(Preprint) AAS 21-745

VIRTUAL-REALITY-BASED ASTRODYNAMICS APPLICATIONS USING A-FRAME: A TUTORIAL

Dhathri H. Somavarapu[*], Parker Landon[†], Joselyn Busato[‡], Marcus Kaiser[§], Kaela Martin[¶], Elif Miskioğlu[‖], Davide Guzzetti[**]

This paper is a short introduction to the design and development of virtual-reality-based immersive experiences for illustration of astrodynamics concepts. Virtual reality (VR) experiences may be built for any VR capable web-browser using state-of-the-art web software technology, such as the A-Frame development framework. VR experiences based on A-Frame can also be cast to a VR head-mounted-display device directly from a VR capable web-browser. We experimented with the creation of several VR experiences to illustrate astrodynamics concepts, and selected two examples that better demonstrate the key processes for generating astrodynamics-related content in A-Frame. The first example aims to showcase a client-side implementation; the second example is based on a client and server-side technology. A short tutorial is provided to replicate both examples. The deployment of VR experiences for astrodynamics education is also demonstrated in this work by including several VR scenes developed with A-Frame within a massively open online course (MOOC) on multi-body dynamics. In a formal assessment of the MOOC, which included a question on the effectiveness of the VR experiences to their learning, 58.7% of the students indicated a positive response to the VR content. Negative responses to the VR content are attributed to the need for better context-setup for the VR experiences and not to the experiences themselves.

1 INTRODUCTION

Visualization of abstract concepts is important in astrodynamics applications, where human intuition and interpretation of the data plays a critical role.[1,2] Virtual-reality (VR) platform provides an easier way for visualizing orbits and other technical data with the help of a real third dimension. In this manuscript we showcase the mechanisms to create astrodynamics applications using A-Frame (`https://aframe.io/`), a web framework for building virtual reality (VR) experiences. We developed different applications using the A-Frame web technology to illustrate the potential of immersive visualization for creating educational and collaborative frameworks in astrodynamics. In Section 3, we selected two examples among the different applications developed to illustrate the key mechanisms and procedures that are needed to create VR scenes for astrodynamics concepts. Note

[*]Ph.D. Student, Department of Aerospace Engineering, 211 Davis Hall, Auburn University, Auburn, AL 36849.
[†]Undergraduate Student, Department of Computer Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86301.
[‡]Undergraduate Student, Department of Biology, Bucknell University, Lewisburg, PA 17837.
[§]Undergraduate Student, Department of Aerospace Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86301.
[¶]Associate Professor, Department of Aerospace Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86301.
[‖]Assistant Professor, Department of Chemical Engineering, Bucknell University, Lewisburg, PA 17837.
[**]Assistant Professor, Department of Aerospace Engineering, 211 Davis Hall, Auburn University, Auburn, AL 36849.

that this tutorial is based on the design and development experience of the authors. While several possible techniques to create immersive visualization scenes exist, the techniques presented in this work (i.e., those based on the A-Frame framework) are appealing because of a low entry-barrier for new users. A-Frame allows users to create three-dimensional astrodynamics scenes that can be viewed on a desktop web browser or played on a VR headset directly from the browser.

Easily developed virtual reality scenes can be a new educational tool for instructors that teach astrodynamics. We first experimented with VR as an educational tool within astrodynamics by developing A-Frame-based scenes for the massively open online class (MOOC) "Designing the Moonshot: an Introduction to Multi-Body Dynamics", which is being developed in partnership with the NASA STEM Engagement Office. The MOOC covers a short introduction to gravitational multi-body dynamics delivered through five core modules. The first four modules cover gravitational fields, chaotic behavior, orbits in multi-body environments, targeting and optimization. A fifth module is, instead, dedicated to more advanced topics. The modules are hosted on an open-access Canvas page provided by Auburn Online.[3] Each module contains a series of short lecture videos instructing students on key concepts, quizzes on the lecture material, coding assignments that test the student's understanding of the module's content, and virtual reality scenes. These virtual reality scenes were reviewed by students as part of the overall MOOC assessment. Early feedback from the MOOC assessment exercise indicated positive reactions to VR illustrations and experiences.

In this paper, we first introduce fundamental elements of virtual reality in Section 2. We then describe techniques based on the A-Frame suite that the authors designed to illustrate various astrodynamics concepts in VR. We illustrate different techniques to build astrodynamics applications in VR by providing two detailed examples in Section 3. Finally we assessed the effectiveness of VR scenes as part of the MOOC on multi-body dynamics developed by the authors in collaboration with NASA in Section 4. With this short introduction and corresponding tutorial VR scenes, we hope to provide a starting point and a blueprint for the astrodynamics community to leverage VR state-of-the-art technologies for astrodynamics education and research.

