AAS 19-449

VISUAL INTERACTIVE TRAJECTORY DESIGN

Ravishankar Mathur*

A method of altering trajectories via visual "grab-and-drag" gestures with immediate graphical response, called Visual Interactive Trajectory Design (VITD), is presented. The various components of VITD are defined (e.g. draggable objects), and how they affect the trajectory is discussed. An implementation of VITD is demonstrated that uses the General Mission Analysis Tool (GMAT) to propagate trajectories, and visualizes them in realtime using the OpenFrames 3D visualization API. Examples of using VITD are shown, including finding a lunar free-return trajectory by visually rotating the trajectory's initial velocity vector. The benefits of VITD are explored, including for teaching, exploring the design space, and generating initial-guess trajectories for optimization.

INTRODUCTION

As a result of advances in astrodynamics applications over the past two decades, high-fidelity visualizations are no longer considered a novelty or optional component of mission design and analysis. Widely-used commercial and government-developed tools such as Systems Tool Kit (STK)[1], Copernicus[2], and the General Mission Analysis Tool (GMAT)[3] all contain 3D visualizations that provide critical information far above and beyond what text-based data or static plots can achieve. These visualizations have become an essential part of effective trajectory design and optimization, so much so that they are used to affect design decisions and not merely for the sake of output pictures and presentations. For example, design for the NASA OSIRIS-REx mission's Touch-and-Go (TAG) trajectory was performed in part with STK[4], and 3D visualizations played an important role in validation of that trajectory.

These uses of 3D visualizations fall under the general field of *visual analytics,* which addresses how to effectively incorporate visual feedback into analysis and design processes. While showing the results of simulations is a necessary first step in visual analytics, it is also important to directly incorporate these visualizations back into the design process using visual interaction with onscreen data and objects. For example, Copernicus enables the use of graphical user interface (GUI) widgets to directly affect mission parameters and displays updated trajectories in real time. As shown in Figure 1, Copernicus users can alter optimization variables such as inclination, semimajor axis, and epoch using familiar widgets such as sliders. For each incremental change to a variable, the software immediately propagates and displays updated trajectories using any supported dynamics model. This allows users to explore the design space by rapidly changing variables and seeing the results in real time, which circumvents the traditional process of manually entering new values.

* Sr. GN&C Engineer, Emergent Space Technologies Inc., Laurel MD. AIAA Member.

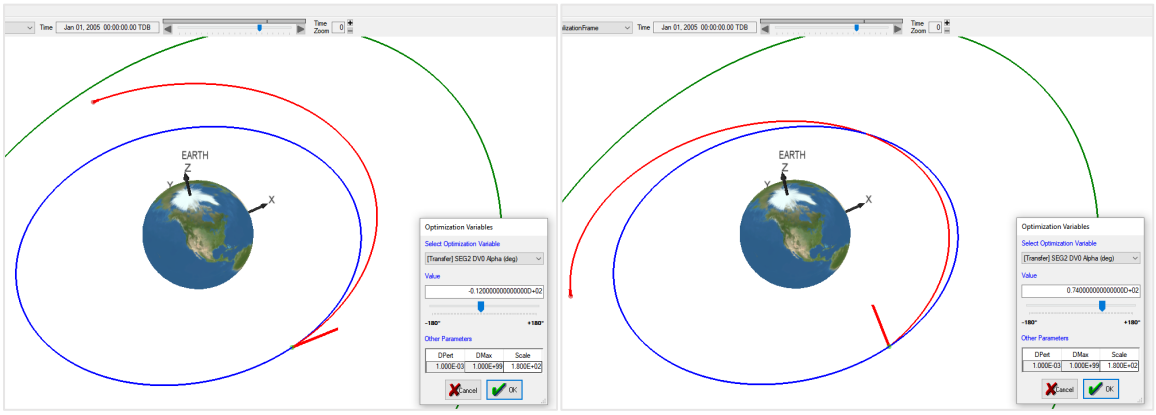

Figure 1. Interactively adjusting the right ascension of the ΔV vector in Copernicus. The trajectory changes smoothly as the value is changed from the initial (left) to final (right) values.

Furthermore, academic aerospace research has yielded interactive visualization-based approaches to extracting information about the topologies of specific dynamical systems. For example, Schlei presents a novel approach that transforms trajectory states in the circular restricted three-body problem (CRTBP) using affine transformations (geometric rotations, translations, and scales).[5,6] The user performs these transformations by using intuitive "interactive transformation editors", which include rotation and stretching operators. The result is a trajectory that does not satisfy the system dynamics, but can be used as input to a differential correction process that determines the final continuous transformed trajectories. This provides a powerful approach to designing trajectories in the CRTBP, and can also be adapted to other dynamical systems.

The interactive trajectory design capabilities enabled in Copernicus and demonstrated by Schlei are clear improvements over simply displaying 3D visualizations, and their approaches can be further enhanced by allowing the user more direct control over objects' state changes. The research and development described in this paper presents a new application of visual analytics, referred to herein as Visual Interactive Trajectory Design (VITD), that allows the user to effectively "grab and drag" spacecraft and immediately see the effect of state changes on the spacecraft's trajectory. The primary goals of this research are:

1. Define the roles and responsibilities of the simulation and visualization when implementing VITD in a trajectory design application.
2. Research and design GUI objects that can be used to easily transform spacecraft states. The transformation editors described by Schlei[6] are examples of such GUI objects.
3. Determine the feasibility of recomputing trajectories in real time during user interaction. Here, feasibility is determined as a combination of the complexity of the dynamical system and the efficiency of the trajectory propagation algorithm.
4. Demonstrate the effectiveness of VITD for both learning orbital mechanics concepts and aiding in the trajectory design process.

It is important to note here that VITD is not designed as a replacement for traditional orbital mechanics teaching methods or advanced trajectory design techniques such as initial guess generators, mathematical analyses, machine learning algorithms, etc. Rather, it is a tool that aids in these endeavors by providing a new interactive approach to changing design parameters that more fully engages the user's inherent visual processing and analysis capabilities.

