NASA/TM-20205000040

On Development of Three-Dimensional Visualization Capabilities in Glenn Research Center Communication Analysis Suite

Lucas D. Shalkhauser, Eric Henderson, and Bryan W. Welch
Glenn Research Center, Cleveland, Ohio

July 2020

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Technical Report Server—Registered (NTRS Reg) and NASA Technical Report Server—Public (NTRS) thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers, but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., "quick-release" reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 757-864-6500

- Telephone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Program
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM-20205000040

On Development of Three-Dimensional Visualization Capabilities in Glenn Research Center Communication Analysis Suite

Lucas D. Shalkhauser, Eric Henderson, and Bryan W. Welch
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

July 2020

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
703-605-6000

This report is available in electronic form at http://www.sti.nasa.gov/ and http://ntrs.nasa.gov/

On Development of Three-Dimensional Visualization Capabilities in Glenn Research Center Communication Analysis Suite

Lucas D. Shalkhauser*, Eric Henderson†, and Bryan W. Welch
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Summary

With NASA's mission to return to the Moon sustainably by 2024 and using that success as a means to step onto the barren world of Mars, it remains important to conduct research and planning as thoroughly and efficiently as possible. In a mission as complex as landing humans on another celestial body, a network of orbiting satellites and ground stations must accurately and reliably communicate with each other, enabling crucial data communications throughout the mission. Visualizing this important data communications increases the understanding of the data and can accelerate analyses efforts. To help with this, three-dimensional (3D) visualization software was developed allowing a 3D representation of various communications systems to be created. The purpose of this software development was to create an interactive visualization with data exported from MATLAB® (The MathWorks, Inc.) scripts in the Glenn Research Center Communication Analysis Suite (GCAS) that is easy to understand, can show all necessary data, and display the data accurately. The main types of data to visualize are from the State Propagation, Line-of-Sight (LOS), and Dynamic Link Margin (DLM) scripts. These all show positions and orbits of satellites and ground stations, while the LOS data also shows when the satellites and ground stations have the ability to communicate with each other based on their respective antenna positions and fields of view. Additionally, the DLM mode color codes the communications link performance onto the LOS access lines. Visualization requires a graphics language that is easily accessible, contains required features, and can easily read data produced by the GCAS MATLAB® scripts. Three.js, a graphics library for Web Graphics Library and coded in JavaScript, was selected for this visualization. The software development was to convert the data from MATLAB® to JavaScript by implementing a MATLAB® function converting the output data of the scripts to a JavaScript Object Notation file. A key part of the software development was creating the visualization within JavaScript and Three.js to visualize any combination of planets, moons, orbits, satellites, ground stations, and LOS links and handle future features without changing major parts of the code. The current visualization capability runs directly from MATLAB® and can dynamically create any scene. This software development currently supports the lunar communications analysis underway by NASA, and can be easily expanded upon in the future to aid any analysis requirements to help plan current and future space missions.

Nomenclature

3D	three dimensional
CSS	Cascading Style Sheets
DLM	Dynamic Link Margin

*Summer Intern in Lewis' Educational and Research Collaborative Internship Program (LERCIP), undergraduate at Baldwin Wallace University.
†Summer Intern in Lewis' Educational and Research Collaborative Internship Program (LERCIP), undergraduate at The Ohio State University.

FOV	field of view
GCAS	Glenn Research Center Communication Analysis Suite
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
LOS	Line-of-Sight
NaN	not a number
WebGL	Web Graphics Library
XAMPP	Cross-platform, Apache (The Apache Software Foundation), MariaDB (MariaDB Foundation), PHP (The PHP Group), and Perl (Perl.org)

Introduction

There are many factors at play during a mission to safely travel to another planetary body. One such factor is the ability to accurately simulate communications satellite orbits and vehicle trajectories. This is necessary and important as the Agency determines the architecture for upcoming missions, such as revisiting the Moon or traveling to Mars. The ability to take massive amounts of simulation data and analysis and condense it into an interactive graphical simulation can assist in understanding the data, as well as allow for faster testing of parametric analyses. The Glenn Research Center Communication Analysis Suite (GCAS) is a powerful tool to simulate the potential orbit of spacecraft, such as the Lunar Gateway, allowing the Agency to optimize all aspects of the mission in an easily understood visual representation.

Simulating and analyzing space communications is an essential part of planning and executing a successful mission. When performing an analysis, GCAS returns raw data (e.g., results of potential connections between spacecraft and ground stations) that can easily be plotted in a chart or graph. While charts and graphs provide researchers and engineers the data of interest, they do not help the user visualize and understand the orbits and links in a dynamic way. The three-dimensional (3D) GCAS visualization software program can take the GCAS raw data, generated from MATLAB® (The MathWorks, Inc.), and turn it into an accurate, interactive, and modern visualization. This allows users to generate simulation data and view the analysis results at the same time, providing an exact visual representation of the data. The output of the visualization software is extremely useful for demonstrations during a presentation or event to help the audience understand the data.