2 KEY TECHNOLOGIES AND METHODS

Immersion into virtual worlds has captured the fascination of the human thought for decades.[4] Modern VR has its roots in the 1980s, when the term itself became popular and firms like the Visual Programming Lab began to promote VR technology.[4] However, VR commodity hardware (i.e., VR as consumer electronics) did not appear until around 2012.[4] To facilitate the development and deployment of astrodynamics VR experiences, we start with describing necessary commodity hardware and software technologies.

2.1 Consumer-Grade Virtual-Reality Technology

VR experiences for visualizations of astrodynamics concepts can be played on any VR capable web-browser and head-mounted displays (HMDs). When consumer electronics headsets were first released, VR scenes could only be played on a game engine such as Unity or Unreal with a computer connected to the headset. Today, web-browsers are also capable of playing simple VR scenes on a web-page, as well as cast such scenes to a connected VR headset. Figure 1 displays an example of custom-made VR experience that is delivered via web-browser. This capability provides the following advantages: (1) lower barrier of entry cost-wise and test-wise, (2) portability across various browsers, and (3) potentially higher acceptance of the VR experience from users that are resistant to new technologies. In the applications illustrated in this paper, we leverage this new capability and

develop VR applications for astrodynamics that can both play on the browser and on a VR headset from the browser. In particular, we had access to the Oculus line of headsets from Facebook and VIVE line of headsets from HTC. Recent Oculus headsets can operate as stand-alone devices. However, the HTC VIVE headsets in our possession are currently require a wired or wireless connection to communicate with a computer.

Figure 1. Example of virtual reality exposition created for the Department of Aerospace Engineering at Auburn University. You can visit the scene by navigating to `https://hub.link/cwwv22g`

2.2 A-Frame

In this project, A-Frame is the core web software technology that enables the development of VR scenes to illustrate astrodynamics content. VR scenes that are created via the A-Frame development framework can be visualized on any VR compatible web-browser. A VR compatible web-browser is a web-browser with the appropriate application programming interfaces (API), such as WebVR, that are required to play VR content. The most recent version of popular web-browsers is typically compatible with VR content.

A-Frame is a JavaScript technology suite[5] that provides custom HTML tags and an implementation of the Entity-Component-System (ECS) architecture[6] to develop VR applications for various browsers. An HTML tag[7] is a declarative command for the web-browser to render a particular type of content on the web-page. A-Frame provides various mechanisms to customize the behavior of the VR HTML tags in the VR scene with code extensions and programmatic hooks, such ones that are required for the programmatic animation of scene objects. The key A-Frame mechanisms that are required to create VR scenes within A-Frame are introduced in the following sections.

2.3 Entity-Component-System Architecture

Entity-Component-System (ECS) architecture is the backbone of the A-Frame technology suite. To understand how to develop VR experiences with A-Frame, it is important to understand the ECS architecture[8] and how A-Frame is built on the ECS architecture. In a ECS architecture, every element in the application/scene is treated as an independent “entity”. Each such entity’s attributes and behavior in the application/scene are dictated by what are called “components”. Each component addresses a particular type of attribute or behavior for the entity. For example, a planet may be a entity. The planet has mass, gravity, position, scale and orientation in a reference coordinate

system, which are the components of the entity (i.e., the planet). A "system" is the collection of processes that enforces the attributes and behavior dictated by components for each entity in the application/scene. In the planet example, a "system" process may be a function that reads the mass and gravity (i.e., the components) of the planet (i.e., the entity) and enforces those attributes to dictate the movement of the planet in the given reference coordinate system.

In the ECS architecture, A-Frame is the system. A-Frame provides various HTML tags with default behavior that are considered entities of the A-Frame system. Finally, each HTML tag or entity can have an unlimited number of behavior extensions known as components. A-Frame provides some default components for each entity but also provides a mechanism for users to develop customized behavior components for all entities. More details about how to develop a custom component is described by the A-Frame documentation.[9] In the work and instructions presented in this paper, we use the custom component development method extensively. We discuss the custom component code structure next to gain an appreciation of the behavior extension capabilities that are possible to create astrodynamics content. Understanding custom component development is important because the behavior required to create effective astrodynamics scenes may not be available from the default A-Frame components.

2.4 Custom Component Code Structure

Custom components are the code that allow A-Frame technology to be extensible to astrodynamics applications. Custom components enable the definition of behaviors that allow illustration of astrodynamics concepts, which are otherwise undefined or unavailable in the native A-Frame components. An example custom component code structure is shown in Figure 2. This example component serves to display a two-body orbit.

A custom component is comprised of schema and at least four methods. A schema defines the attributes of the component necessary to render the component in the application. The example "orbit" component, for instance, requires the following information: 1) the URL of the orbit file containing positional data for the orbit, 2) a scale factor to scale the position data by, 3) the name of the orbit to uniquely identify it in code, and 4) the color of the orbit to be displayed in the scene. All four pieces of information are defined as code variables under the schema part of the component definition as can be seen in Figure 2.