VISUAL INTERACTIVE TRAJECTORY DESIGN (VITD)

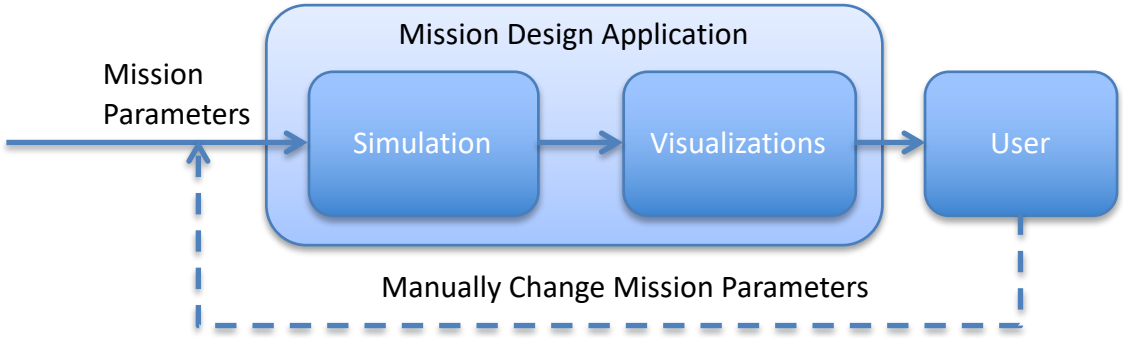

Figure 2. The standard trajectory design process, in which the user manually adjusts mission parameters in response to an analysis of the software simulation's visualizations.

The most common visual analytics workflow in current trajectory design software is represented in Figure 2, and is employed in software like GMAT, Copernicus, and STK. Here, the user first sets up the mission parameters using the software's GUI or other input mechanisms. The software then runs a simulation (which may consist of multiple parts) and produces visualizations of the simulation process and results. The user analyzes these visualizations using their orbital mechanics and trajectory design experience, and determines a new set of mission parameters based on the pertinent design goals. For example, the user might decide that the initial guess for velocity magnitude is too high, or that the step-size parameter for an optimization algorithm needs to be fine-tuned. These changes are then made manually (via the GUI or other input mechanisms) and the simulation/visualization/analysis process is re-run iteratively until the final results meet the desired design goals. The Copernicus interactive parameter-changing widgets (Figure 1) are an advanced implementation of this workflow where the GUI allows modifications to be made much more quickly than conventional text boxes, which in turn enables rapid scans of selected parameters.

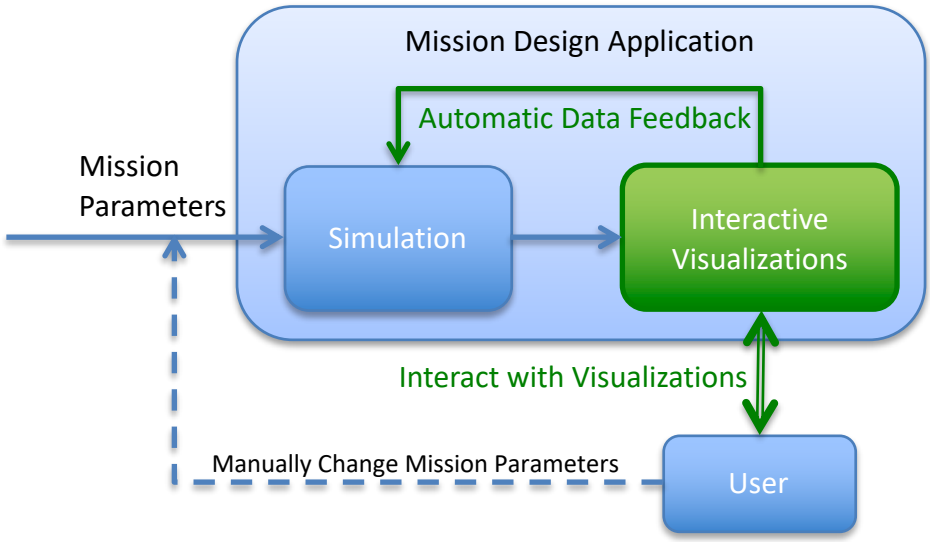

Figure 3. The VITD workflow enables direct interaction between the user and the visualizations, which communicate in a closed-loop manner with the simulation.

With a slight modification, this standard process is transformed into the VITD workflow as shown in Figure 3. The primary change here is that the visualization component becomes interactive, which means that users can directly select onscreen objects (e.g. click and drag them with a

mouse) and the visualizations communicate changes as needed with the simulation. In this manner the user no longer needs to determine new numeric values for mission parameters; instead they simply move visualized objects, and the visualization software computes the necessary changes to mission parameters. These new parameters are immediately fed back into the simulation, which computes new trajectories and displays them. Assuming the trajectory propagation is not a bottleneck, this process occurs fast enough to appear instantaneous to the user.

This definition of VITD expands on previous implementations of visual analytics by immediately computing fully-propagated trajectories while the user interacts with the control objects, without the use of intermediate discontinuous trajectories. By directly computing feasible trajectories, VITD allows users to instantly see the effects of every incremental change that they make to on-screen objects. By combining the mission designer's inherent visual processing capabilities with their experience with the dynamical system, VITD allows mission designers to develop a powerful understanding of their mission design space. Even in cases where the user is not an experienced mission designer – such as a student in the classroom – VITD provides a vast amount of information that can be used to augment the orbital mechanics and trajectory design learning process.

Necessary Components of a VITD Implementation

As shown in Figure 3, the primary differentiator in VITD is the truly interactive nature of the visualizations. Before defining this level of interactivity, it is important to note that while standard 3D visualizations such as those in Copernicus, GMAT, and STK do have interactivity, this is generally limited to viewpoint changes; e.g., panning, zooming, and translating the scene to view it from different points of view. STK's "3D Object Editor" tool goes a step further and allows selected changes to be made by grabbing and dragging certain control points, but this does not currently apply to spacecraft trajectories. In contrast, the interactivity in VITD requires the following capabilities from the visualization framework as shown in Figure 4.

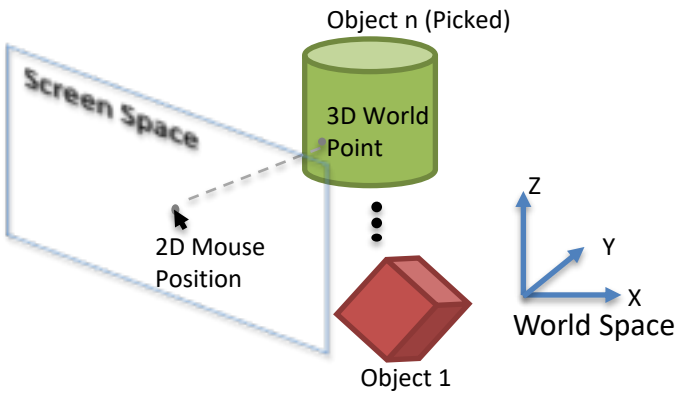

Figure 4. The basic components of a VITD implementation work together to determine how objects should be altered based on user interaction.