The 3D visualization software was developed using open-source software elements. Open-source software provides numerous advantages (e.g., low or no cost, accessibility, etc.) to the developer and user. In this case, the only software cost involved is the MATLAB® license, as JavaScript and all associated open-source libraries are free of charge. The software runs using Google Chrome™ (Google, Inc.), meaning it is highly accessible, cross-platform, and does not require external programs to run outside of MATLAB®. Developing 3D visualization simulation software in this manner provides NASA the ability to expand the visualization functionality when a new aspect of an analysis becomes available.

Visualization Software Overview

The GCAS 3D software currently supports the visualization of three modes of analyses: Orbit Propagation, Line-of-Sight (LOS), and Dynamic Link Margin (DLM). The visualization of a satellite's orbit is compatible with eight planets, two dwarf planets, and 21 moons. The software shows realistic lighting and planetary body surface textures and it has custom 3D models that are moved around the scene to represent ground stations, satellites, and other object types. The software draws out the paths of satellite orbits, which are represented as grey lines. Figure 1 visualizes a configuration with the Moon in the foreground with one lunar south pole ground station and three communication satellites, while having the Earth in the background with the three Deep Space Network ground stations illustrated.

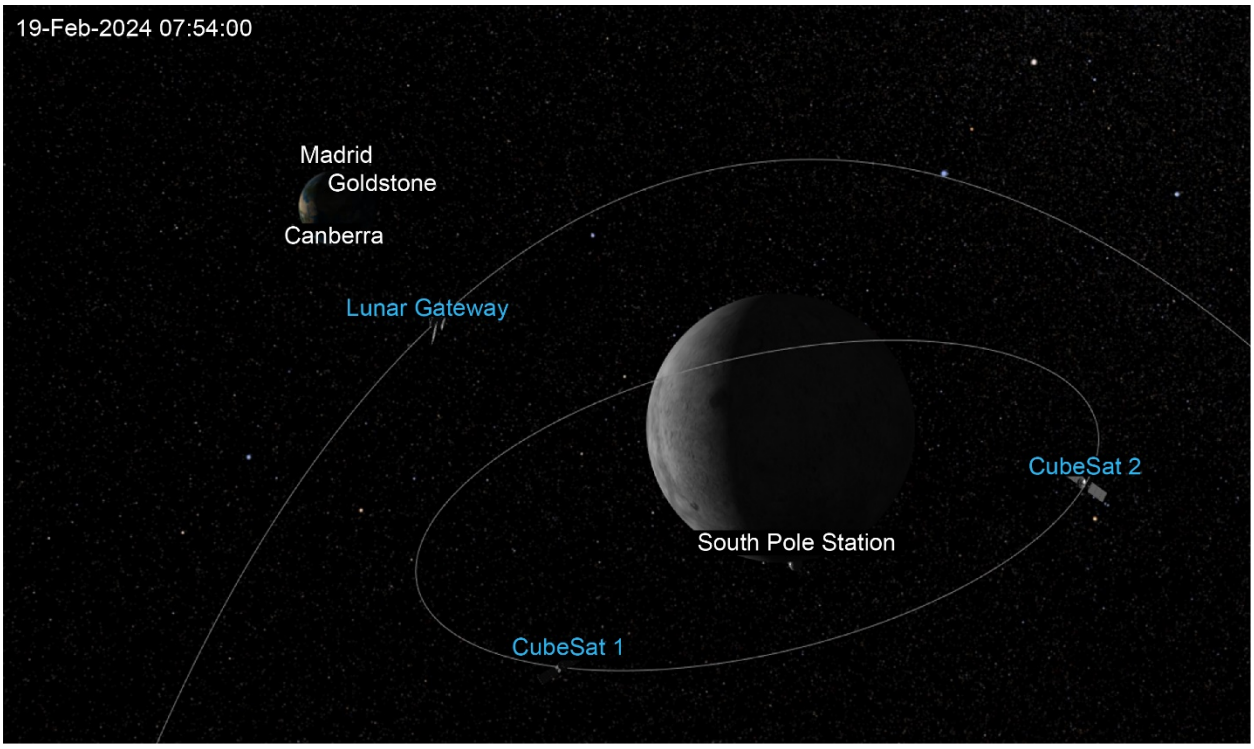

Figure 1.—Communications system visualization.

For LOS and DLM modes, the software creates colored lines that can turn on or off to illustrate satellite connections to other satellites or to ground stations visualizing the presence and/or performance of different link types. In DLM mode, the color of the line dynamically changes to match the communications link margin of the link of interest. There are also many easy to use graphical user interface options that allow the user to customize elements in the scene such as line color, turn labels on or off, switching what body the camera is centered on, changing the size of the 3D models, changing the width of the lines, autorotating the camera, and much more.

