Software Design Document
for
Trek VR Room

Version 2.0

Prepared by Lucca Andrade Guedes Galvao Coutinho, Fabio Carrasco, Enrique Guardado, Ruben Heredia, Ari Jasko, Bryan Lopez, Ly Jacky Nhiayi, Ayush Singh, Rizwan Vazifdar, Justin Vuong

Sponsored by NASA JPL

May 11, 2023

Revision History

Name	Date	Reason For Changes	Version
Everyone	11/20/2022	Inserted most information excluding section 6,7, and 9	1.0
Jacky	12/1/2022	Inserted all subsections of 7	1.1
Everyone	12/5/2022	Finalizing all of section 7 and 6	1.2
Everyone	12/9/2022	Polishing grammar/small errors	1.3
Everyone	5/10/2023	Polishing and adding more subsections to 6 and 7	2.0

<Add rows as necessary when the document is revised. This document should be consistently updated and maintained throughout your project. If ANY requirements are changed, added, removed, etc., immediately revise your document.>

1. Introduction

1.1 Purpose

1.1.2. This document will focus on the software design used to build the collaborative features for TrekVR. The purpose of the collaborative features is to allow users to share in real-time exploration of planetary bodies using actual data from the Jet Propulsion Laboratory (JPL).

1.2 Document Conventions

1.2.1. The title for each section is Times New Roman, with a font size of 20.
1.2.2. The subtitle for each section is Times New Roman, bold, with a font size of 14.
1.2.3. The body and bullet points for each section are in Times New Roman, with a font size of 12.

1.3 Intended Audience and Reading Suggestions

1.3.1. This document is for project managers, developers, users, document writers, and people with some background in computer science. This includes staff, faculty, advisors, and NASA JPL liaisons. The recommended sequence for reading starts with the introduction and then moves to a topic the user is interested in.

1.4 System Overview

1.4.1. The JPL Trek VR Room is an open-standard Virtual Reality (VR) application that has been thoughtfully designed for use with VR headsets. The application seamlessly retrieves scientific data from the Jet Propulsion Laboratory (JPL) TrekVR database, which serves as a repository for terrain data of various celestial bodies. Through the astute use of VR technology, the software provides an immersive and distinctive ground-level view of these celestial wonders. This is made possible by creating a VR user interface that empowers users to select a point of interest (POI) on the celestial body. Following the selection of the POI, users are able to access pertinent data related to it and view it in a manner that highlights paths through the celestial body. The VR headset enables users to manipulate their viewing direction and utilize virtual hands to navigate through the user interface.

2. Design Considerations

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

2.1 Assumptions and Dependencies

2.1.1. Hardware: Vive, Meta Quest 2, Windows Computer

2.1.1.1. The consumer of the software is expected to have consistent access to a stable internet

2.1.1.2. A consumer may need to install Steam to run the application

2.2 General Constraints

2.2.1. Hardware Limitation: The application needs to render somewhat high-quality graphics. Therefore, some users will need a strong GPU in order to run it.

2.2.1.1. VR (Wired)

- The processor must be Intel i5-4590 / AMD Ryzen 5 1500X or greater
- The operating system must be Windows 10 or above

- If the VR device is wireless
  - Must have enough batteries
  - Components of the VR device must be functional

2.2.1.2. Development is limited to Unity and C#

2.2.1.3. Network communication: Must have a stable enough connection to the internet to access the NASA API calls.

2.3 Goals and Guidelines

2.3.1. Our team is presently executing the Agile methodology, entailing biweekly scrum meetings within the team and weekly meetings with our liaisons. Our main aim is to mitigate the limitations imposed by the waterfall approach when incorporating new features. To achieve this, we prioritize the development of functional features and then progressively expand the project.

2.4 Development Methods

2.4.1. Our team is utilizing the Agile process and conducting semi-weekly scrum meetings internally, while liaisons are joining us on a weekly basis. Our primary objective is to facilitate the addition of new features without being constrained by the limitations of the waterfall method. As the potential for new features arises, the Agile approach allows us to maintain flexibility in such situations.