Mouse Position: Detect the pixel location of each mouse event, including button click, button unclick, and drag. This capability already exists in any visualization that allows users to pan, zoom, and translate using the mouse.

Mouse Picking: Transform a selected pixel location (a 2D point) into a 3D point in the world space of the scene being viewed. While the pixel location depends only on the width and height of the window displaying visualizations (called *screen space*), the world space is highly dependent on the scene being viewed. Its origin may be at the center of the primary body, the spacecraft, or at any other body being visualized. Regardless of how world space is defined, a unique transformation must be defined between it and screen space. Fortunately many such transformations exist in the

literature[7] and in online computer graphics resources. An example is given in the Implementation section below.

Object Picking: Determine the object at a 3D point in world space. Since the list of objects being rendered is always known, in the worst case this comes down to iterating over every object and checking which one's world position overlaps the given point. There are more sophisticated algorithms available, and one of them based on scene subdivision is explained in the Implementation section below.

Object Transformation: If the mouse is moving while clicked, then the picked object should be transformed accordingly. Common transformations include rotation, scaling, and translation, but other transformations could be possible based on need. It is also often useful (and in fact sometimes necessary) to allow the user to constrain these transformations. For example, the user might want to only rotate about the body-fixed X-axis, or only translate in the world X-Z plane. A comprehensive VITD implementation will take these different types of constraints into account when allowing users to interact with objects.

Picking Feedback Interface: Communicate information about the picked object back to the simulation through an application programming interface (API), which could consist of callback functions, persistent data structures, or other common programmatic interfaces. The communicated information should include the object's state (e.g. position, orientation, size, etc.), pick location in both world and object-local coordinates, and any other pertinent information that can aid the simulation in performing dynamics transformations.

Once the simulation has this programmatic information about how an object has been transformed, it is responsible for determining how any system dynamics should be recomputed. For example if the object being modified is a spacecraft, and the modification is a rotation to the spacecraft, then the application might choose to rotate the spacecraft's velocity vector through the same angle. The simulation would then repropagate the spacecraft from its current epoch and position, but with the new velocity. Alternatively, the application might directly apply the spacecraft orientation change and repropagate its attitude dynamics from the new orientation. Table 1 lists possible actions for various types of object transformations.

Table 1. Possible actions taken during interactive object transformation.

Object Transformation (based on the user's mouse motion)	**Potential Simulation Action** (dynamics repropagated in all cases)
Rotation (angle α about vector $\mathbf{m}$)	Rotate object using (α,$\mathbf{m}$)
	Rotate velocity vector using (α,$\mathbf{m}$)
	Rotate thrust vector using (α,$\mathbf{m}$)
Translation (vector $\Delta\mathbf{r}$)	Translate object using $\Delta\mathbf{r}$, repropagate using full dynamics
	Translate object using $\Delta\mathbf{r}$, repropagate using reduced dynamics when full dynamics are too expensive. e.g. using 2-body dynamics, linearized STM, etc. (Figure 5)
	If object position is a targeting objective, then re-run targeter using previous trajectory as initial guess and using $\Delta\mathbf{r}$ perturbation as the new target

Scale (scale factor *f* along body-fixed axis **a**, can be either **x**, **y**, or **z** axes)	If a stochastic simulation is being performed, then set the new size of the sample ellipsoid using (*f*, **a**). Here the dynamics could be repropagated only after the scaling is finished.

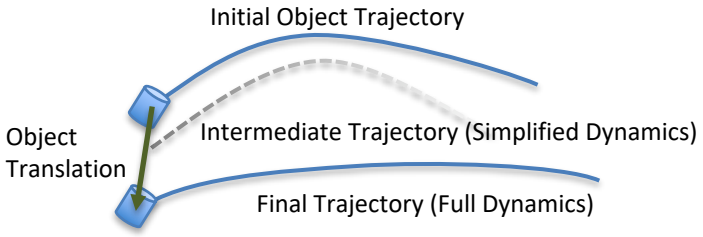

Figure 5. Using simplified dynamics during translation allows the appearance of real-time trajectory modification without the expense of propagating with full dynamics.

VITD Definition

A trajectory design application with both VITD-capable visualizations and VITD-capable simulations would be considered to fully implement VITD. VITD-capable visualizations implement all of the graphics-related capabilities described herein. It should be noted that a VITD-capable visualization framework does *not* generally compute any dynamics. Rather it is responsible only for the graphics-related analysis and transformations of onscreen objects and for exposing those transformations – in real-time – to the simulation that employs its visualizations.

Likewise, a VITD-capable trajectory simulation is one that can ingest the object transformations computed by its visualizations – again, in real-time – and take appropriate action. However, the simulation is not responsible for computing those object transformations. It is important to make this distinction between visualization and simulation capabilities, even though in some applications the simulation and visualization are tightly intertwined at the source-code level. This separation of responsibilities allows the definition of VITD given herein to apply to *any* simulation, including those that use object-oriented programming paradigms or externally-developed visualizations, such as Copernicus and GMAT.

VITD IMPLEMENTATION

In order to test and demonstrate the benefits of VITD, two prototype implementations were created that allow users to rotate a spacecraft's velocity vector and immediately see the results. Both implementations have two coupled components: a simulation and a visualization. The simulation component for the first VITD prototype uses a simple Earth-centered two-body propagator. The second VITD prototype uses the GMAT software for its simulation component; GMAT allows the user to customize the force models, spacecraft parameters, and propagators to suit their needs. GMAT is also Open Source Software (OSS), and as such is readily available to anyone for research and mission development purposes.

The visualization components of both VITD prototypes were created using the OpenFrames visualization API. OpenFrames is OSS that allows developers to add 3D interactive real-time visualizations to their simulation without having to significantly modify the software itself.[8,9] These visualizations have several benefits, including multithreaded rendering for scalability on multi-core computers, feature culling for improved rendering performance, unlimited depth precision for large solar system-scale visualizations, dynamically-computed viewpoints, and virtual reality (VR) rendering support. Various NASA programs, including Copernicus and GMAT, use OpenFrames for their 3D interactive visualizations, as shown in Figure 6 and Figure 7.