Software Development

Three.js (Ref. 1), a library for the Web Graphics Library (WebGL) graphics engine, serves as the graphics library for this visualization software and is coded using JavaScript. Three.js is easy to use, has powerful features, and has very strong documentation. As a web graphics library, the WebGL software runs from an internet browser. Using a browser has many advantages, such as compatibility between different operating systems, the ability to make changes without compiling, the option to later integrate it into other websites or programs, and the ability for the software to run without installing a program or requiring anything other than Google Chrome™. A security feature in Google Chrome™ prevents images and textures from loading properly, and so a local server hosting program called the cross-platform, Apache (The Apache Software Foundation), MariaDB (MariaDB Foundation), PHP (The PHP Group), and Perl (Perl.org), or XAMPP, was used during development. For general usage, the software uses a Python™-based web framework (Python Software Foundation) called Django® (Django Software Foundation) (Ref. 2) for the local server hosting program functionality. Django® executes without installation, which allows for greater portability and ease of use.

The next development consideration was moving the large amounts of planet, orbit, position, and link data from MATLAB® to JavaScript. It was important to make it automatically readable in JavaScript while retaining the format of the MATLAB® data, which contains large amounts of arrays and structures. The solution was to convert the data to a JavaScript Object Notation (JSON) file format. A JSON file is an open-standard file format that stores human-readable data and preserves all data types. The JSON files are most commonly used on the web, and JavaScript has great support for the file type. MATLAB® has a function natively built-in called jsonencode (Ref. 3), which will turn any MATLAB® datatype into a standard JSON formatted file. One issue discovered was that some of the MATLAB® data contained NaN (i.e., not a number) and infinity numbers, which are not valid characters in JSON files, causing the JavaScript JSON loader to fail. Since the data containing these invalid characters was not needed in the visualization, a script was written to remove the invalid characters from the MATLAB® data before encoding to JSON format. The encode times, file size, and JavaScript load times were cut in half due to the removal of the invalid characters and unnecessary data from MATLAB®.

User Controls of Visualization Scene

The Orbit Propagation mode of the simulation software, which is capable of running large quantities of data (including days of continuous orbit), and is a means for the user to control the flow, was important. The bottom of the screen features a large range slider and button panel. Each “step” on the slider represents one time step in the data, as shown in Figure 2. The program automatically advances at a user-controllable rate through the data, and the range slider indicates that movement, similar to how an online video’s progress bar operates. The range slider is semitransparent by default, allowing the tools to be less obtrusive when not in use. However, when the user hovers their cursor over the range slider, it brightens and becomes fully opaque. By dragging the slider either to the left or right, the scene travels to the corresponding point in time.