One example method is the "init" method that is called when the component is attached to some entity in the scene. In this method, anything necessary to initialize the component should be coded. In the example code shown in Figure 2, the "init" method loads the positional data from the orbit file and creates the line component necessary to display the orbit. The "tick" method is called upon every scene frame refresh and thus should include code to update the component state to appropriate values necessary for that time instant of the scene rendering. For example, in the Hohmann transfer scene described later on in the paper, the "tick" method updates the position of the spacecraft to the current value at that time instant. The "update" method is called when there are modifications to the variables defined in the schema of the component. The "remove" method is called when the component is dis-associated/detached from an entity that it was previously attached to. This method is a good place to add code for cleaning up the state of component.

```
AFRAME.registerComponent("orbit", {
  schema: {
    orbit_file: { type: "string" },
    scale_factor: { type: "number" },
    name: { type: "string" },
    color: { type: "string" }
  },

  init: function() {
    // Do something when component first attached.
    console.log(this.data.orbit_file);
    console.log(this.data.scale_factor);

    function generateRandomHexCode() { ↔ }

    async function read_orbit_data(csv_file_url, scale_factor, csvData) { ↔ }
    this.orbit_data = new Array();
    read_orbit_data(
      this.data.orbit_file,
      this.data.scale_factor,
      this.orbit_data
    ).then(() => {
      // Plot orbits
      var material = new THREE.LineBasicMaterial({
        color: new THREE.Color(this.data.color)
      });
      var geometry = new THREE.BufferGeometry().setFromPoints(this.orbit_data);
      var line = new THREE.Line(geometry, material);
      this.el.setObject3D(this.data.name, line);
      console.log(this.el.getObject3D(this.data.name));
    });
  },

  update: function() {
    // Do something when component's data is updated.
  },

  remove: function() {
    // Do something the component or its entity is detached.
  },

  tick: function(time, timeDelta) {
    // Do something on every scene tick or frame.
  }
});
```

Figure 2. An example custom component code structure

2.5 Client-Side and Combined Server/Client-Side Web Applications

VR scenes are a web application created by code, or instructions, written using the A-Frame framework. Such instructions may contain commands to read or load data files. A web application is any application built to run on a web-browser. The code, or instructions, and data files defining a web application are hosted on a remote server. However, the code for rendering the VR scenes can be executed on a local (e.g., a desktop computer's web-browser) or a remote machine (e.g., a server). Web applications may be classified into two categories depending on whether the underlying code and data files are executed locally or remotely. A web application that is executed locally is called a client-side application. Client-side web applications can only be run within the web-browser of a

client computer to be rendered as a web-page to the user. Alternatively, the code execution and data files that are necessary to render a web application may be strategically distributed across a local machine (the client) and a remote machine (the server). This solution is called a server/client-side application. Note that, regardless of where the code is executed, both client-side and server/client-side require a remote website to permanently host the source code and data files. Fortunately, free content hosting platforms exist, such as the one provided by Glitch (`https://www.glitch.com`). We utilize the free version of the Glitch platform (`https://www.glitch.com`) to host the source code and data files of our VR scenes.

Understanding the differences between client-side and server/client-side applications is important to decide the appropriate architecture when illustrating an astrodynamics concept. For example, if the data required for a VR application is large, a server/client side architecture may be more appropriate. In fact, it may not be practical to include large data volumes in the client side of the application. In such situations, large data volumes may be hosted on a server and distributed to the client on demand. The typical means to request the server for the necessary data is an application programmer's interface (API). The A-Frame allows the the user to develop VR applications both as pure client-side and server/client-side. Both client-side and server/client side architectures are demonstrated in the examples provided in the following section and in the short tutorial in the appendix.

3 EXAMPLES FOR ASTRODYNAMICS APPLICATION DEVELOPMENT

Astrodynamics VR experiences are created by orchestrating the behavior of different application/scene entities. In astrodynamics applications, especially in the ones developed for this work, the following scene elements are necessary: (1) entities for planets, (2) entities for spacecraft, (3) components for spacecraft/planetary orbits, spacecraft trajectories and coordinate axes, (4) textual entities for describing various elements/events in the scene, and (5) animation of the spacecraft/planetary motion. A-Frame provides a rich catalog of entities and tools to include custom 3D objects such as 3D spacecraft models, spheres to display planets, as well as programming hooks to extend the behavior of the entities to animate the entities. One such hook is the possibility to add custom components to a particular scene element. Taking advantage of this programming hook, we developed custom components for displaying spacecraft motion for a given application. This procedural animation technique is described next where we describe how to utilize a programming hook to animate the spacecraft motion within an astrodynamics VR scene.

