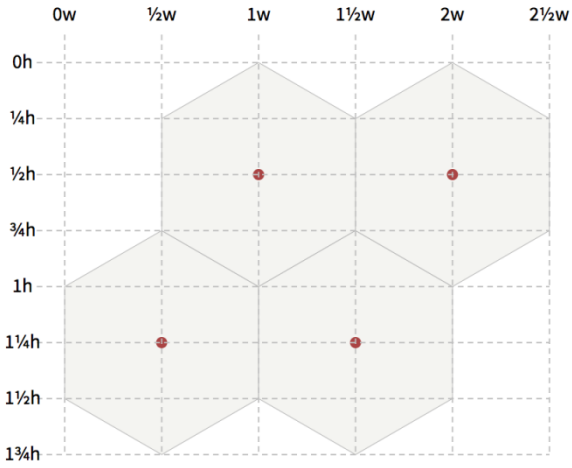

Figure 4.1 Geometry property of hexagon

The computational formulas are as follows:

height = size * 2.

vertical distance = 3/2 * size.

width = sqrt(3) * size.

horizontal distance= sqrt(3) * size.

4.1.3 Base Platonic Solid

In this section I will introduce how to create hexagons on the sphere. At first, making it easier to calculate, I approximately consider the earth as sphere. Then, I create a hexagonal grid on plane and map it to sphere.

The five platonic solids are shown in Figure 4.2[17]. In general, the number of faces in the base platonic solid is proportion to the precision of projecting between a face of the polyhedron and the corresponding spherical surface. The tetrahedron and cube have the smallest number of faces (4 and 6 respectively), thus they are relatively poor substitution for the sphere[18]. I choose icosahedron because it has the greatest number of faces (20) and therefore projections defined on it tend to have relatively higher accuracy and less distortion.

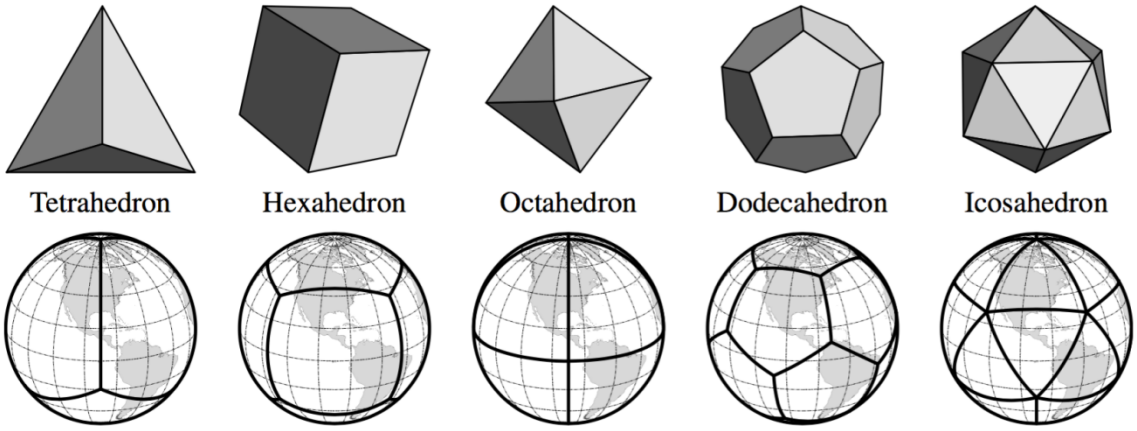

Figure 4.2 The platonic solids: the tetrahedron, cube, octahedron, dodecahedron, and icosahedron [9]

4.1.4 ISEA Map Projection

Base on the platonic solids conversion method, one of the most classic map projections is the Icosahedral Snyder Equal Area (ISEA). ISEA was first introduced by John P. Snyder [19], which is an equal area projection from icosahedron to sphere. ISEA attempts to solve the mapping projection problem and it is the foundation of the grid system.

However, the faces of an icosahedron are inadequate, if we use a high-accuracy polyhedral fitting method, we are continually subdividing the icosahedron. For example, Joel Davis[11] uses OpenGL to make recursive triangle subdivision on icosahedrons. After two iterations, the degree of fitting is improved significantly, as figure 4.3 shows. With the increasing of iteration numbers, the surfaces of the polyhedron become more smooth.

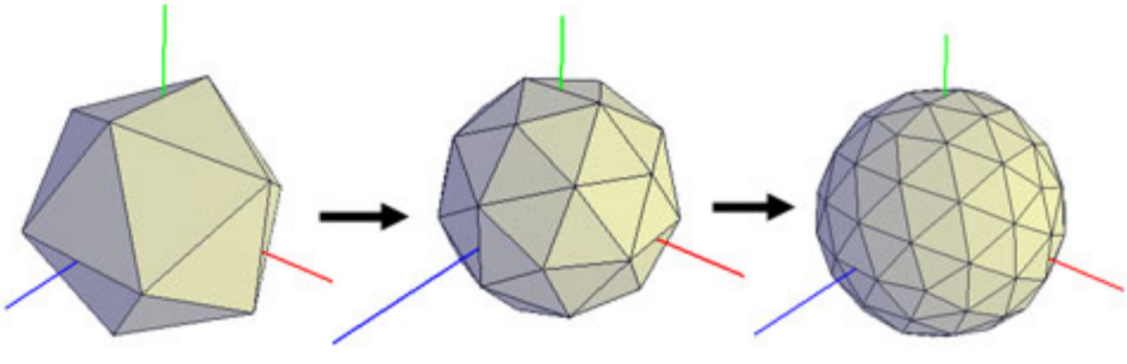

Figure 4.3 Surface subdivision start with icosahedron[11]

4.2 Naive Approach for Hexagon Grid Creating

There are many ways to create hexagon grids on the plane. One of the most commonly used regular DGGs are those based on the geographic (latitude-longitude) coordinate system. Hexagons are tightly spaced at the surface of earth based on the latitude and longitude coordinates.