The OpenFrames support in GMAT is enabled by the newly-developed OpenFramesInterface plugin, which was created under SBIR by Emergent Space Technologies and released as an OSS tool that enhances GMAT's visualization capabilities. Because the OpenFramesInterface plugin is OSS and available to the public, it was selected as the tool in which to integrate the VITD prototype.

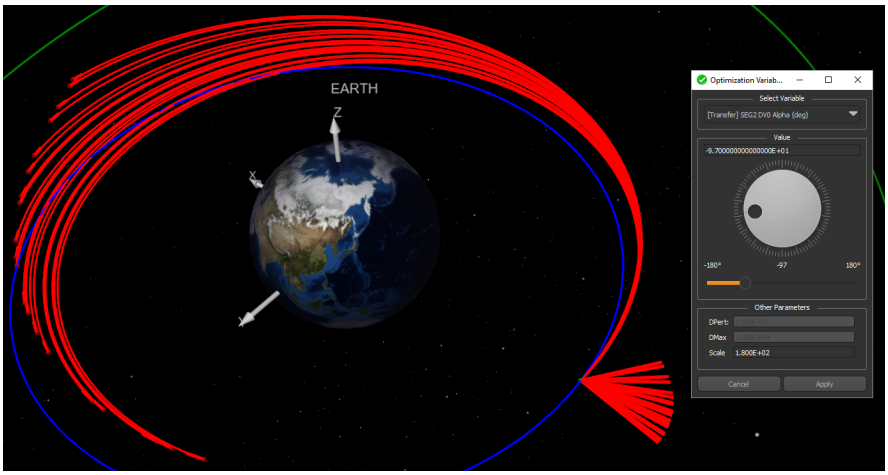

Figure 6. OpenFrames used in Copernicus to visualize all trajectories as the user changes the initial ΔV vector's right ascension with an interactive GUI widget.

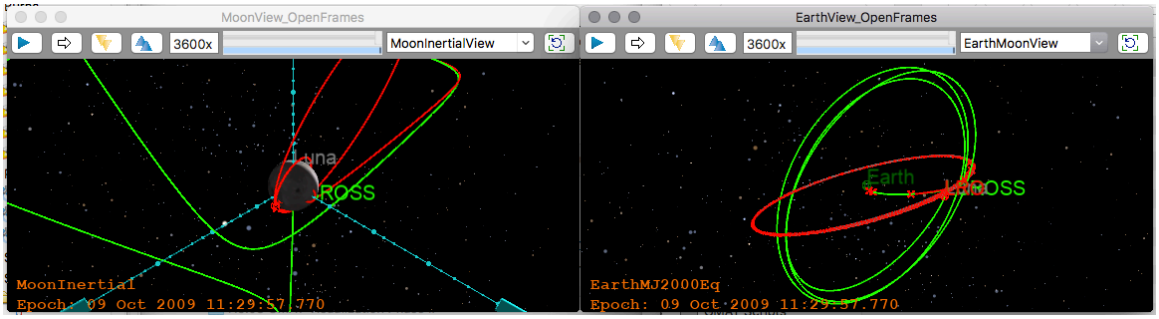

Figure 7. OpenFrames used in GMAT to visualize the LCROSS and LRO trajectories in the inertial Moon-centered (left) and Earth-centered (right) reference frames.

OpenFrames makes use of the OpenSceneGraph (OSG) software, which is a widely-used OSS API for high-performance rendering.[10,11] OSG achieves its real-time performance gains in part by representing visualized scenes using a *scene graph*, which is a tree-like hierarchical data structure that organizes objects by their relative positions. Because a scene graph intrinsically contains position information at each node, it is ideally suited to quickly finding objects at specified locations. Additionally, the scene graph underpinnings of OSG allow it to rapidly perform the following operations necessary for VITD.

Mouse-based Object Picking: OSG uses a ray-shooting technique to determine which object is selected when the user clicks on the screen. First the 2D cursor pixel location is transformed into a 3D ray (an origin and direction) using the camera's current position and orientation. Then the scene graph is traversed; at each node it is determined whether the ray intersects that node's bounding sphere using a simple geometric test. If so, then the node's children are traversed, otherwise they are skipped. In this manner the algorithm quickly hones in on the object selected by the user as well as the selected location on that object's surface. It should be noted that this approach is

exponentially faster than a brute-force search through all objects in the scene, since a tree search is generally logarithmic with respect to the number of objects, whereas an array search is linear.[12]

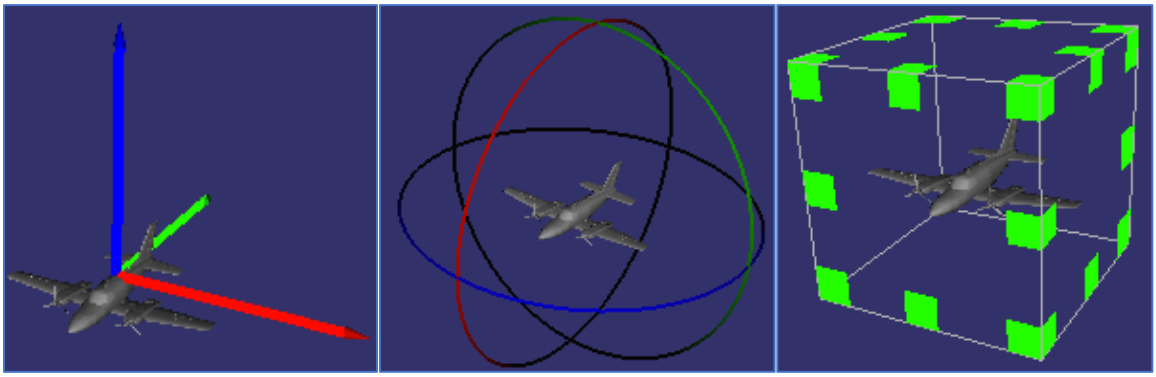

Figure 8. OSG draggers interpret user actions and apply changes to their associated object. Shown are the Translate (left), Rotate (middle), and Scale (Right) draggers.

Object Transformation: OSG contains several objects called *draggers* that allow users to visually manipulate other objects. A dragger is attached to any object in the scene and modifies that object based on user interaction such as mouse-based object picking and dragging. As shown in Figure 8, various types of draggers are available, including trackball draggers (free 3D rotation or constrained rotations about any axis), translation draggers (free 3D translation or constrained along any axis), and scale draggers (scale along any axis).