3.1 Procedural Animation Technique

One of the processes to animate entities in a VR scene using component coding (and the corresponding programming hooks) is procedural animation. Procedural animation is in contrast to other animation techniques that are based on specialized tools in game development frameworks such as Unity and Unreal. Such specialized tools may utilize physics engines to animate objects in a VR scene and require little to no programming. However, physics engines for object animation are not developed for space applications and may not possess an adequate level of dynamical fidelity, especially for multi-body dynamics. In fact, A-Frame possesses a physics engine that works for motion of objects on the surface or close to surface of the Earth, but does not possess any orbital mechanics capabilities. Hence, we resorted to the development of a procedural animation technique that is homegrown.

In procedural animation, we rely only on the positional data history of the object needed to be

animated. The positional data history is typically generated by numerical propagation of the spacecraft/planets/moons motion in space under either two-body or multi-body dynamics, using tools such as Python or MATLAB. In our applications we used MATLAB integration libraries to numerically propagate the motion of a target object and generate the corresponding positional data history. Positional data histories are typically generated at equal time-steps when they are utilize to animate objects in the VR scene. Equal time-steps are not necessary for positional data series that are not used in procedural animation (e.g., for objects such as orbits that are statically displayed in the scenes). For any object that needs to move in the scene, typically the spacecraft in our applications, we associate a custom component that addresses all items related to the movement/animation of that object. Inside the "init" method of this custom component, we load the positional data history for that object into memory, and initialize the position of the object to the first position in its positional data. Inside the "tick" method of the same custom component, we update the current position of the object to the one corresponding to the next time-step in the positional data history. Procedural animation allows for capturing the relative speed of the object in space and visually reproduces two-body or multi-body dynamics without a physics engine. The following two examples showcase both client-side and server/client-side VR applications that also include different forms of procedural animation.

3.2 Example for a Client-Side Application: Hohmann-Transfer Animation Scene

A Hohmann transfer is a fundamental problem in astrodynamics, often used to illustrate trajectory design and optimization in two-body systems. We utilize this canonical problem to illustrate the creation of a client-side VR scene for astrodynamics that leverages the ECS architecture to develop a custom component code for procedural animation in A-Frame. Figure 3 shows a snapshot of the Hohmann-transfer animation scene (`https://jewel-abrupt-tote.glitch.me/`). In this VR scene, the spacecraft begins from an arbitrary position in the inner circular orbit, performs a burn to transfer onto the inner-to-outer transfer elliptical orbit, reaches the apogee of this elliptical orbit, performs another maneuver to transfer to the outer circular orbit, performs one full revolution in the outer circular orbit, performs another maneuver to transfer to the inner circular orbit via the outer-to-inner transfer elliptical orbit. This whole cycle is repeated in an infinite loop in the scene.

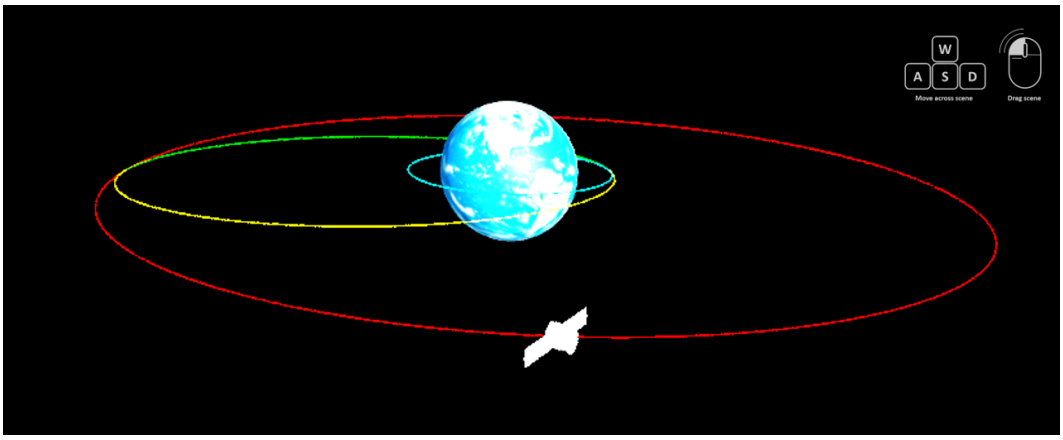

Figure 3. A snapshot of the Hohmann-transfer animation scene. Snapshot edited to enhance printed visual quality

As displayed in Figure 3, the Hohmann-transfer visualization comprises different stationary (e.g.,

Earth and orbit tracks) or animated (e.g., the spacecraft) objects. Positional data for both the orbit tracks and spacecraft animation are generated using MATLAB, as A-Frame does not possess native capability to numerically propagate orbit motion. As spacecraft motion requires the spacecraft to move across all the orbits in the scene, the orbital position data history is necessary at equal time-steps for all orbit tracks (see Section 3.1 for our procedural animation technique requirements). The MATLAB code for generating this orbit positional data history used in the scene is available on GitHub (`https://github.com/dhathris/Hohmann-Transfer`). Then, comma-separated value (CSV) files containing the orbit data generated in MATLAB are uploaded to the Glitch web application for the Hohmann-transfer animation scene as client-side assets.