As mentioned in the previous chapter, the horizontal distance between the center of two adjacent hexagons is sqrt(3) times size and the vertical distance is three-seconds times size. Hexagons are generated from the center to the boundary of canvas based on this method.

However, the distribution of latitude and longitude is asymmetric in different regions. It is conspicuous that the latitude and longitude is sparsely distributed at the equator and concentrated at the poles. The opposite of this is every hexagon has homogeneous alignment on the canvas. The discrepancy leads to distortion on the poles when mapping the plane to the sphere which means incurring serious damage to visualization precision.

Moreover, owing to the height and width of the canvas, one cannot guarantee integral multiples of hexagons, it has inevitably leads to incomplete hexagons existing on the border of the canvas and certain areas not covered by hexagons.

Therefore, the naive approach of hexagon grid generation is incompatible with our requirements. Compared with the equator, the distortion at the poles is too obvious which seriously influences the visualizations. Moreover, this method does not guarantee hexagon grid coverage throughout the sphere. Even just a small omission is unacceptable for the awareness system.

4.3 Spatial Partitioning Method

According to preceding ISEA method, the triangles on icosahedron can map to the surface of sphere. So the spatial partition on sphere is therefore a hexagon substitution on triangle faces of plane.

4.3.1 Two Types of Classic Hexagon Partition Methods

In the situation awareness system I use two hexagon partition methods. One is the basic method which is performed on the spherical triangle by connecting the midpoints of edges with the center and then recursively performing the same operation on each of the resulting triangles as figure 4.4 shows.

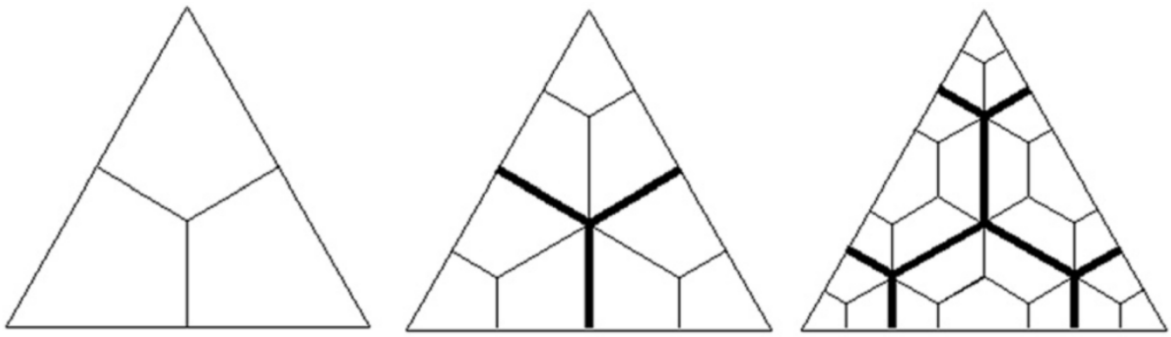

Figure 4.4 Three levels of an aperture 4 Class I hexagon subdivision.

This technique was used by Fekete and Treinish which is the most simple and popular method of subdivision (aperture 4 Class I hexagon subdivision). Thuburn (1997) [20] apply and calculate this approach on spherical triangles to define the global hexagon grid.

However, this method has limitations. Each arc on the earth can be treated as lines approximately based on the ISEA map projection mentioned above. We perform it directly on the spherical surface. So the arcs correspond to cell edges on the planar face. The more points on the edge, the less error in coordinate conversion. Furthermore, a number of researchers have attempted to adjust this method to meet application-specific criteria. For example, my space situational awareness system requires the cell regions of each discrete grid resolution to be equal in area. However, the grids discussed above do not have this property. So, more complicated methods are needed to form spherical partitions.

William (1968) [21] proposed the second hexagon subdivision method (aperture 4 Class II hexagon subdivision) by aggregating the cells of the aperture 9 triangle grid as figure 4.5 shows. The hexagon is available by connecting the third point of each edge. While, there are too many hexagons generated at a time which impose heavy computational requirements.

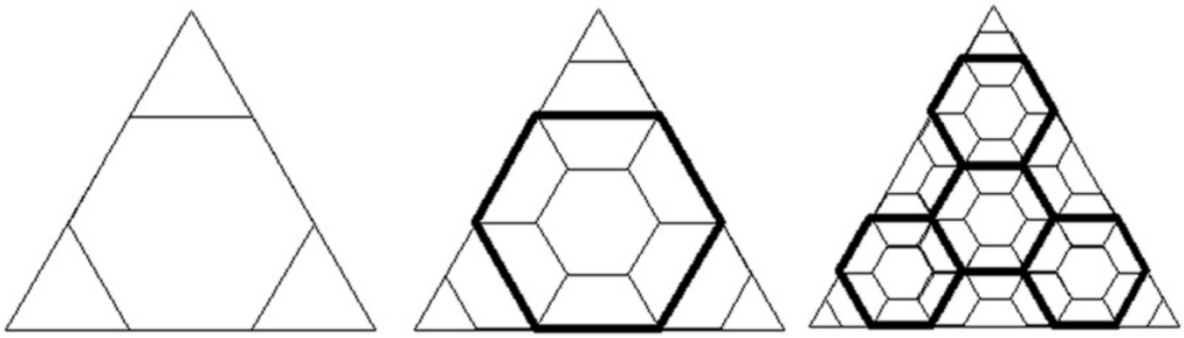

Figure 4.5 Three levels of an aperture 4 Class II hexagon subdivision.

