NASA/TM-20240011691

Creating Camera Controls for First-Person Camera in VulkanSceneGraph

Kristie O'Brien and Bryan W. Welch
Glenn Research Center, Cleveland, Ohio

December 2024

NASA STI Program Report Series

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 NTRS Registered and its public interface, the NASA Technical Reports Server, 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 counterpart 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 cosponsored 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.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

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

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

NASA/TM-20240011691

Creating Camera Controls for First-Person Camera in VulkanSceneGraph

Kristie O'Brien and Bryan W. Welch
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

December 2024

Acknowledgments

I would like to thank those who came alongside to stand by me in my work and the writing of this report. I appreciate Vaughn Richards for his insights into the mathematics behind first-person cameras, as well as his willingness to answer questions. Thanks also go to Joseph Symons and Ana Granite for their advice on integrating the camera functionality with the rest of the project. But my greatest appreciation goes to my mentor and this report's coauthor, Dr. Bryan Welch, without whom this report would not have been possible. Thank you for your guidance, support, and for answering my many questions. To everyone mentioned here, Ad Astra and keep up the good work.

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

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

NASA STI Program/Mail Stop 050
NASA Langley Research Center
Hampton, VA 23681-2199

Creating Camera Controls for First-Person Camera in VulkanSceneGraph

Kristie O'Brien* and Bryan W. Welch
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Summary

The way users interact with a virtual, three-dimensional (3D) scene is heavily influenced by the way the camera used to view the scene is controlled. A first-person camera is a common form of camera control for computer programs. Its role is to create an immersive viewer experience, which allows users to traverse a scene as they might in the real world. Allowing for the implementation of the first-person camera makes for a more holistic, well-rounded way to interact with the visualization program. Accomplishing this involves mathematical calculations that define how the camera should be moved for the computer system. These movements equate to the rotation, translation, and scale of the changes. It should be noted that a computer does not inherently process what movement directions (left, right, up, or down) mean. The mathematical equations used define these principles in a way the computer can process. Further aspects to consider are detecting when the camera has moved and how far. This is most frequently accomplished through user |input through external devices. These devices, for the purpose of this report, include mouse input and keyboard input. Additionally, the in-development program this report is referencing works with the VulkanSceneGraph (VSG) library to create the scene and build the base of the camera controls. Although VSG is a powerful library with many capabilities, additional Application Programming Interfaces (APIs) might be needed during development to produce the desired results, as is the case in this program. Through combining proper mathematical calculations, utilizing additional APIs, and implementing the existing VSG library capabilities, implementing first-person camera controls is possible in a 3D scene.

Nomenclature

3D	three-dimensional
API	Application Programming Interface
GCAS	Glenn Research Center Communications Analysis Suite
VSG	VulcanSceneGraph
WinAPI	Windows API

Introduction

In the scope of the NASA Glenn Research Center Communications Analysis Suite (GCAS), users can primarily interact with the three-dimensional (3D) rendering of our solar system in two key ways (Ref. 1). The first is with an object-relative camera. The second is with a first-person camera, which will be the primary focus of this report. The first-person camera allows for a high level of freedom of movement, designed to mimic a person observing a scene from multiple perspectives (Ref. 2). For example, on the surface of a node object such as a planetary or lunar object, the camera would allow users to observe and move about on the

*NASA OSTEM Spring 2024 Intern, Southern New Hampshire University.

surface of that node as they might in the real world on an actual planetary object. GCAS allows for several planetary objects and their respective moons to be modeled and observed through these cameras (Ref. 1). As of this writing, the version of GCAS currently in development utilizes the VulkanScreenGraphic (VSG) library to render its scenes. This modern, cross platform, high-performance scene graph library is built upon the Vulkan graphics/compute API. The software is written in C++17. The source code is published under the MIT License. The VSG library has some built-in camera object functionality and a base for camera controls that can be modified to fit certain aspects of a complex camera system. However, custom functionality and more extensive modifications are required to implement a first-person camera.

Foundational Data

Before diving further into the particulars of camera development for GCAS, some key aspects of how a computer infers a camera must be addressed. It should be noted that a computer program does not instinctively know directions, orientation, position information, or what it means to change these aspects of looking around a scene (Ref. 3). These must be defined within the program itself. Despite programming languages and libraries having their own unique and particular ways of defining camera structures, the use of the LookAt function's input argument vector elements is one commonly known way. Such elements can be defined differently between programs and libraries, but most are comprised of the following base components:

- A vector of the camera's position in the world
- A vector of the camera's target
- A vector of the camera's upward direction

Figure 1 offers an example of these components. The camera's position in the world is just what its name implies: it is the position in the world where the camera is currently stationed (Ref. 3). This can be thought of much like a person's coordinates on a map. The vector of the camera's target is typically the direction in which the camera is looking (Ref. 3). For example, imagine a person sitting stationary at a desk, first looking straight ahead at a wall. Then, the person moves their head and is now looking up at the ceiling. The person is still in the chair, so their position has not changed, but what they are looking at has. This example allows visualizing the basic difference between position and target. Finally, there is the vector of the camera's upward direction. This is basically the vector that points upward from the camera, defining its orientation relative to other objects (Ref. 3). These elements can then be combined to define the basic directional, placement, and observation data for the camera These elements will also be the foundation of working with changes in these camera variables that are caused by movement.