Two key modifications were made to OpenFrames in order to make it a VITD-capable visualization tool. First, the rotation dragger was integrated and made available for certain types of objects in the scene (e.g. spacecraft). Additionally, the OpenFrames interface was modified to allow external applications to specify callback routines that are executed when the user interacts with a dragger. This allows applications to customize their response to user interaction, which is a key feature of a VITD-capable visualization as described above.

Next, the simulation components of both VITD prototypes were modified by creating a callback routine that is provided to OpenFrames via its new dragger interaction API. This callback is designed to convert spacecraft rotations (due to user interaction) into equivalent rotations of that spacecraft's inertial velocity vector (with a fixed magnitude). It then requests the simulation to repropagate the spacecraft from its current epoch and inertial position, but using the new (rotated) inertial velocity. The simulation performs this propagation using the already-defined spacecraft and force model definitions, and OpenFrames displays the updated trajectory. The speed of this entire process – starting with the user clicking-and-dragging the spacecraft and ending with the new trajectory being displayed – is limited primarily by the speed of the simulation's propagation, which in turn depends on the complexity of the force model. Naturally, the first prototype with a simple two-body propagator has a trivial simulation runtime. However, as shown in the following examples, even for an Earth-Moon-Sun system with a JGM-2 4x4 Earth gravity model and solar radiation pressure, the process of making an incremental change and seeing the propagated results takes a fraction of a second and is sufficient for real-time visualization updates.

EXAMPLES

Exploring the Two-Body System

The two-body system consisting of a spacecraft orbiting the Earth (modeled as a point mass) is often used to introduce students to basic orbital mechanics concepts. This example seeks to show

how VITD can be used to further a student's knowledge of the two-body system by allowing them to easily rotate a spacecraft's inertial velocity vector and see the resulting trajectory and orbit parameters. As shown in Figure 9, the system consists of the Earth and one spacecraft that has been VITD enabled via the rotation dragger described previously. The spacecraft is initially set to a circular equatorial 8000km radius orbit, and the user can place the spacecraft at any point along that orbit. The user visually rotates the spacecraft about any dragger axis, which results in OpenFrames computing a rotation quaternion. This quaternion is then applied to the inertial velocity vector of the target, causing it to rotate with constant magnitude. OpenFrames then sends the new velocity vector back to the simulation, which uses a two-body propagator to compute the new trajectory using the modified velocity vector. It should be noted that in this example position and time are kept constant when rotating the spacecraft's inertial velocity vector.

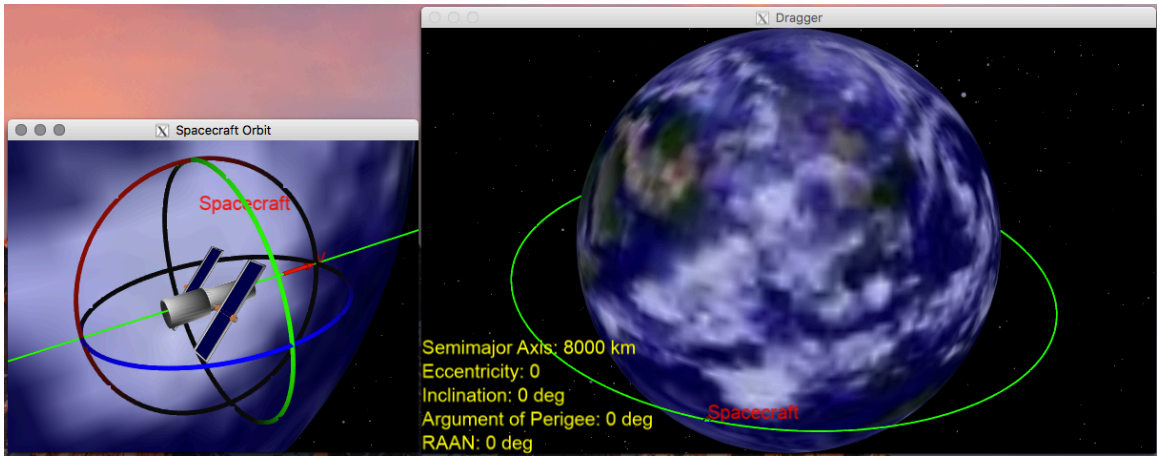

Figure 9. Initial setup of the VITD Two-Body example.
Left: Dragger, Right: Earth-centered equatorial circular (8000km) orbit.

The result of interactively rotating the spacecraft, and consequently its inertial velocity vector, is shown in Figure 10. Here it can be seen that rotating the dragger upwards (i.e. out of the orbit plane) has the effect of changing the orbit inclination and definitively locating the right ascension of the ascending node. It should be noted here that the 730m variation in semimajor axis is due to the spacecraft position being linearly interpolated between propagator steps, while slight variations in eccentricity and argument of perigee are due to limited numerical precision in the simple two-body propagator written for this example.

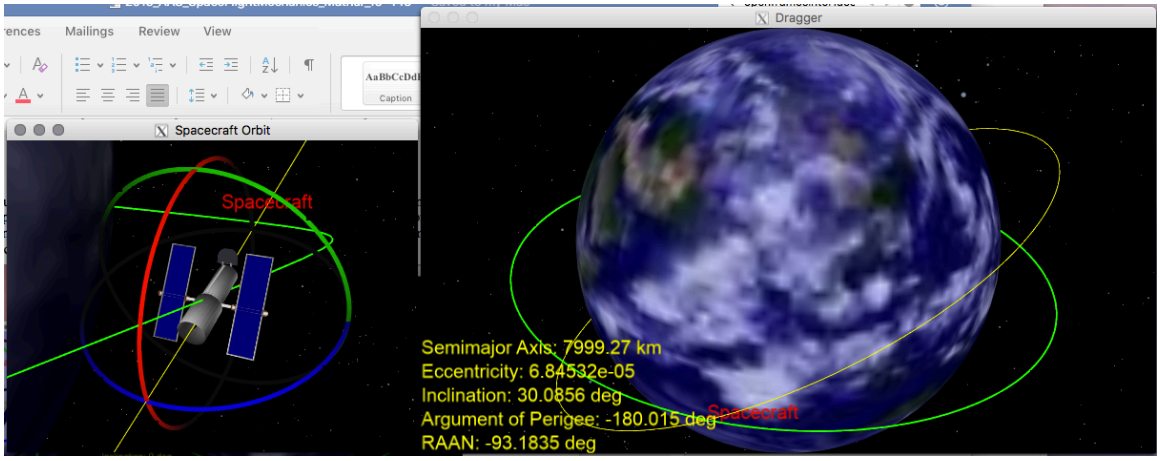

Figure 10. The effect of rotating the spacecraft on its trajectory.