4.3.2 Improved Hexagon Partition Method

Icosahedral Snyder Equal Area Aperture 3 Hexagonal Grid is the integration of two types of partitions which can take full advantage of both. The orientation of hexagon grids alternates in successive resolutions between Class II and Class I which is shown in figure 4.6. Gray and black hexagons represent the hexagon subdivision method of Class II and Class I, respectively. The advantage of this approach is it is processing two subdivisions in one iteration which ensure the high space utilization and little surface distortion.

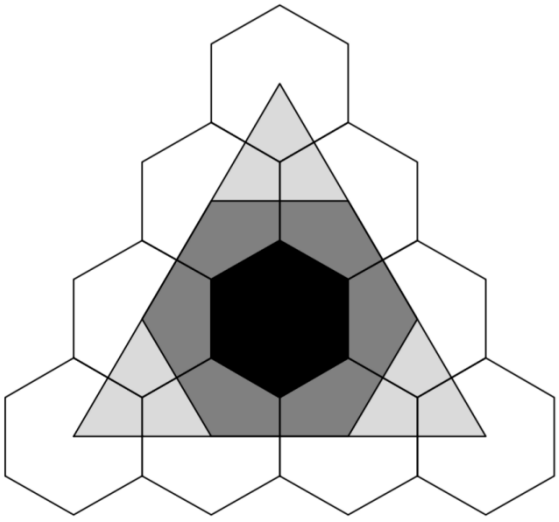

Figure 4.6 Improved hexagon subdivision

4.3.3 The Process of Spherical Hierarchical Subdivision

Spherical hexagon subdivision has been explored by academicians for many years. The earliest research dates back to 1944 which was advanced by German Fuller [34] . The process is as follows.

First, the sphere should be treated as a polyhedron. In this thesis, I choose the icosahedron as the basis of the spherical grid which makes it simple to connect poles with both sides of equator on the virtual earth. The rendering behavior initially draws twelve surface triangles corresponding to faces of the icosahedron.

Second, we convert Latitude/Longitude and Cartesian coordinates to determine the central point of each hexagon.

Third, based on the geometric property of hexagons introduced above, the coordinates of vertices in each hexagon can be calculated by the position of central points, and thus we can obtain the coordinates of points of trisectors in each edge. Then, we apply the aperture 4 Class II algorithm by connecting every point of trisectors in the edges of hexagons. As figure 4.5 shows, the hexagon is divided into a smaller hexagons and three triangles.

Fourth, we consider the subdivided hexagon as six triangles, because the core of aperture 4 Class I algorithm is subdivide triangle. Specifically, this algorithm connects the midpoint of each edge with the central point of triangle, as figure 4.4 shows.

We repeat step two, step three and step four to generate more triangles until the required level of sub-division is met. The flow chart of this process is figure 4.7.

The goal of applying hexagon algorithms is calculating the geographic location of each hexagon. This is the key point, since a geographical discrete hexagon grid system is generated based on it by applying geometric knowledge of hexagons.

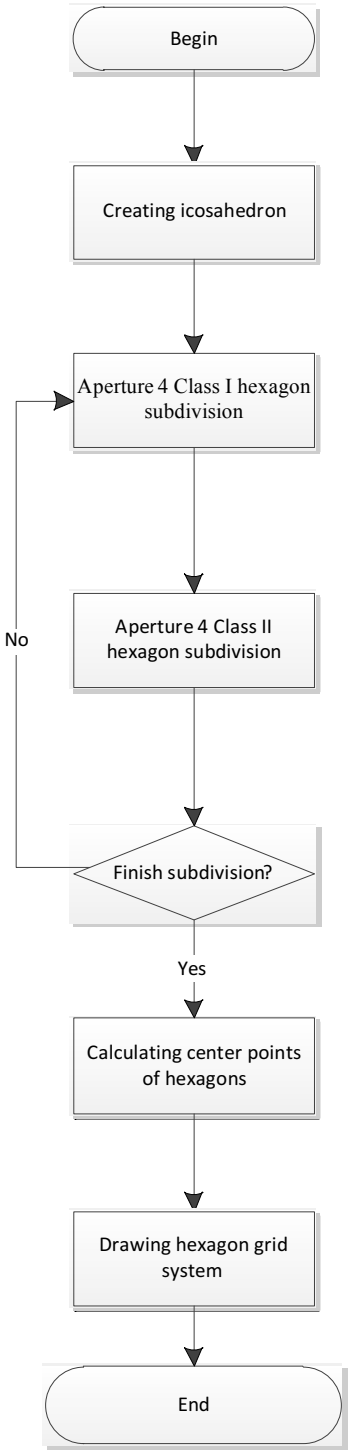

Figure 4.7 Spherical Hierarchical Subdivision

4.4 Converting between Latitude/Longitude and Cartesian coordinates

To apply this improved hexagon partition method, some geographic and mathematical formulas are needed. Whether the system determines the midpoint and point of trisector of the hexagons or the centroid of spherical triangle, it has to go through a conversion between Latitude/Longitude and Cartesian coordinates. Although there are unavoidable subtle errors in the conversion process, it is in an acceptable range. These conversion formulas are easy to use in JavaScript environments and greatly simplify geometry computation [22].