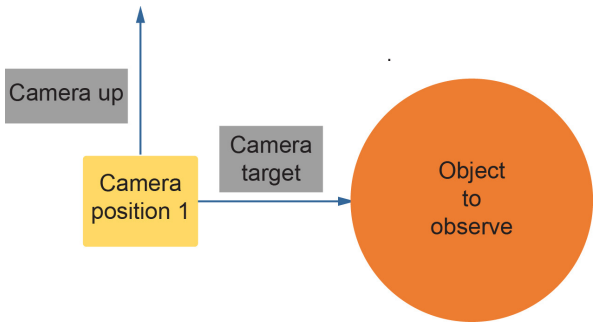

Figure 1.—Basics of three component vectors for working with camera direction and position information.

WinAPI and the Importance of Mouse Cursor Position

Now, with the three basic vectors in mind, it is important to understand they can be transformed to show changes in the camera's position and directions. As in the example of a person sitting at a desk and turning their head to look at something new, a camera on a computer can figuratively turn its metaphorical head to look at other objects or areas of a scene. However, unlike a person, the computer's camera has no built-in frame of reference to instruct it how far to turn. This must also be defined. One way of doing this is by measuring how much something related to the movement event moves with each movement iteration. In the case of the GCAS system, mouse controls are used to capture movement event information for the camera (Ref. 2). In theory, the camera should be able to look wherever the mouse is pointing. There are a few ways this can be accomplished. One way is simply to capture the mouse's location and have the camera's target vector follow it. However, this could produce issues when the mouse goes too close to the edge of the screen, thus restricting movement. For more natural movement, allowing for more freedom of movement, another method is used that involves placing the mouse cursor at the center of the screen, moving the mouse in the desired direction, then, after the move event is concluded, placing the mouse once more in the center of the screen. This allows for the difference between the mouse's new location compared to the center to be gathered each time the mouse moves. Although some VSG methods allow capturing mouse input, a more streamlined approach that allows for ease of implementation with a client window can be found with the Windows API (WinAPI). The WinAPI allows mouse input to be captured along with mouse position information. From there, the WinAPI has built-in commands to find the center of the client screen and place the mouse position at that center. The difference found between mouse locations can then be used to justify how much the mouse has moved. This allows the computer to have a working frame of reference for how much the camera should change its target vector.

The Math Behind the Method

The three basic vectors from LookAt might make up the foundation of the math that must be done, but three basic vectors do not make the math itself. These vectors can be used in a variety of ways to produce varied outcomes in terms of how a camera moves about a 3D environment. However, alone, they are only the building blocks of the math. The mathematical equations will use these vectors to understand and apply the appropriate changes. First and foremost, it must be fully understood that the first-person camera in the scope of GCAS must function like a person walking around a scene. For the purpose of this report, consider the situation where a person might be walking around the surface of a planetary object. This object might not be a perfectly round sphere, but it will still be spherical in nature under the scope of a 3D shape. With this in mind, it is important to understand how the camera is expected to behave. The camera must appear to be standing vertically relative to the planet in such a way that it mimics a human's view on the surface of a planet, regardless of where they are on the surface (Ref. 3). As such, the camera's orientation must be relative to the object on which it is standing (Ref. 3). This orientation is controlled by the up vector discussed in previous sections. Figure 2 offers an example of what this looks like; note that the up vector is always positioned perpendicular to the planetary object's position. This ensures that the camera is always oriented such that it is sitting on the surface of the planetary object.