Finally, A-Frame custom components are coded to display the orbits and animate the spacecraft motion. A-Frame does not provide default entities or components to display orbits. Hence, a custom component for displaying any two-body orbit is developed and is named"orbit" (see Figure 2). Likewise, since the motion of the spacecraft is not associated with the physics engine of A-Frame, animating the spacecraft motion so that it obeys two-body dynamics also require developing a custom procedural animation component for the scene. Thus, we developed a custom component to animate the spacecraft motion along the Hohmann transfer orbit sequence, named "hohmanntransfer". A code snippet for the "hohmanntransfer" component is reproduced in Figure 4. The code for these two custom components, as well as the entire code-base for the Hohmann-transfer scene can be viewed online on Glitch website. The index HTML file for this application imports and collects all the necessary elements of the scene, including defining and attaching the custom components to the scene, so that the animation plays correctly. Interested readers may access the source code for the Hohmann-transfer scene by visiting (`https://jewel-abrupt-tote.glitch.me/`) and clicking on double-fish icon at the top-right corner of the scene in the web browser and then selecting 'view source' button (see Figure 5). Using the same double-fish icon also provides an option to copy the project code to a new Glitch project. This option is a button with label, 'Remix on Glitch'. The steps to recreate the Glitch project for this VR scene are also listed in Appendix A.

```
AFRAME.registerComponent("hohmanntransfer", {
  schema: {
    orbit1_file: { type: "string" },
    transfer_orbit1_file: { type: "string" },
    orbit2_file: { type: "string" },
    transfer_orbit2_file: { type: "string" },
    scale_factor: { type: "number" },
    animation_speed: { type: "number" }
  },
```

Figure 4. Code Snippet for Custom Hohmann Transfer Component

Figure 5. A snippet of the Glitch button (double-fish icon)

3.3 Example for a Server-Client Application: Depiction of Halo Orbits (Earth-Moon System)

To demonstrate a server/client-side implementation of a VR scene for astrodynamics, we created a halo orbit scene. Recall that a server/client-side architecture may be necessary to develop data-heavy web applications, such as the VR scene described next. In this scene, CR3BP halo orbit families are shown around the co-linear Lagrange points. Halo orbits are one of the most fundamental orbit types in the Circular Restricted Three-Body Problem (CR3BP), a dynamical model that may be utilize to render orbit dynamics with the Earth-Moon system.[10] In this scene (`https://respected-deeply-allspice.glitch.me/`), ten halo orbits each around the L_1, L_2 and L_3 points are depicted together with the Earth, the Moon, and a set of coordinate axes. Figure 6 shows a snapshot of the scene.

Since it is not practical to list all thirty of the orbit files as assets in client-side application, these orbits are stored on the server-side and retrieved from the server using an on-demand RESTFul API[11] request at the initialization of the scene by the client web-browser. This process is effectively a client/servers-side implementation of the web-application rendering the halo orbits scene. All the elements of this scene are stationary with no animation. The orbits are displayed in the scene using a custom component "orbit". Note that the "orbit" component here is different from the one developed for the Hohmann-transfer scene. The rest of the elements of the scene, the Earth, the Moon, the Lagrange points, the coordinate axes and the text are displayed using the custom A-Frame HTML tags, as coded in the "index.html" file. On the server-side, the API for returning the orbit positional data history to the client on demand is implemented in the "server.js" file. The script and orbit files can be obtained from the Glitch website (at `https://respected-deeply-allspice.glitch.me/`) using the "Remix on Glitch" button in the drop-down menu of the double-fish icon at the top-right corner of the scene (see Figure 5). This will create a new copy of this project on the Glitch website. The copy contains all the necessary files for the project including the thirty orbit