φ and θ are two important parameters included in converting formulas which are angle from north pole down to latitude and angle from Greenwich to longitude, respectively, as figure 4.8 shows.

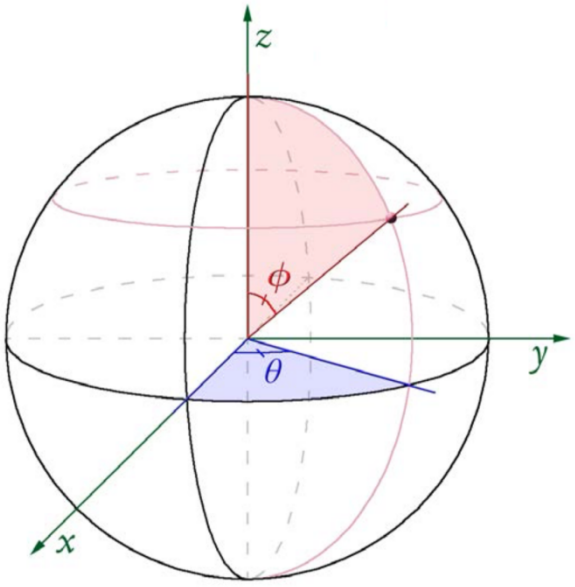

Figure 4.8 φ and θ in coordinate transform formula

The formulas of converting Latitude / Longitude to Cartesian coordinate on the Sphere are:

$$\phi = \frac{(90 - \text{Latitude})\pi}{180} \qquad \theta = \frac{(\text{Longitude})\pi}{180}$$

$$X = R\sin\phi\cos\theta$$
$$Y = R\sin\phi\sin\theta$$
$$Z = R\cos\phi$$

φ and θ are functions of latitude and longitude which can easily be computed. The Cartesian

coordinates X, Y and Z are trigonometric functions of φ and θ. In practice, R is usually assigned a value such as 6371, while the value will not impact on the result due to the reduction in calculation Because, the Cartesian coordinates generated by formulas above have to be converted back to Latitude / Longitude coordinates for visualizing in the end. In these processes, R is reduced.

4.5 Implementation

For testing purposes, I performed the experiment many times with different iterations of hexagon partitioning. As figure 4.9 and 4.10 show, hexagons of the fourth iteration are generated from external hexagons of the third iteration. It is worth noting that the performance of distortion on the pole is good in both iterations. There are no differences in outcomes among hexagons in different regions.