While the result of this simple VITD example is well-known to most people with a basic understanding of orbital mechanics principles, it is nevertheless a meaningful visual and interactive demonstration to students who first start learning this system. The ability to rotate the spacecraft's inertial velocity vector and immediately see how the orbit parameters change is a powerful supplement to existing orbit mechanics teaching tools.

Lunar Free Return in GMAT

A free return trajectory is one that visits a celestial body of interest, and in the absence of any spacecraft maneuvers returns to Earth. Free return trajectories and their benefits for various mission types have been studied for decades, including Lunar free return[13,14,15] and asteroid free return[16] trajectories. Free return trajectory design is generally a two-step process; first an initial-guess trajectory is found, and then optimized according to mission parameters, constraints, and goals. Finding an initial-guess trajectory is critical, because the design space is often very sensitive to variations and contains local minima that may be suboptimal for the desired mission.

This example seeks to find various Lunar free return trajectories using VITD, and also aims to increase the user's understanding of how trajectories in the Earth-Moon system evolve with varying initial conditions. The VITD-enabled OpenFramesInterface plugin for GMAT is used, and the initial GMAT setup contains two spacecraft (the *Dragger* and the *Target*), a numerical propagator, and two output windows (each an instance of the OpenFramesInterface). The Dragger spacecraft is set such that the user can rotate it about any axis, as described previously. The Dragger is linked to the Target, so that rotations of the Dragger affect the velocity vector of the Target. This approach gives the user flexibility to interact with the Dragger in one window, while seeing changes to the entire Target trajectory in another window. Finally, the propagator is set up as follows: Runge-Kutta 8/9 variable step, central body Earth with the JGM-2 4x4 gravity model, including Luna and the Sun as point masses, and enabling spherical solar radiation pressure.

Table 2. Initial trajectory parameters for the Dragger and Target spacecraft using the Modified Keplerian state parameterization.

Epoch	RadPer	RadApo	Inc	RAAN	AOP	TA
22 Dec 1999 11:59:28.000	15000 km	450000 km	5°	0	0	0

The initial orbit of the Dragger and Target spacecraft are set identically, with parameters given in Table 2. These parameters result in a highly elliptical geocentric trajectory whose apogee is beyond the lunar orbit but does not perform a lunar flyby. Figure 11 shows the location of all objects at the time when the Target reaches a distance equal to the lunar orbit radius.

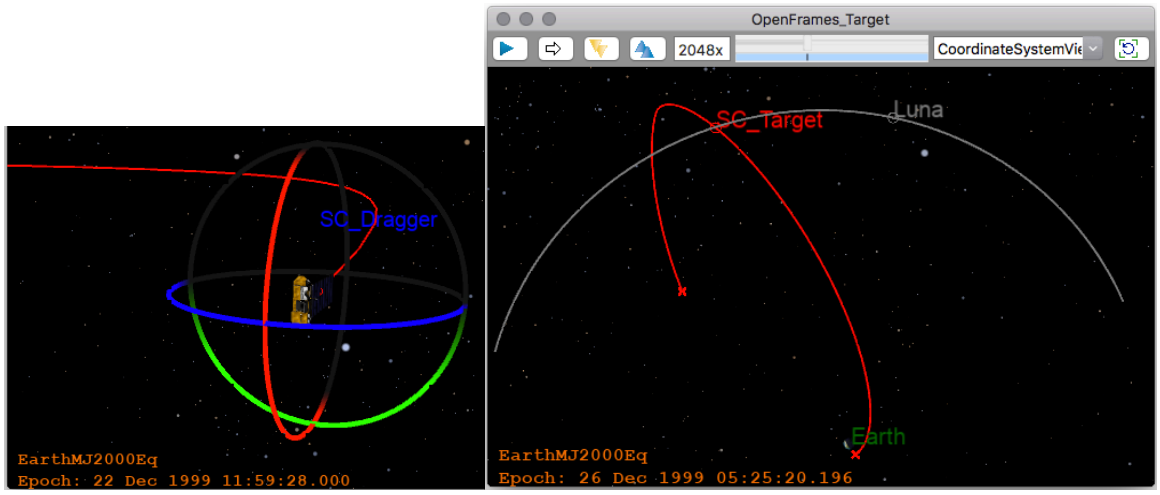

Figure 11. Initial setup of the VITD Lunar Free Return example. Left (cropped): Dragger, Right: Earth-Moon system with Target trajectory.

It can be seen that at this time, the Moon is to the right of the Target spacecraft from the shown viewpoint; i.e. it is considerably behind in its trajectory if the goal is for the Target to encounter the Moon. Intuition into multibody dynamics suggests that there are at least two possible solutions to this position mismatch. One solution is to delay the Target's departure epoch to give the Moon enough time to "catch up". This may not be feasible if the launch and staging conditions are based on other mission constraints and are not sufficiently flexible. Alternatively, if the Target trajectory could be rotated clockwise, there may exist a lunar close-approach encounter. This is the approach taken in this example to demonstrate VITD.

In order to rotate the trajectory, the user rotates its initial velocity vector using the Dragger's overlaid rotation handles (Figure 12). It quickly becomes apparent that while the Target does appear to encounter the Moon from a top-down viewpoint, an alternate in-plane viewpoint shows that the Target is considerably below the Moon. This is not unexpected, since the original (top-down) viewpoint from Figure 12 did not provide any information about the out-of-plane location of the Target.