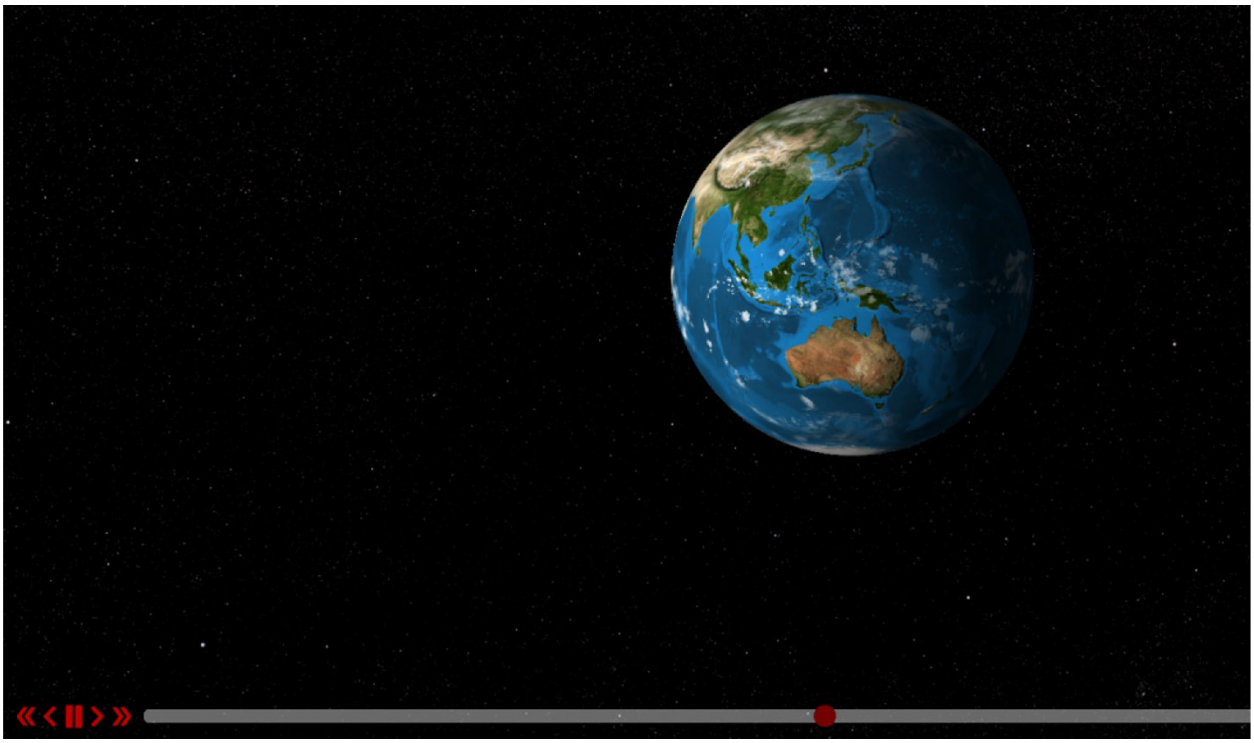

Figure 2.—Button panel and slider bar.

The primary button panel resides to the left of the range slider bar, shown in Figure 2, and includes buttons for manipulating the scene. In addition to the functionality for playing and pausing the simulation, additional buttons allow the user to either advance one time step forward or backward, restart the animation from the beginning, or skip to the end entirely. These buttons become fully opaque and increase in size when the cursor hovers over them to improve visibility. To make these basic functions even more accessible, keyboard shortcuts were implemented, using JavaScript event key codes to code certain events (in this case, initiate the button's functionality). For example, pressing the spacebar yields the same result as pressing the actual play or pause button on the screen. Likewise, the left and right arrow keys, respectively, retreat and advance the time step by one step. Holding down either shift or control and pressing the left or right arrow key will either restart the simulation or skip to the finish, respectively. Each function of the primary button panel is represented by these accessible keyboard shortcuts, providing the user with the flexibility to move throughout the scenario without a mouse.

Screen Capture and Video Recording

It is often valuable to save a particularly noteworthy segment of a simulation. The visualization software offers two such ways to do so: capturing a screenshot image or recording a video. As seen in Figure 3, the user accesses the buttons for these features on the bottom right of their screen, directly to the right of the range slider.