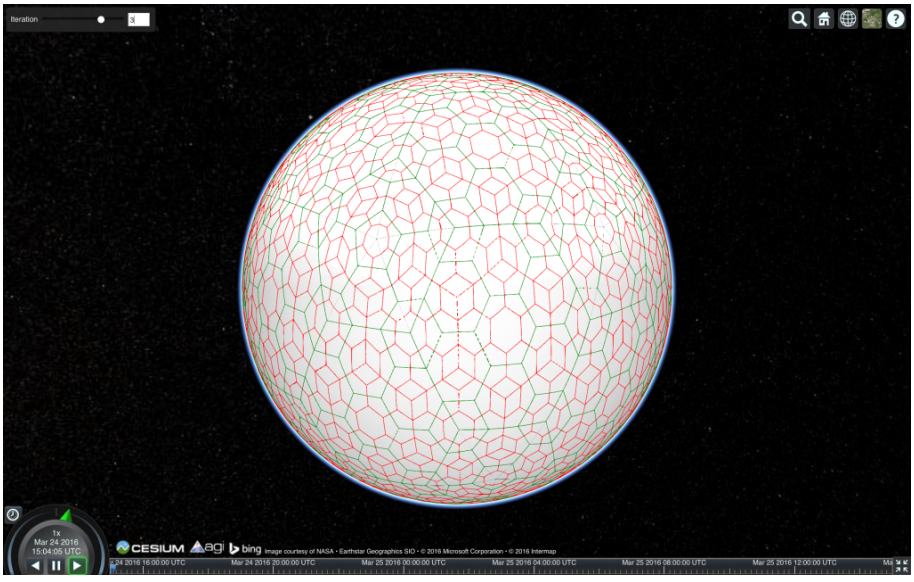

Figure 4.9 Three iterations of hexagon partition

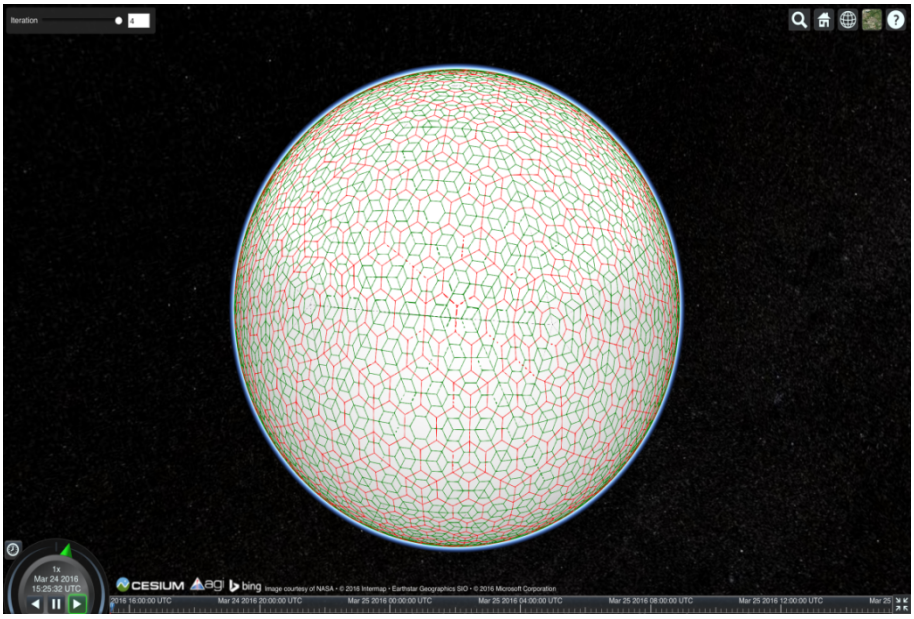

Figure 4.10 Four iterations of hexagon partition

Chapter 5 Cooperation Between 2D and 3D Components

I introduced how to create a geographical discrete hexagon grid system with cesium in the previous section. However, generating a grid system is just preparatory work, my ultimate goal is visualizing threats in the situation awareness system which should be synchronous with 2D dashboards.

Cesium and D3.js are able to synchronous with each other, because both of them are developed with JavaScript. By referencing Cesium and D3.js libraries in the same HTML page, users can use any functions without an issue. Thus, incoming data is shared by dashboards and the generated virtual earth. In this part I will introduce how to implement the final visualization system as a cooperating system.

5.1 Parsing JSON Document in D3

Modern browsers support the JSON.parse() function in D3 which is the recommended way to parse a JSON document. If the input string does not meet the JSON standard, error messages can be given to users.

The first parameter of JSON.parse() is the incoming string. The second parameter of JSON.parse() is a function. This function has two arguments: name and value. When passing a JSON string, each group should call this function. The return value of the function will be assigned back to the current name.

```
var str = '{"tag": TX00086, "height": 20000, ”latitude”: 36.5, ”longitude”: 108.1 }';
var obj = JSON.parse(str, fun);
function fun(name, value){
        if( name == " TX00086" )
                value = 30000;
        return value;
}
```

The code above represents how to change and get a value from a JSON document. In my JSON documents, data is recorded as key-value pair including the name, height, latitude and longitude of space borne threats. I create an instance for each kind of space borne asset. Every time D3 gets the new incoming data, it will call the built-in function to update the data.

5.2 Space Situational Awareness System Framework

The Space Situational Awareness System is divided into two parts (Figure 5.1). The first part is a 2D visualization that includes a donut chart and scatter plots primarily to represent the identities and threats of space debris. Users can quickly find out the biggest present threat and predict trends by analyzing the scatter chart. The second part is 3D visualization framework implemented on the discrete global grid system of virtual earth. The system parses incoming data to determine the geographic locations of space trash, to present them on the global grid. The specific architecture map is below.