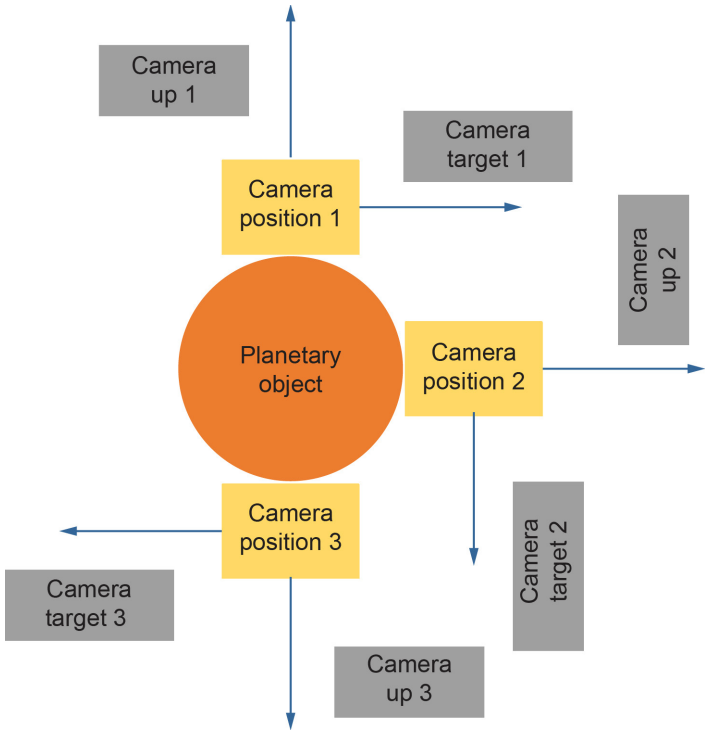

Figure 2.—Camera positions with their corresponding target and up vectors.

Mathematically speaking, the camera's up vector will be the perpendicular vector of the camera's position as it is relative to the surface of the planetary object, like the up vector in a local topocentric coordinate frame. But from here, it must be understood what must be done to the other aspects of the other two elements. Both the camera's position vector and what the camera is looking at might change. To account for changes in the position vector, the user will use keyboard inputs to control the position of the camera at a given time. Distinct keys are bound to change the reference position of the camera forward, backward, left, or right (Ref. 3). The target vector changes are controlled by the mouse as described previously.

Because a computer system does not have a frame of reference for movement, such movement must be defined. In order to move forward, backward, left, and right, the computer needs definitions for the position, the direction of the movement, and the speed of the movement. Without a context for speed, the camera could move very fast or very slow in disproportionate movements. Therefore, having a speed variable assisting with updating the position is essential. Putting all of these together, it is possible to obtain the following mathematical equations.

Necessary Variables

position = camera position coordinates
direction = direction component of a camera
speed = constant value for speed of the camera's movement.

Equations for Forward, Backward, Left, and Right

If the camera is moving forward, use this equation:

$$position = position + (direction * speed)$$

If the camera is moving backward, use this equation:

$$position = position - (direction * speed)$$

If the camera is moving left, use this equation:

$$position = position + (direction * speed)$$

If the camera is moving right, use this equation:

$$position = position - (direction * speed)$$

The scenario described handles the base movement, but that leaves changes in viewing vectors from a common reference position unaddressed. Rotation is the key here as, just like when someone turns their head, the movement is not always simply forward and backward or side to side. Remember: the amount of rotation is determined by how far the mouse moved from the center of the screen during a movement event. The resulting difference can then be broken up into *x* and *y* coordinated for their corresponding differences. These differences are then used to calculate the proper angles for rotation.

Necessary Variables

position = camera position coordinates
target = the direction in which the camera is looking
Δx = *x* variable difference from mouse position to screen
Δy = *y* variable difference from mouse position to screen
up = camera up vector
target = camera target vector
Center = center of the client screen
xAngle = angle of rotation for *x* in radians
yAngle = angle of rotation for *y* in radians
xRotation = rotation matrix for *x*
yRotation = rotation matrix for *y*
right vector = right vector related to the camera

Steps for Rotation

To create *x* rotation, rotate by xAngle around the camera up vector
To create *y* rotation, rotate by yAngle around the camera right vector

Equations for Rotation

Rotate first with respect to x:

$$target = (target - position) \cdot (xRotation + position)$$

Calculate the right vector related to the camera target:

$$\text{right vector} = \|target \times up\|$$

Rotate next with respect to *y*:

$$target = (target - position) \cdot (yRotation + position)$$

Results and Discussion

The results in the implementation of the rotations as described allow for smooth, complete rotations and the ability to move about a scene. The type of rotation chosen works with two different angles around two different axes for rotation to avoid what's known as gimbal lock. Gimbal lock might occur in various locations when moving about a spherical or sphere-like object due to the nature of the vectors and their normalized components. As a result, gimbal lock would result in incomplete or locked movements at certain locations. For example, gimbal locking might occur with more regularity at polar regions on a planetary object. This is like how in real-world navigation, there can be no specific assignment of longitude for latitude at the true North or South Pole. This is because the latitude values would be either positive or negative 90° at the different poles. In the real world, this information plays a role in navigation. In terms of computer programming in this instance, this information informs instances of gimbal locking in cameras. This is not ideal for the type of camera created here. Given the issue of spherical angles at the true poles (either north or south), where longitude is undefined, an alternate process was needed to support camera control. The rotation calculations work to control input from the mouse, allowing the camera to turn much as a person might turn their head. Meanwhile, the key controls allow for the camera's position to be updated in the scene, much as one might walk around a room. The results allow for a much more holistic viewing experience of the GCAS scene. This in turn allows for more detailed site analysis within GCAS as the user has the option to move freely within the world space. This type of rotation while looking on the horizon of an object in space can be seen in Figure 3 to Figure 5. Note that these images were taken on the surface of the Moon within GCAS. Figure 3 shows a horizon view with the camera turned toward the left side of the screen. Figure 4 shows the same kind of horizon view, but the camera has been turned slightly to the right, allowing the view to be focused more toward the center of the screen. The final figure, Figure 5, shows the camera view now rotated toward the right side of the screen. As such, it can be seen how the view operates and works within the actual system without gimbal lock while allowing for a horizon view on a nearly spherical surface.