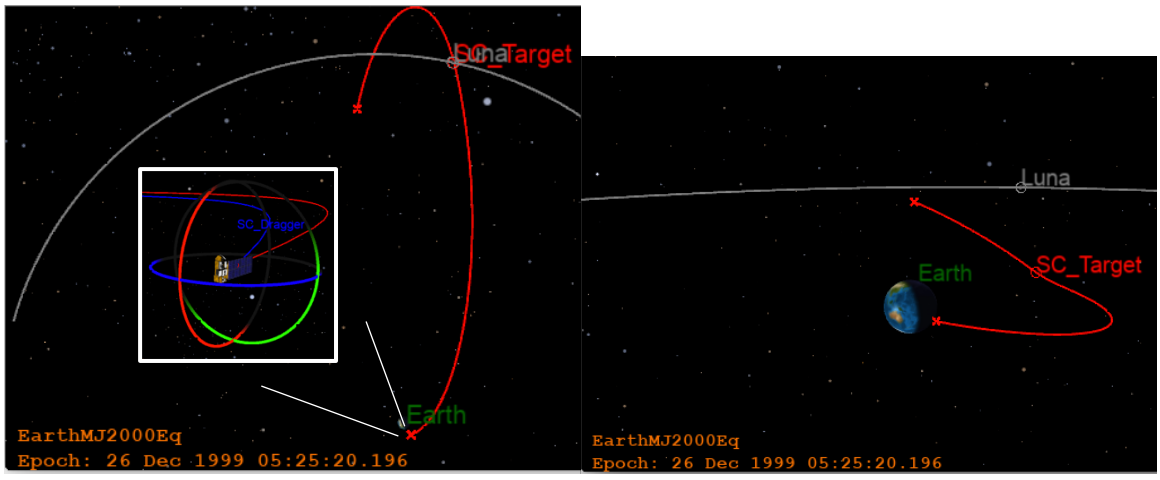

Figure 12. Rotating the Target trajectory at its epoch using the Dragger. Left: top-down view with Dragger inset, Right: in-plane view.

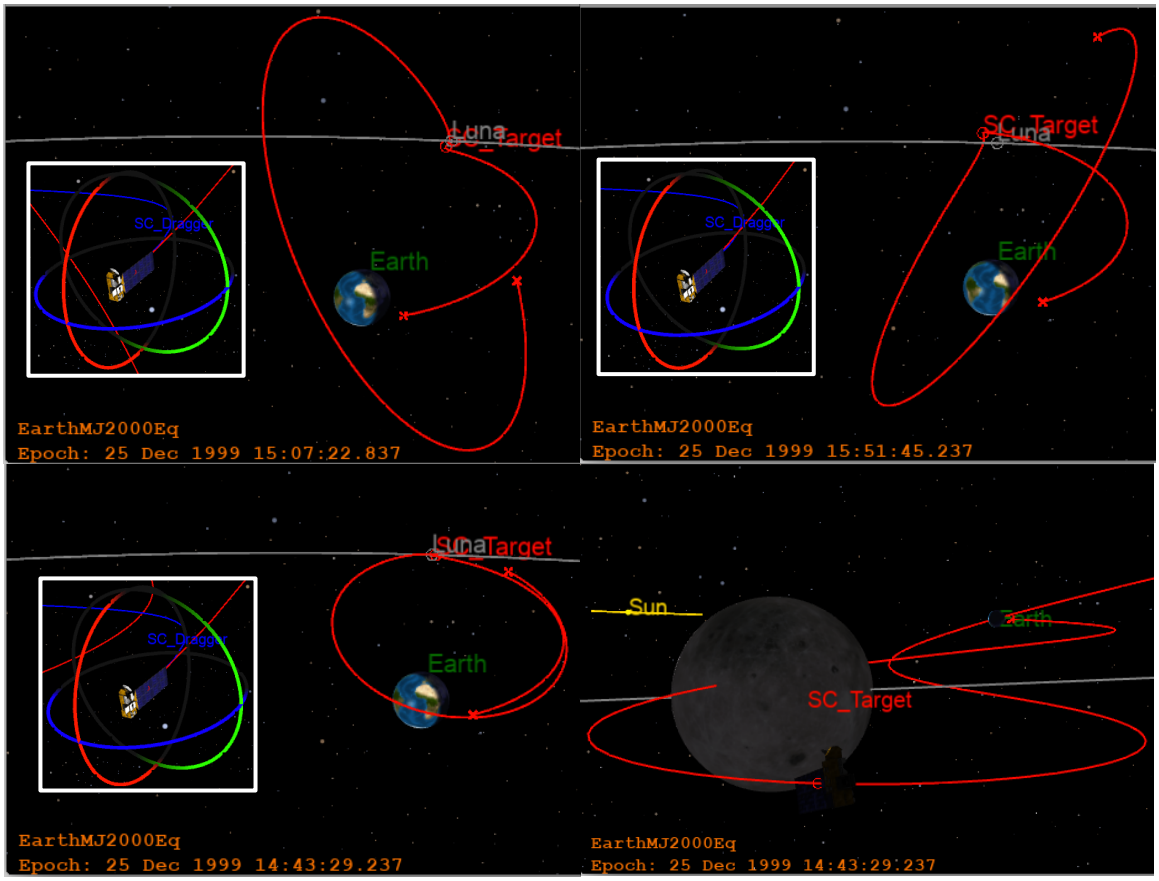

Figure 13. Various lunar free return solutions using VITD. Clockwise from Top-Left: Southern flyby, Northern flyby, In-plane flyby, Earth-Moon-Sun system as seen by Target during flyby.

This planar position discrepancy is quickly corrected by rotating the Dragger upwards, once again using its overlaid rotation handles. Slight adjustments to the expected flyby epoch are easily made as needed with the OpenFramesInterface time-adjustment widgets. Figure 13 shows several possible Lunar free return flyby trajectories found with this implementation of VITD. Specifically, it can be seen that very slight rotations made by the user to the Dragger can have a significant impact on the type of lunar flyby as well as the post-flyby Earth arrival conditions.

It should be noted here that the entire process of rotating the Target trajectory and finding a lunar free return flyby only takes a few minutes for users not already familiar with VITD. Not only is this an important and intuitive learning tool for orbital mechanics students, but it is also an powerful new teaching tool for educators and for experienced trajectory designers who are seeking to further understand the design space for their mission.

CONCLUSIONS

The research performed and described herein presents Visual Interactive Trajectory Design (VITD), an application of visual analytics to the field of trajectory design and optimization. The primary goals of this research were all met, namely:

1. The roles and responsibilities of the visualization and simulation components were defined
2. A VITD prototype was implemented using the OpenFrames visualization API and the GMAT trajectory design software

3. The VITD prototype was used to demonstrate that lunar free return trajectories can be found by visually interacting with the spacecraft and seeing the trajectories change in real time. It was shown that although this is a manual process, it is very fast due to the natural vision-based processing capabilities of all users.
4. The demonstration makes it clear that VITD is useful for a wide variety of end users. Students learning orbital mechanics benefit by being able to instantly see the results of common parameter changes. For example, attempting to change an orbit's inclination at any point other than the ascending/descending node will immediately make it evident that this is not the appropriate location for a plane-change maneuver. Additionally, educators benefit from VITD because it allows them to demonstrate trajectory design techniques in the classroom in real time (in concert with current teaching methods), and integrate those lessons into assignments that allow students hands-on experience with modifying trajectories and seeing the results immediately. Finally, mission design engineers benefit from having a fully interactive tool with which to explore the design space during pre-Phase A mission design, to quickly generate initial-guess trajectories for further optimization, and to easily see the effects of arbitrary perturbations on existing trajectories.