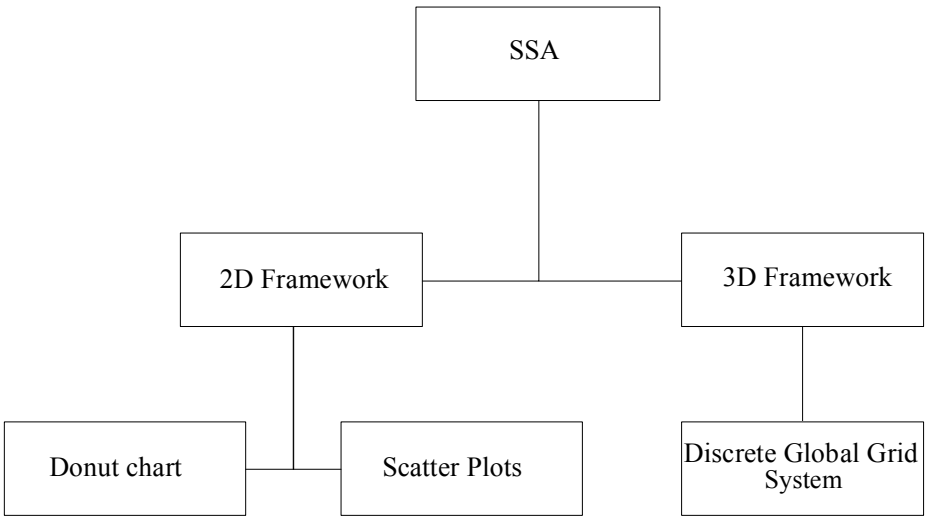

Figure 5.1 Space situational awareness system framework

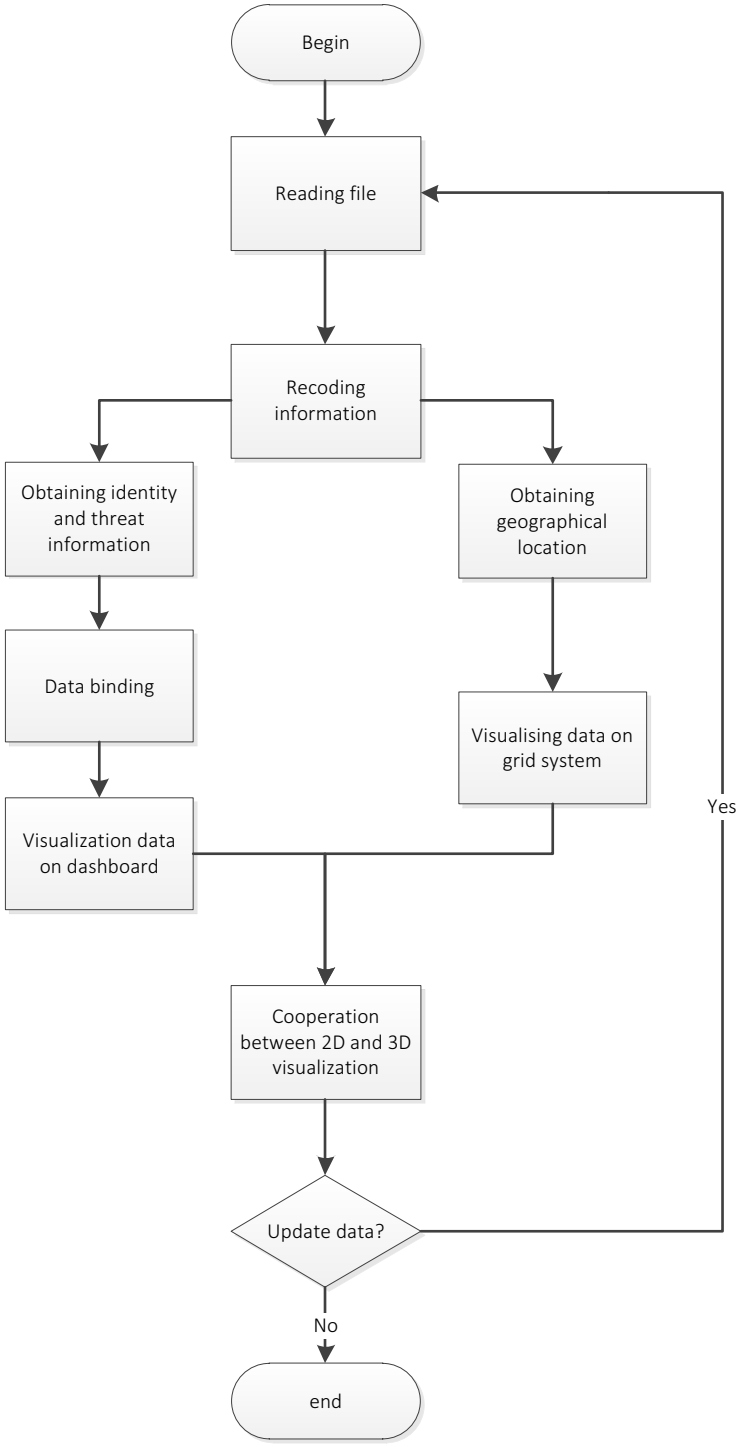

Figure 5.2 Visualization process of space situational awareness system

The flow chart above (Figure 5.2) represents the entire visualization process of my situational awareness system. For the purpose of testing the system, I create some random data stored in JSON documents. I also create instances for each space borne object to store specific information. So when the incoming data arrive, each instance updates its own internal

information to change the visualization of the system. The process is as follows.

First, I parse threat data in the JSON documents and obtain identification and geographical information of space borne objects.

Second, dashboards in the 2D visualization framework acquire identification and position information of space borne objects and visualize them on the donut chart. Because of data binding in d3.js, updating data can be performed in the system in real time. At the same time, the 3D visualization framework acquires latitude and longitude information of space borne objects. Each hexagon on the discrete global grid system represents an area, if one of the objects belongs to this area, the system changes the color and transparency of this hexagon based on threat levels.

Third, if the user clicks the update button, the system will generate other random data for visualizing and repeats this process for the purpose of demonstration.

Figure 5.3 represents the final orbital debris threats visualization system.