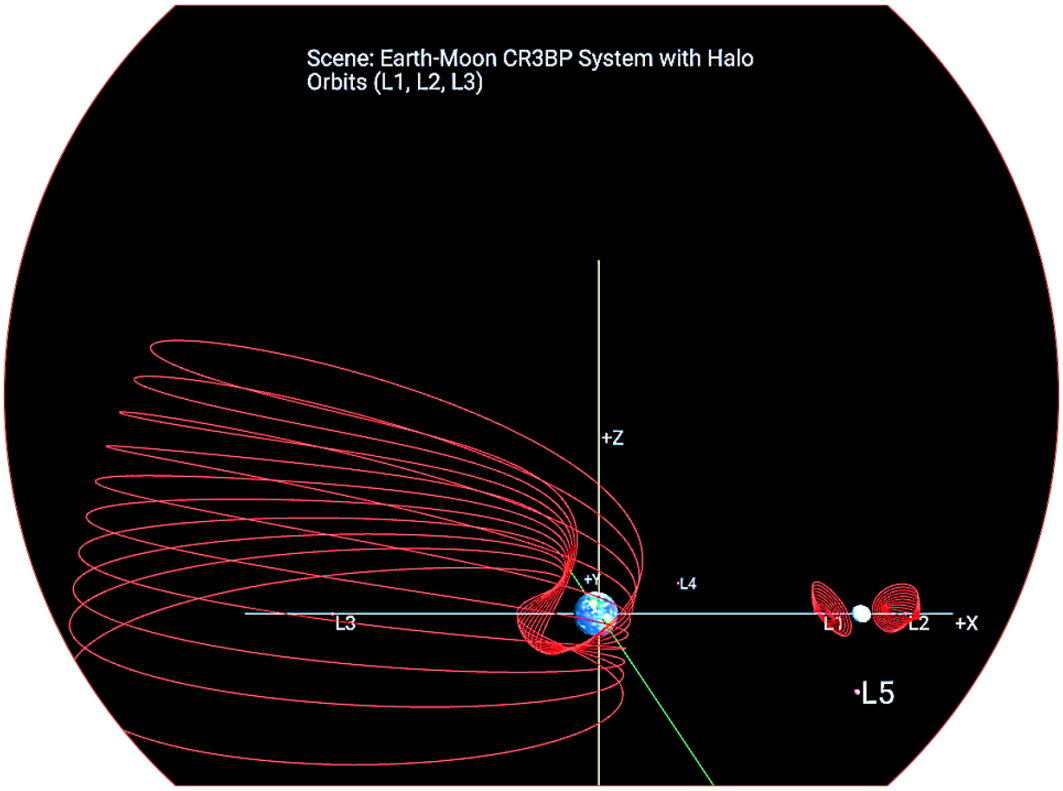

Figure 6. A Snapshot of the CR3BP halo orbit scene

files and the custom component for displaying these orbits. The steps to recreate the Glitch project for this VR scene are also listed in Appendix B. This halo orbit scene illustrates how to develop VR experiences in A-Frame that require large volume data to be stored, accessed and visualized.

4 ASSESSMENT OF VR SCENES AS PART OF MULTI-BODY DYNAMICS MOOC

Multiple VR scenes were developed for the previously described multi-body dynamics MOOC, "Designing the Moonshot," using the techniques described in Section 3.1. These additional VR scenes include:

- A visualization of the Apollo-11 trajectory from Earth to Moon in the inertial frame (`https://fancy-chain-hare.glitch.me/`)
- A particle-flow simulation within CR3BP dynamics (`https://raspy-tranquil-idea.glitch.me/`)
- A Poincaré-map of CR3BP trajectories (`https://peaceful-ebony-driver.glitch.me/`)
- The illustration of CR3BP halo orbit families from Section B (`https://respected-deeply-allspice.glitch.me/`)
- An animation of CR3BP orbit bifurcations around the L_2 equilibrium point in the Earth-Moon system (`https://lake-gold-milk.glitch.me/`)

The inclusion of VR scenes in the multi-body dynamics MOOC provided a unique opportunity to methodically assess the efficacy of VR as a tool to teach astrodynamics. The first assessment of the MOOC modules, which included an assessment of the VR experience, was for usability and accomplished via remote think-aloud sessions. In the think-aloud sessions, students were observed while they reviewed the the modules, and students provided feedback on the content and their interaction with the modules. After conducting interviews, each module was modified based on feedback and the think-aloud process repeated.

Learning outcomes and experience assessment was performed through ASTROCAMP, a two-week virtual event held in June of 2021, where participants had access to the modules for self-paced study complemented by a series of live events. The assessment from ASTROCAMP also provided feedback on the VR astrodynamics scenes we developed via A-Frame. ASTROCAMP assessment was conducted through pre-quizzes, quizzes, short reflection questions, and a final survey. Participants were asked to complete pre- and post-quizzes, watch lecture videos, work through coding assignments, and reflect on their coding assignment work. Within the final assessment, the participants were asked to provide feedback on their learning experiences within the VR scenes. Participants were asked to respond to the following prompt: "Please rate your level of agreement with the following statements – Visualizations of the concepts provided by the VR scenes were valuable to my learning". Each participant could answer either strongly agree, somewhat agree, somewhat disagree, or strongly disagree. Table 1 shows the assessment results. Forty-five individuals responded. More than half the participants (27 of 45) agreed that the VR segments were valuable in their learning. However, based on some open-ended answers, some VR scenes might be more effective with further editing to facilitate scene navigation and explain the content presented in the scene.