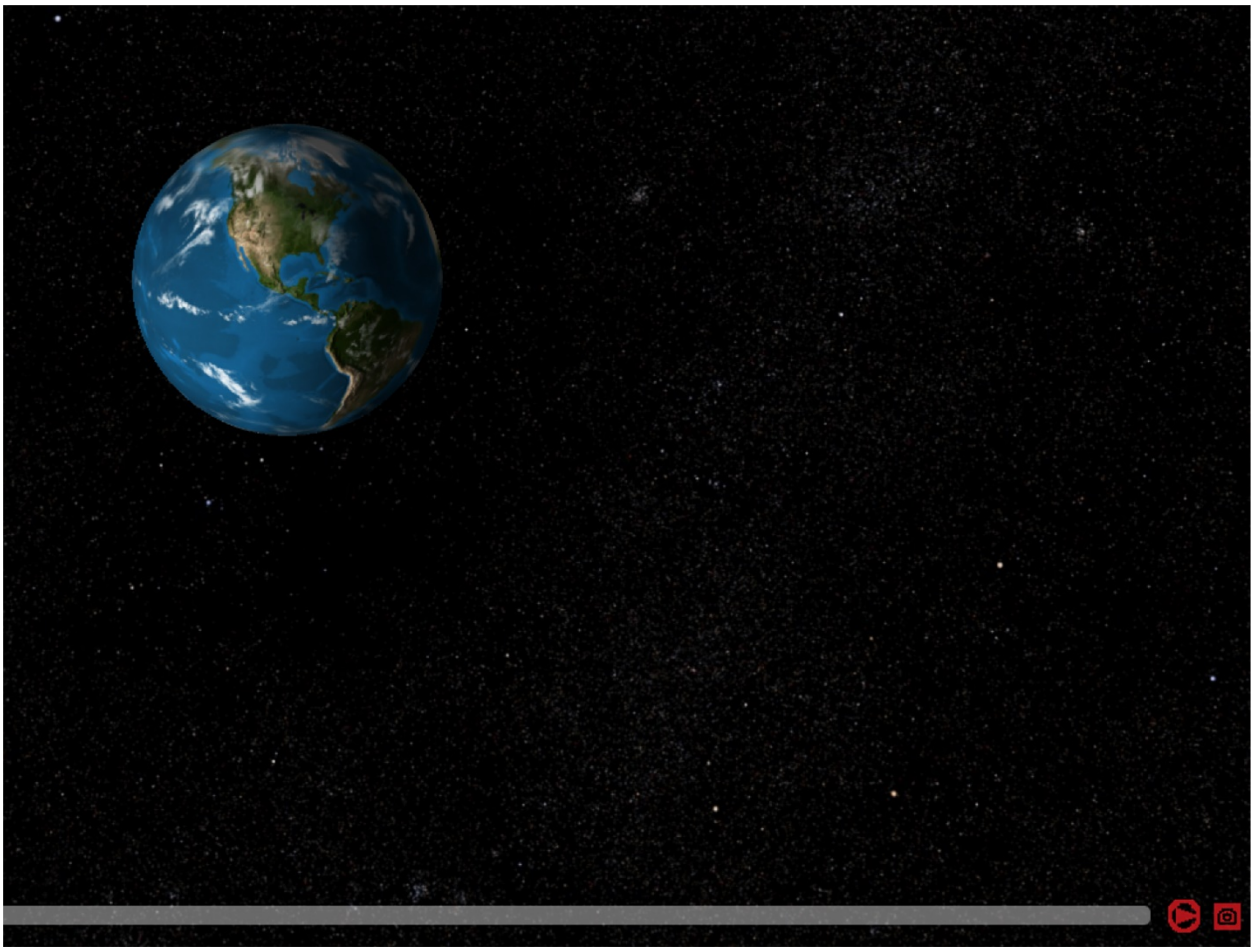

Figure 3.—Record (left) and screen capture (right) buttons.

The screen capture capability builds the screenshot based on the information available on the page (Ref. 4) using a JavaScript library named html2canvas. While there exists plenty of screen capture software, many are unable to capture the HyperText Markup Language (HTML) elements, which in this application contain vital information such as the simulation time and satellite and station labels. By calling the html2canvas library and saving the created object as a blob or a file-like object of immutable, raw data that is not necessarily in a JavaScript-native format, the screenshot can be locally saved to the system for future use (Ref. 5). Unwanted segments of the visualization, such as buttons and sliders, were explicitly hidden from the screenshot.

The ability to record video registers the audio and video as well as screen activity recording (Ref. 6) using a JavaScript library called RecordRTC. Other JavaScript libraries, such as CCapture, only record the canvas (the part of the screen that contains the simulation), but excludes text-based items such as the timestamp and labels. At the time of writing, Google Chrome™, and most other browsers, do not support the necessary RecordRTC features that allow the program to automatically start recording upon clicking the button; instead, the user must select exactly what they want to record from a list of screens, applications, and tabs, which may not be immediately obvious to the typical computer user. To appropriately guide the user to the correct selection, a modal window gives the user precise and easily understood instructions, as demonstrated in Figure 4.