An additional benefit of the VITD prototype implementation presented herein is that it is free Open Source software (OSS). The OpenFramesInterface plugin to GMAT is distributed freely for use with the latest public release of GMAT R2018a, and the VITD-enabled version of the plugin will be distributed freely for use with the next public GMAT release. This provides a free and easily-accessible new tool to augment existing trajectory design tools and methods for learning, teaching, and mission design purposes.

FUTURE WORK

While the theory of VITD presented herein is broadly applicable, the VITD demonstration developed by this research is a prototype and therefore many improvements can be made with further development. For example, enabling translation and time draggers in addition to the existing rotation dragger would allow users to edit all aspects of the state. Furthermore, allowing users to freeze certain states or to affect states in non-cartesian state parameterizations (e.g. inclination, apoapse radius, etc…) would also improve users' ability to design trajectories in various scenarios.

Another improvement that takes advantage of recent technological advances is support for VITD inside Virtual Reality (VR) environments such as the Oculus Rift and HTC Vive.[17] By employing a user's vast intrinsic stereoscopic vision processing capabilities, VR allows users to view, understand, and interact with vast greater quantities of data than traditional 2D monitors. While prior research into immersive technologies in trajectory design has been performed[18], it predates modern commercial low-cost VR systems. As a result of related research performed by Emergent Space Technologies, OpenFrames now has the ability to render any scene either to a 2D monitor or to any VR environment supported by the OpenVR[19] software development kit, including the Oculus Rift and HTC Vive. This ability has been enabled in the OpenFramesInterface plugin for GMAT, and can visualize any GMAT mission script in VR. While grabbing and dragging objects in VR is not yet supported, doing so would require only minor changes. Specifically, the Mouse-based Object Picking capability of VITD would have an equivalent (but more simple) VR-based Object Picking counterpart. Since devices in VR inherently exist in 3D space (not in 2D space like a mouse), getting the 3D position of the VR controller becomes trivial. All of the subsequent VITD steps (Object Transformation, Feedback Interface, etc.) remain the same.

ACKNOWLEDGMENTS

The research presented herein was performed at Emergent Space Technologies under the Phase II SBIR contract NNX16CG16C. The author would like to thank the SBIR COR, David Folta

(NASA GSFC), for his support during the performance and various presentations of this SBIR. The author would also like to thank Steven Hughes (NASA GSFC) for his support and invaluable feedback, as well as Darrel Conway (Thinking Systems Inc.) for his critical GMAT-related support during the design, development and testing phases of this SBIR.

REFERENCES

[1] Systems Tool Kit (STK) developed by Analytical Graphics, Inc. http://www.agi.com/products/engineering-tools

[2] J. Williams, J. S. Senent, C. A. Ocampo, R. Mathur, "Overview and Software Architecture of the Copernicus Trajectory Design and Optimization System", 4th International Conference on Astrodynamics Tools and Techniques, May 2010.

[3] S. P. Hughes, R. H. Qureshi, D. S. Cooley, J. J. Parker, and T. G. Grubb, "Verification and Validation of the General Mission Analysis Tool (GMAT)," AIAA/AAS Astrodynamics Specialist Conference, 2014.

[4] K. Berry, B. Sutter, et. al., "OSIRIS-REx Touch-And-Go (TAG) Mission Design and Analysis," AAS Guidance and Control Conference, Paper AAS 13-095, Breckenridge, CO, 2013.

[5] W. Schlei, "An Application of Visual Analytics to Spacecraft Trajectory Design," M.S. Thesis, School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN, 2011.

[6] W. Schlei and K. Howell, "A Visual Analytics Approach to Preliminary Trajectory Design," 22nd AAS/AIAA Spaceflight Mechanics Meeting, Charleston, SC, 29 Jan.-2 Feb. 2012, pp. 1-20.

[7] H. Zhao, X. Jin, et. al., "Fast and Reliable Mouse Picking Using Graphics Hardware," International Journal of Computer Games Technology, Volume 2009, Article ID 730894.

[8] R. Mathur and C. Ocampo, "An Architecture for Incorporating Interactive Visualizations into Scientific Simulations," 17th AAS/AIAA Space Flight Mechanics Meeting, Sedona AZ, January 2007.

[9] The OpenFrames Application Programming Interface, developed by R. Mathur and Emergent Space Technologies, Inc. http://www.sourceforge.net/projects/openframes

[10] OpenSceneGraph developed by R. Osfield, http://openscenegraph.sourceforge.net

[11] D. Burns and R. Osfield, "Open Scene Graph A: Introduction, B: Examples and Applications," IEEE Virtual Reality 2004 Conference, March 27 - 31, 2004.

[12] J. Döllner and K Hinrichs, "A generalized scene graph," 2000 Conference on Vision Modeling and Visualization, pages 247–254, Saarbrücken, Germany, Nov 22-24, 2000.

[13] R. Mathur and C. Ocampo, "An Algorithm for Computing Optimal Earth Centered Orbit Transfers via Lunar Gravity Assist," AIAA/AAS Astrodynamics Specialist Conference, Toronto, August 2010.

[14] M. Jesick and C. Ocampo, "Automated Generation of Symmetric Lunar Free-Return Trajectories," Journal of Guidance, Control, and Dynamics, Vol. 34, No. 1 (2011), pp. 98-106.

[15] R. Russell and C. Ocampo, "Geometric Analysis of Free-Return Trajectories Following a Gravity-Assisted Flyby," Journal of Spacecraft and Rockets, Vol. 42, No. 1 (2005), pp. 138-152.

[16] N. Bradley and C. Ocampo, "Optimal Free-Return Trajectories to Near-Earth Asteroids," Journal of Guidance, Control, and Dynamics, Vol. 36, No. 5 (2013), pp. 1346-1355.

[17] Oculus Rift developed by Oculus VR, http://www.oculus.com. HTC Vive developed by HTC Corp. and Valve Corp. http://www.vive.com

[18] K. Museth, A. Barr, and M. W. Lo, "Semi-Immersive Space Mission Design and Visualization: Case Study of the Terrestrial Planet Finder Mission," IEEE Visualization Conference Proceedings, 2001.

[19] OpenVR SDK and API developed by Valve Corp., https://github.com/ValveSoftware/openvr