Table 1. VR personal agreement responses

Total	Strongly Agree	Somewhat Agree	Somewhat Disagree	Strongly Disagree
45	10	17	14	4

Further analysis was conducted to identify VR effectiveness for students compared to professionals. Our main target audience was undergraduate students. Each demographic was defined by their current or continuing completion of their degree. A total of 44 individuals provided their current degree levels shown in Figure 7. Bachelors equivalent and/or Bachelor pursing students had a 58% positive response rate. Again, the overall general response was positive. Higher degree individual responses were similar. Figure 7 shows that 60% of those with a Masters equivalent or higher had a positive response to VR. Once again, the general response was positive, but VR segments will require additional revisions in the future. Primarily, comments focused on the speed of the scenes.

Additional open-ended questions were included for participants to express their course experiences. They were asked the following question: "What three aspects of the course were most beneficial to your learning?". Eighteen of 45 individuals responded saying that videos were beneficial, five of which explicitly stated the animations and visualizations benefited them.

5 FINAL REMARKS

In this work, techniques to develop visualizations for astrodynamics applications using the virtual-reality platform and the A-Frame browser technology suite are described in detail. This presentation may serve as pseudo-tutorial to develop additional VR experiences for astrodynamics applications.

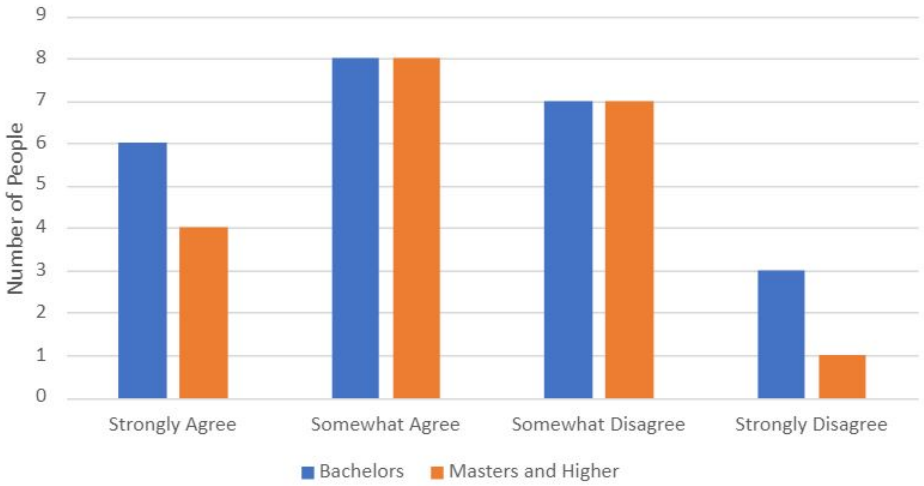

Figure 7. VR responses based on degree level

Insight is provided based on the authors' experiences with the selected platform and the technology. The VR experiences designed by the authors are included in a MOOC to expand the educational impact of the course. An assessment of the course was conducted in June, 2021, and included an assessment of the efficacy of the VR scenes as a teaching tool. More than 50% of the participants indicated a positive response to the VR experiences. While the VR experiences were modified after early think-aloud sessions, the later assessment based on ASTROCAMP revealed that the VR experiences could still be improved by a proper context setup for the learning goals of the course.

ACKNOWLEDGEMENTS

We would like to thank Ms. Kanak Parmar for the management of the ASTROCAMP virtual event for the Multi-Body Dynamics MOOC "Designing the Moonshot: an Introduction to Multi-Body Dynamics", that helped our evaluation of the VR scenes. We are also thankful to Auburn Online for hosting our MOOC and to Dr. Alicia Harkless who tirelessly helped us creating the layout of the course. We would also like to thank the large community of enthusiastic students who participated in the ASTROCAMP virtual event and provided us with invaluable feedback to improve the VR scenes and the course in general. This work was completed under NASA Grant NNH20ZHA001O.

APPENDIX A STEPS TO REPRODUCE THE HOHMANN-TRANSFER SCENE

To reproduce the example Hohmann-transfer scene, complete the following steps:

1. Execute the MATLAB code to generate orbit data using four files (inner_orbit.csv, outer_orbit.csv, transfer_orbit_to_inner.csv, and transfer_orbit_to_outer.csv). Alternatively, download and save these files in the assets folder from the view source option of the Hohmann-transfer scene website.

2. Download and save the assets files (Earth4kTexture.png and Satellite1_0.obj) from the view source option of the Hohmann-transfer scene website.

3. Download and save the root files (index.html, orbit.js and HohmannTranfer.js) from the view source option of the Hohmann-transfer scene website.

4. Go to Glitch website (`https://www.glitch.com`) and sign up to create an account.

5. On the homepage of this website, click on the 'New Project' button at the top-right corner and select the 'glitch-hello-website' option. This step should take you to your new project page on the Glitch website. The project is given an auto-generated three word name separated by dashes.