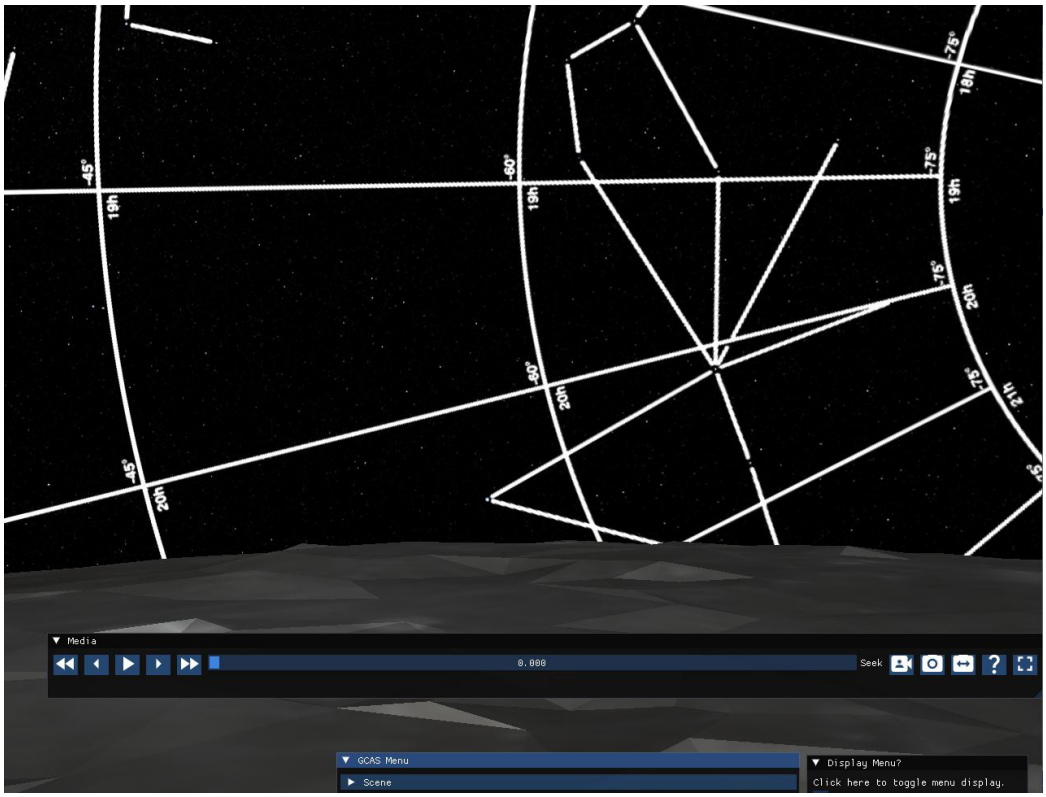

Figure 3.—Horizon view on lunar surface looking toward left of screen.

Figure 4.—Horizon view on lunar surface looking toward center of screen.

Figure 5.—Horizon view on lunar surface looking toward right of screen.

Concluding Remarks

This report summarizes software development efforts to enable camera control in a new Glenn Research Center Communications Analysis Suite (GCAS) three-dimensional (3D) visualization environment. Camera control processes were determined to enable first-person views from a fixed location, along with the ability to rotate about that location and to move away from that location. The efforts were about trying to use existing processes within the VulkanSceneGraph capability to replicate some of the existing first-person camera view/control capabilities within the browser-based GCAS 3D visualization software. It is hoped that these efforts are able to influence future development in support of viewing the 3D scene about the location of orbital nodes or vehicle-based nodes.

References

1. Shalkhauser, Lucas D.; Henderson, Eric; and Welch, Bryan W.: On Development of Three-Dimensional Visualization Capabilities in Glenn Research Center Communication Analysis Suite. NASA/TM-20205000040, 2020. https://ntrs.nasa.gov
2. Richard, Vaughn M.; and Welch, Bryan W.: Improvements to GCAS Visualization Functionality. NASA/TM-20230018222, 2024. https://ntrs.nasa.gov
3. de Vries, Joey.: Camera. Learn Open GL, 2014. https://learnopengl.com/Getting-started/Camera Accessed July 31, 2024.