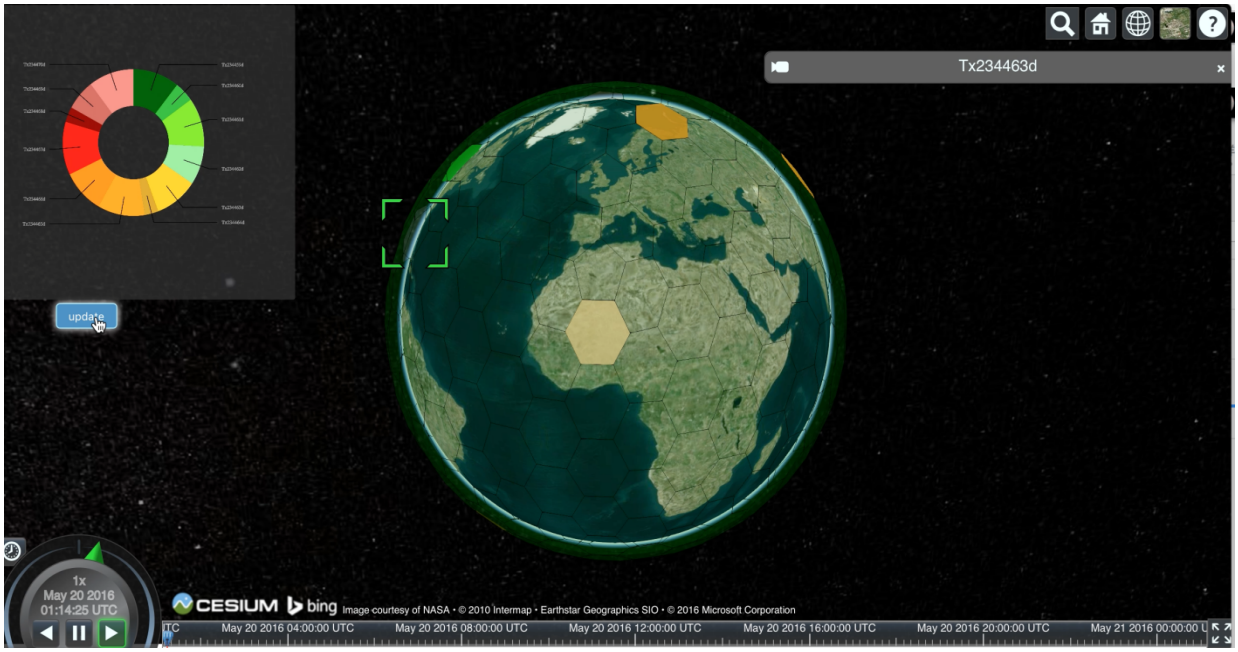

Figure 5.3 Final orbital debris threats visualization system

Chapter 6 System Test

Testing is an essential part of software development. It runs through the entire development process when an key part of the development is finished. It incorporates stability testing, security testing and compatibility testing. The purpose of the test is to verify that the system meets the needs of users and is robust. This chapter introduces the idea of testing the system, test methods and specific test procedures.

Test consists of two main aspects: UI instance test and Geographical Discrete Global Grid System test. All tests were conducted on the same system with the following specifications:

CPU: 2.2GHz Intel Core i7

Memory: 16GB 1600 MHz DDR3

GPU: Intel Iris Pro 1.5GB

Operation System: OS X 10.11.3

For the UI instance test, the content is testing the operation of the dashboard as well as the interactive animation of the donut chart, as table 6.1 shows

Table 6.1 UI instance test

Instance Number	Goal	Preset Condition	Operation	Expected result	Test result
001	If the buttons and dashboards work well. If the interactive animation work well.	Space situational awareness works well	Click update button	Animation and dashboard display correctly	Success

For the Geographical Discrete Global Grid System test, the content is testing the distortion of hexagon grids as well as the operating speed of the system. Because all hexagons are generated recursively, distorted hexagons are distributed in the grid regularly. In this test, I determine either the hexagon is distorted or not by observing the edges. If each edge of the hexagon is straight line, it is regular, otherwise the hexagon is distorted. Table 6.2 shows the results.

Table 6.2 Geographical discrete global grid system test

Instance Number	**Goal**	**Preset Condition**	**Operation**	**Expected result**	**Test result**
002	If the grid system work well. If the distortion is not apparent.	Using improved hexagon partition method with one iterations	Start generate grid system	Each hexagon with the same size. Reasonable time consumption	Time consumption: 1 second. 98 hexagons are generated. 45 hexagons have distortions. CPU utilization 35%. GPU utilization 37%.
003	If the grid system work well. If the distortion is not apparent.	Using improved hexagon partition method with three iterations	Start generate grid system	Each hexagon with the same size. Reasonable time consumption	Time consumption: 5 seconds. 294 hexagons are generated. 98 hexagons have distortions. CPU utilization 50%. GPU utilization 45%.

004	If the grid system work well. If the distortion is not apparent.	Using improved hexagon partition method with four iterations	Start generate grid system	Each hexagon with the same size. Reasonable time consumption	Time consumption: 22 seconds. 882 hexagons are generated. 196 hexagons have distortions. CPU utilization 75%. GPU utilization 62%.

By observing from the above table, we see that the upper limit of the hexagon partition iteration is four. As I execute the fourth iteration, the CPU utilization is over 75%, and it is difficult for my computer to show images effectively and the operations are not smooth. In the end, Cesium sends back an error about excess calculations.

So the best result appears at the third iteration with the minimum distortions. According to my statistics, there are 50 percent of hexagons are distorted in one iteration partition method, 30 percent hexagons are distorted in three iterations partition method and 22 percent hexagons are distorted in four iterations partition method. Even though the four iteration partition method produces the best results, it is beyond the calculation capacity of CPU. So considering the balance of accuracy and calculations, I choose three iterations partition method for my situational awareness system. As hardware improves in the future, one can adjust the trade off accordingly.

Chapter 7 Conclusion and Future work

7.1 Conclusion

In this thesis, I designed and implemented a space situation awareness system to visualize orbital debris threats. This system is composed of two sections: a two-dimensional dashboard and a three-dimensional virtual earth. These two parts collaborate together to accomplish the overall visualization.

I implemented a geographical discrete global grid system on the virtual earth within Cesium. Based on the analysis of several spatial partitioning methods, I adopted an improved hexagon partition method which is the integration of two classic partition technologies. This trade-off provides a balance between precision and complexity.

In my SSA system, I implemented a collaborative operation between the two-dimensional and three-dimensional visualizations and incorporate the spatial partitioning method and dashboard design principles. Finally, using these methods, I Improve the performance of distortion on the pole of the virtual earth.

7.2 Future work

There are several aspects of my system that may be improved. First, for testing purposes I use random data in the system. In a practical application, the system should have the ability to receive real-time data and filter incoming data.

Second, the maximum iteration of hexagon partitioning is four, due to the computing capacity of my computer. More optimized hexagon partition algorithms which can reduce the amount of calculations could be investigated.