6. Upload the 4 orbit data files and the 2 assets files to the assets folder in your new project on the Glitch website.

7. Upload the files (index.html, orbit.js and HohmannTranfer.js) to the root of the new project on the Glitch website.

To verify that the scene is working, click on the 'show' icon and select the appropriate option on the new project page on the Glitch website. Compare how your scene plays with the one playing at the web-page (`https://jewel-abrupt-tote.glitch.me/`).

APPENDIX B STEPS TO REPRODUCE THE HALO ORBIT SCENE

To reproduce the example halo orbit scene, complete the following steps:

1. Produce the halo orbit csv files containing position history for halo orbits using a single-shooting targeter algorithm[12] for halo orbits. Grebow's thesis contains the initial conditions to correct to generate ten halo orbits each around the L_1, L_2 and L_3 points.

2. On the homepage of the Glitch website, click on the 'New Project' button at the top-right corner and select the 'glitch-hello-node' option. This step should take you to your new project page on the Glitch website. The project is given an auto-generated three word name separated by dashes.

3. Upload all thirty halo orbit csv files to the assets folder of the new project.

4. Copy the halo orbit files from the assets folder on the client-side to the server-side. For each of the thirty orbit files repeat the following three steps.

   (a) Click on a halo orbit file inside the assets folder. A window appears with a copy option for the URL for that file. Copy the URL by clicking on the 'copy' button.

   (b) Go to the terminal at the bottom of the page and run the command 'wget file-URL', where "file-URL" is the URL copied from the previous step. This command will copy the file from client-side to the server-side.

   (c) Next, rename the file that was copied to server-side to the original filename such as 'Halo_L1_Orbit_1.csv' from the client-side.

5. In the "server.js" file, add the following RESTFul API end-points (See "server.js" file in the source code for the example project as a reference on how to develop the end-points):

   (a) A POST end-point addressed as "/loadHalos"

(b) A GET end-point addressed as "/getHalosSize"

(c) A GET end-point addresses as "/getHaloData"

6. Develop the "orbit" component for using the aforementioned API to display the halo orbits in the scene (See "orbit.js" file in the source code for the example project as a reference).

7. Finally, bring everything together by composing the required HTML tags for the scene in "index.html" file (See "index.html" file in the source code for the example project as a reference).

To verify that the scene is working, click on the 'show' icon and select appropriate option on the new project page on the Glitch website. Compare how your scene plays with the one playing at the web-page (`https://respected-deeply-allspice.glitch.me/`).

REFERENCES

[1] J. Stuart, B. Ramaswamy, T. Lam, N. Guy, F. Laipert, A. Menzies, N. Bradley, A. Mysore, and N. Arora, "Do You See What I See? Interactive Visualization of Mission Design and Navigation," *Proceedings of the 69th International Astronautical Congress*, Bremen, Germany, International Astronautical Federation, October October, 2018.

[2] D. Guzzetti, D. H. Somavarapu, and G. Turner, "An Assessment of Virtual Reality Technology for Astrodynamics Applications," *AAS/AIAA Astrodynamics Specialist Conference, Tahoe, CA*, 2020.

[3] D. Guzzetti, K. Martin, and E. Miskioğlu, "Designing the Moonshot: An Introduction to Gravitational Multi-Body Dynamics," `https://auburncatalog.instructure.com/courses/1004/pages/please-cite-this-material-as/edit`, 2021.

[4] J. Jerald, *The VR book: Human-centered design for virtual reality*. Morgan & Claypool, 2015.

[5] "JavaScript Technologies Overview," `https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview`. Accessed: 2021-07-18.

[6] R. Nystrom, *Game Programming Patterns*. Genever Benning, 2014.

[7] "Structuring the Web with HTML," `https://developer.mozilla.org/en-US/docs/Learn/HTML`. Accessed: 2021-07-18.

[8] "Entity Systems are the future of MMOG development," `http://t-machine.org/index.php/2007/11/11/entity-systems-are-the-future-of-mmog-development-part-2/`. Accessed: 2021-07-30.

[9] "A-Frame Component Development Description," `https://aframe.io/docs/1.2.0/introduction/writing-a-component.html`. Accessed: 2021-07-11.

[10] D. Guzzetti, N. Bosanac, A. Haapala, K. C. Howell, and D. C. Folta, "Rapid trajectory design in the Earth–Moon ephemeris system via an interactive catalog of periodic and quasi-periodic orbits," *Acta Astronautica*, Vol. 126, 2016, pp. 439–455.

[11] L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs: Services for a Changing World*. O'Reilly Media, 2013.

[12] D. J. Grebow, "Generating Periodic Orbits in the Circular Restricted Three Body Problem with Applications to Lunar South Pole Coverage," Master's thesis, Purdue University, West Lafayette, Indiana, May 2006.