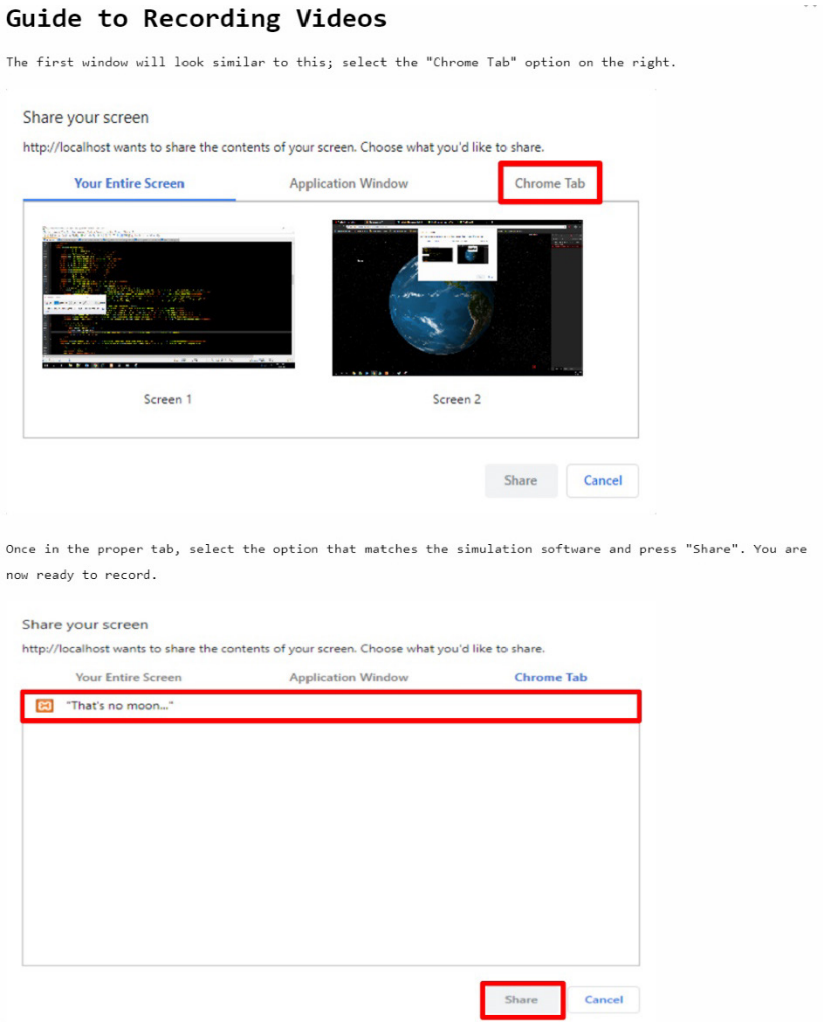

Figure 4.—Video recording modal window.

Understanding and Working With Glenn Research Center Communication Analysis Suite Data

Creating a visualization of planets with moons orbiting around them, ground stations, satellite orbits, LOS links, and other objects, can be done different ways, dependent on the data provided. The JSON file has five structures of data: inputData, nodeParams, nodeConfig, inputConfig, and inputDates. The inputData structure contains the positions of planets, satellites, lighting, moons, and LOS visibility as defined in the configuration. Whereas, nodeParams contains a copy of the parameters, such as ground station positions, satellite orbital parameters, antennas, the planetary body that the satellite is orbiting, and other parameters. Figure 5 illustrates the nodeConfig data structure and contains an array of the configurations for LOS, including the transmit node, antenna, and field of view (FOV), and the receive node, antenna, and FOV. In the example, the number 3 is referring to node 3, which is the third satellite, ground station, or object index that was defined in the MATLAB® configuration. The same is true for the receive number. The other numbers refer to the antenna number and FOV index parameters that were defined in the MATLAB® configuration. The LOS configurations can be node A to node B to node C or even longer multinode configurations, where each extra node would create another six numbers in the array. The structure inputDates contains start and end dates of the data and the time step representing the number of seconds between data points. Whereas, inputConfig contains other parameters that are not currently used in the visualization software.

There are three types of structural elements in the inputData structure: TIMEDATA, LOS configurations, and planetary and node position data. The TIMEDATA element lists the date and time for each time step. This data is used by the onscreen label (illustrated in Figure 1) to display the time and date of the data in the visualization. Next, for each array defined in nodeConfig, LOS configuration provides a corresponding dataset that contains multiple arrays and structures of data about when each individual segment has LOS visibility (if it is a multimode configuration) and when the entire configuration has visibility. This visualization software only uses the intersected visibility across each configuration. Finally, for the planetary and node position data, each planetary body defined in the initial MATLAB® configuration has a lower-level structure defining each body and node's fixed and inertial positions with respect to the parent body's location. The structure contains the positions of all the other planetary bodies, satellites, ground stations, and the Sun from the perspective of that structure's body. For example, if the analysis scenario has satellites around Earth, the Moon, and Mars, there will be a structure for each satellite and each planetary body. The Moon structure will have the Moon at [0, 0, 0] and the position data of the planetary bodies and objects at each time step in Cartesian coordinates in both the Inertial Frame and Fixed Frame coordinate systems. This allows the software to plot the positions of all the bodies and objects from the perspective of each planetary body. When the program animates, the positions of the

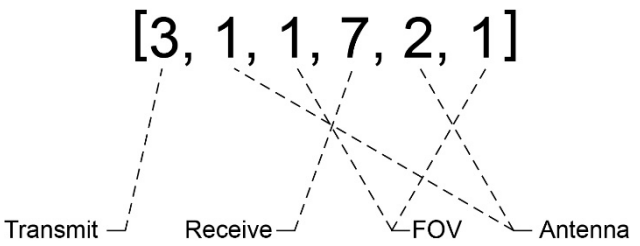

Figure 5.—Transmit node A to receive node B Line-of-Sight configuration. Field of view (FOV).

objects come from one of those data structures. The array index (each time step) of each object position array is determined by the user interface. When the time step changes, the program looks at the position data of each object at that array index and updates the position of each object in the scene. The user has the ability to change the planet camera focus to any of the other planetary bodies. When it is changed, the software looks at the new body's corresponding structure within the inputData structure to obtain its positions. This allows the user to view the scene from the perspective of any planetary body.