References

1. AGI Space Situational Awareness. Retrieved from: http://www.yumpu.com/en/document/view/34597976/space-situational-awareness-agi,2015
2. MikeRostock, The Wealth and health of Nations, https://bost.ocks.org/mike/nations/, March 13, 2012.
3. Interactive satellite viewer, Retrieved from: http://science.nasa.gov/iSat, 2013.
4. Dr. Sun, 4DChoroplMap, https://cesiumjs.org/demos/4DChoroplethmap.html, April 10, 2012.
5. Mikan Stamen Kovich, Robert Creer, Google Earth Navigation Information System, 19-22 Sept. 2011, OCEANS'11 MTS/IEEE KONA.
6. McCall, N. D. (2006). A New World Map on an Irregular Heptahedron (Master's thesis, University of Redlands). Tellus20(4):642-53.
7. Mohammad Imrul Jubair, Multiresolution Visualization of Digital Earth Data via Hexagonal Box- Spline Wavelets, IEEE Scientific Visualization Conference (SciVis), 2015, 25-30 October, Chicago, Il, USA.
8. Public Policy with the Space Situational Awareness, Retrieved from: https://www.spacefoundation.org/programs/public-policy-and-government-affairs/introduction-space-activities/space-situational, 2014.
9. Kevin Sahr, Denis White, A. Jon Kimerling, Geodesic Discrete Global Grid Systems, Cartography and Geographic Information Science, Vol. 30, No. 2, 2003, pp. 121-134.
10. Stefanakis, and M. Kavouras. 1995. On the determination of the optimum path in space. In: Frank, A, U and W. Kuhn (eds), Proceedings of the 2nd international conference on spatial information theory (COSIT 95) Semmering, Austria, September 21-23, New York, New York: Springer. pp.241-57.
11. Joel Davis, Hex Planet Technical Demo, 2005, Retrieved from: http://vickijoel.org/hexplant/Hex%20Planet%20White%20Paper.pdf

12. William, D. L. 1968. Integration of the barotropic vorticity equation on a spherical geodesic grid. Tellus 20(4):642-53.

13. Jamesina J. Simpson, Student Member, Efficient Modeling of Impulsive ELF Antipodal Propagation About the Earth Sphere Using an Optimized Two-Dimensional Geodesic FDTD Grid, IEEE Antennas and Wireless Propagation Letters, 1536-1225/04.

14. Mohammad Imrul Jubair, Multiresolution Visualization of Digital Earth Data via Hexagonal Box- Spline Wavelets, IEEE Scientific Visualization Conference (SciVis), 2015, 25-30 October, Chicago, Il, USA.

14. Matthew J. Gregory, A comparison of intercell metrics on discrete global grid systems, Computers, Environment and Urban Systems, 32, (2008), 188–203.

16. Matthew Guenette, A. James Stewart, Triangulation of Hierarchical Hexagon Meshes, SPM '08 Proceedings of the 2008 ACM symposium on Solid and physical modeling, 307-313.

17. Huiping Zhao, Xiongjun Fu, Research on the visibility of low-orbit debris using space-borne rader, IET Radar, Sonar & Navigation, 31-37.

18. Prabukumar Manoharan, Bimal Kumar Ray, An Alternate Line Drawing Algorithm on Hexagonal Grid, Computers and Graphics, 13(4):461-469.

19. Yuki Igarashi, Takeo Igarashi, Interactive Hexahonal Mesh Editing for Beadwork Design, SIGGRAPH Asia 2012, Singapore, November 28-December 1, 2012.

20. Thuburn, J. (1997). A PV-based shallow-water model on a hexagonal-icosahedral grid. Monthly Weather Review, 125:2328-2347.

21. Williamson, D.L. (1968). Intergration of the barotropic vorticcity equation on a spherical geodesic grid. Tellus 20(4):642-653.

22. Jeff Jenness, Tools for Graphics and Shapes, June 2011.

23. Document Object Model (DOM), Retrieved from: https://www.w3.org/DOM/

24. Space debris, Retrieved from:https://en.wikipedia.org/wiki/Space_debris

25. William Playfair, Playfair's Commercial and Political Atlas and Statistical Breviary,

September 2005, ISBN: 9780521855549

26. Edward R. Tufte (2001). The Visual Display of Quantitative Information. Chesire, CT: Graphics Press, p. 44.

27. R.M. Pickett and G.G. Grinstein, "Iconographic Displays for Visualizing Multidimensional Data," Proc. IEEE Conf. Systems, Man, and Cybernetics, pp. 514-519, 1988.

28. D3: Data-Driven Documents, Michael Bostock, Vadim Ogievetsky, Jeffrey Heer ,IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis), 2011

29. ManyEyes: a Site for Visualization at Internet Scale, Fernanda B. Viegas, IEEE Transactions on Visualization and Computer Graphics (Volume:13 , Issue: 6), Nov.-Dec. 2007

30. Cesium-WebGL Virtual Globe and Map Engine, Retrieved from: https://cesiumjs.org/

31. Cesium - An open-source JavaScript library for world-class 3D globes and maps, Retrieved from: https://github.com/AnalyticalGraphicsInc/cesium/

32. A Guide to Creating Dashboard People Love to Use, November 2009,Juice, Retrieved from: http://www/juiceanalytics.com/poster/

33. Brooks, D.R. 1981.Grid system for Earth radiation budget experiment applications. NASA Technical Memorandum 83233. 40 pp.

34. Tong Xiaochong, The Expression of Spherical Entitles and Generating of Voronoidiagram Based on Truncated Icosahedron DGG, Geomatics& information Science of Wuhan University, 2006, 31(11):966-970