Creating and Managing Graphical Scene

Creating an accurate and realistic scene of satellites, ground stations, sunlight, planets, moons, and orbits, which also provides an easy-to-use and helpful experience for the user, requires many different elements and techniques. To show the visualization properly, the program determines which planets and moons the user wants to include, what planet the selected moons orbit, what satellites orbit which planetary body, the ground station locations, and much more. In three.js, each planet, satellite, orbit, and LOS link has a corresponding software object created. Each object has different parameters and properties stored in a way to easily reference while the program renders the scene. Three.js uses meshes and camera, and lighting objects to create and render the scene, as illustrated in Figure 6. A JavaScript body object keeps track of and stores all the three.js scene objects created. For each planetary body in the scene, a JavaScript body object gets created. Each body object contains all the satellites, ground stations, orbit paths, layer information, body data, and moons associated with that planetary body. The moon property of the object is an array of body objects, one for each moon currently orbiting the main body. This object structure is key to creating a dynamic scene to use with any MATLAB® configuration.

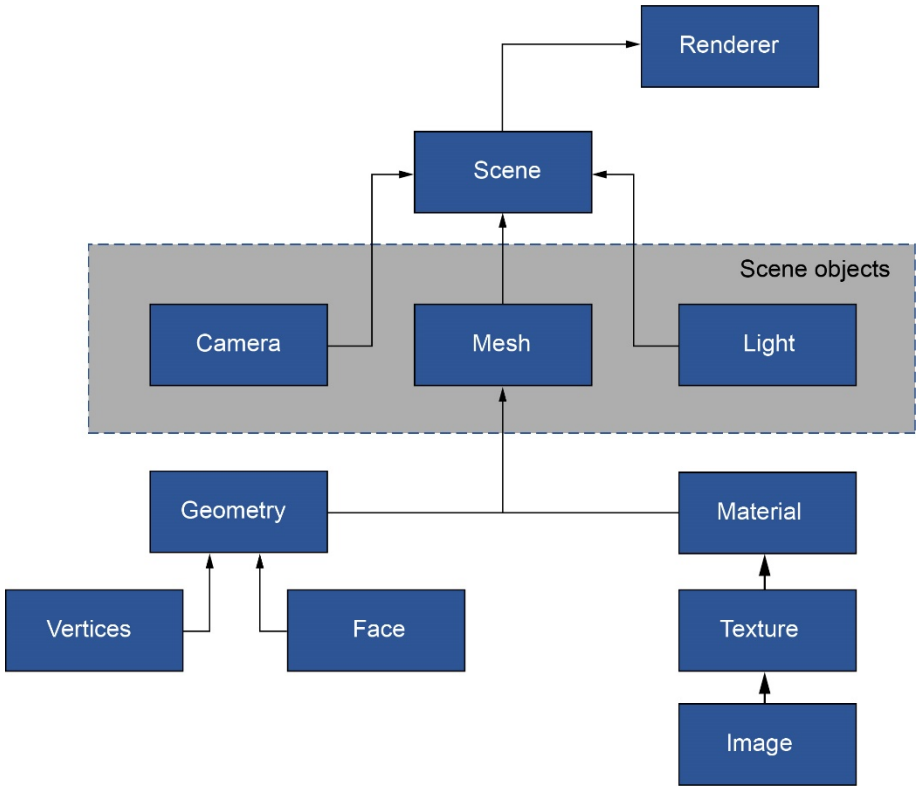

Figure 6.—Three.js graphics layout.

The visualization scene contains two light source objects. The first comes from the Sun, which provides the appearance of sunlight and changes position using a field in the inputData for each planet. The second comes from ambient light, which adds a very small amount of light to the whole visualization scene to make the dark side of the planets look more observable when not illuminated by the appearance of sunlight. The Sun object is an image of the Sun inserted into the scene and moves as time progresses. In addition, the visualization software illustrates planetary body orbits. When enabled, lines in the scene show the path a planet or moon follows in relation to the body of focus in the visualization scene.

The elements used in the visualization scene, as illustrated in Figure 7, are made up of two main groups—the body objects, which are linked to a planetary body and the static scene objects that are not tied to a planetary body. Some of the static scene objects are sunlight, the Sun object, ambient lighting that includes a skybox, which is a high-resolution star background obtained from NASA Goddard Space Flight Center Scientific Visualization Studio (Ref. 7) and illustrated previously in Figure 1. To create the labels (seen in Figure 1) a special "css2drenderer" three.js library converts a Cascading Style Sheet (CSS) label to an object in the scene that can be moved within the scene. This library requires a second three.js renderer along with the normal renderer, to draw the labels in the scene.

A large part of the visualization software development was optimizing the load and run times. When the program is started, three.js creates all six properties (satellites, ground stations, LOS links, satellite orbits, labels, and planetary body) of each object and assigns them to a layer (Figure 7). There is a layer created for each planetary body in the scene (for each item in the body array). Each layer consists of a main body and, if used, any secondary bodies (i.e., if Earth is the main body then the Moon is the secondary body, but only if there is a satellite or ground station set around or on the Moon, respectively). Both the main body and any secondary bodies consist of the six properties. Before the program finishes the derivation of all layers upon program startup, all layers except one are hidden, only showing the objects for that specific planetary body. When the user switches the planetary body focus, the current layer is hidden and the new layer becomes visible. This allows instantaneous switching between planets without removing, computing, and creating those objects again, which would use more computer resources and increase load times.

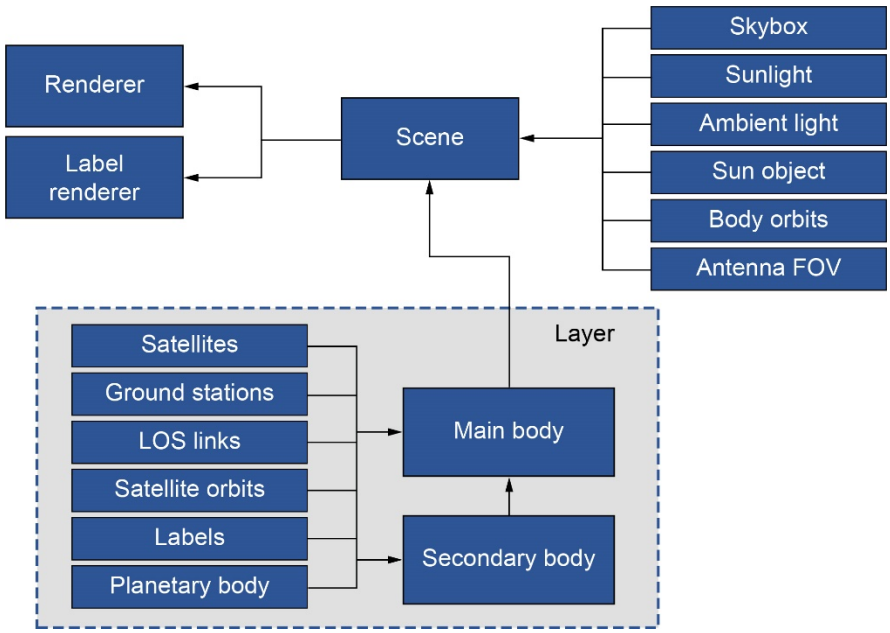

Figure 7.—Visualization objects. Field of view (FOV). Line of sight (LOS).

The textures of the planetary bodies are all smoothly wrapped around the shape of the planetary body. The planetary bodies are defined as ellipsoids, based on the equatorial and polar radii of the planetary body, obtained from data within the GCAS software. Unlike the other planetary bodies, Earth uses four different textures: a map, normal map, specular map, and cloud layer. The map contains the texture of the Earth's surface. The normal map (also known as bump map), adds height to the map in areas of the Earth that are higher than others (i.e., mountains). The specular map differentiates the land from the water, therefore allowing the rendering to add extra light (shininess) to the water. Finally, the cloud layer adds cloud images that move independently of the Earth's surface. All these maps make the Earth more realistic to observe.

Concluding Remarks

The three-dimensional (3D) visualization software described in this report aids researchers and engineers to analyze space communications scenarios to develop and plan current and future missions. It helps users better understand the data output from the Glenn Research Center Communication Analysis Suite (GCAS) capabilities. The portability of the visualization software to multiple users, as well as the ease of use will help to increase the user base of the GCAS software. The visualization software uses modern JavaScript and three.js programming languages and libraries to illustrate communications analysis scenarios to users in Google Chrome™ (Google, Inc.). Additional features to add to the visualization software include the following: antenna field of view, objects orbiting the Sun, different object 3D models, more graphical user interface options, smoother transitions between time steps, support for additional communications analysis functions, and much more. The software platform provides an environment to build upon and can be easily expanded in the future.

References

1. three.js: Manual. three.js, 2010. https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene Accessed April 20, 2020.
2. Django: Python Web Framework. Django Software Foundation, 2019. https://www.djangoproject.com/ Accessed April 20, 2020.
3. jsonencode: Create JSON-Formatted Text From Structured MATLAB Data. MathWorks, 2019. https://www.mathworks.com/help/matlab/ref/jsonencode.html Accessed April 20, 2020.
4. von Hertzen, Niklas: html2canvas. MIT License. http://html2canvas.hertzen.com/ Accessed April 20, 2020.
5. Mozilla: Blob. MDN Web Docs, 2020. https://developer.mozilla.org/en-US/docs/Web/API/Blob Accessed April 20, 2020.
6. Khan, Muaz: RecordRTC: Class. WebRTC JavaScript Library. https://recordrtc.org/RecordRTC.html Accessed April 20, 2020.
7. Bridgman, Tom: The Tycho Catalog Skymap—Version 2.0. NASA Goddard Space Flight Center Scientific Visualization Studio, 2009. https://svs.gsfc.nasa.gov/3572 Accessed April 20, 2020.