3. Architectural Strategies

3.1. Software Used

3.1.1. Programming languages used:

- C#
- Unity C#
- HTML/CSS
- Java
- JavaScript - HTTP request handler/ servlet operations

3.1.2. Development Environments:

- Unity 21.3.8f – Stable long-term support version
- Visual Studio
- Visual Studio Code

3.1.3. External Services and Libraries:

- SteamVR Unity
- OpenXR

3.1.4. Reuse of Existing Software:

- JPL VR Trek - Alvin Quach

3.2. Hardware and Software Interface Paradigms

3.2.1. VR Headsets and controllers

- HTC Vive Headset + Controllers
- Meta Quest 2 Headset + controllers

3.3. Database Usage:

3.3.1. Data Request Frequency

- Data is requested and pulled every session for every host
- Data will be stored within the software as preloaded data.

3.3.2. Requested Data Types:

- Map data
  - Web Map Tile Service (WMTS)
  - Coordinate Data
  - Text Data
- HTTP requests
- Mesh Data
- Locational Information Data Sets
- Text Data

3.4. Future plans for extending or enhancing the software

3.4.1. Integrate OpenXR and grant OpenXR-enabled devices accessibility to JPL VR Trek

3.4.2. Save Session State

- Keep track of changes made by users on the current session

3.4.3. Implement session collaboration

- Text chat enabled for multiple users
- Participants list
- Annotation and custom drawing data sharing between users in the same session

3.4.4. Export current session data into an external file for continuous use

3.5. Memory Management Policies

3.5.1. Data will be saved throughout the session

3.5.2. Data is maintained locally only after the session

3.5.3. User information and data history will not be tracked online

4. System Architecture

The overall system is separated into three different entities:

4.1. User

4.1.1. The user will be able to pick between certain planets and request information on

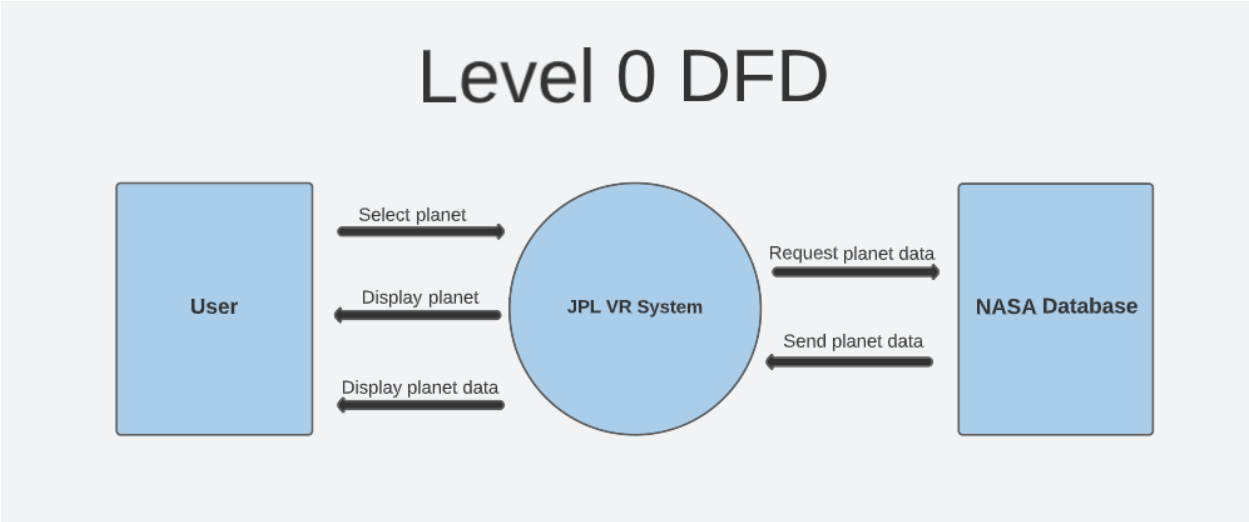
4.2. JPL VR System

4.2.1. The software itself will execute the wanted functions

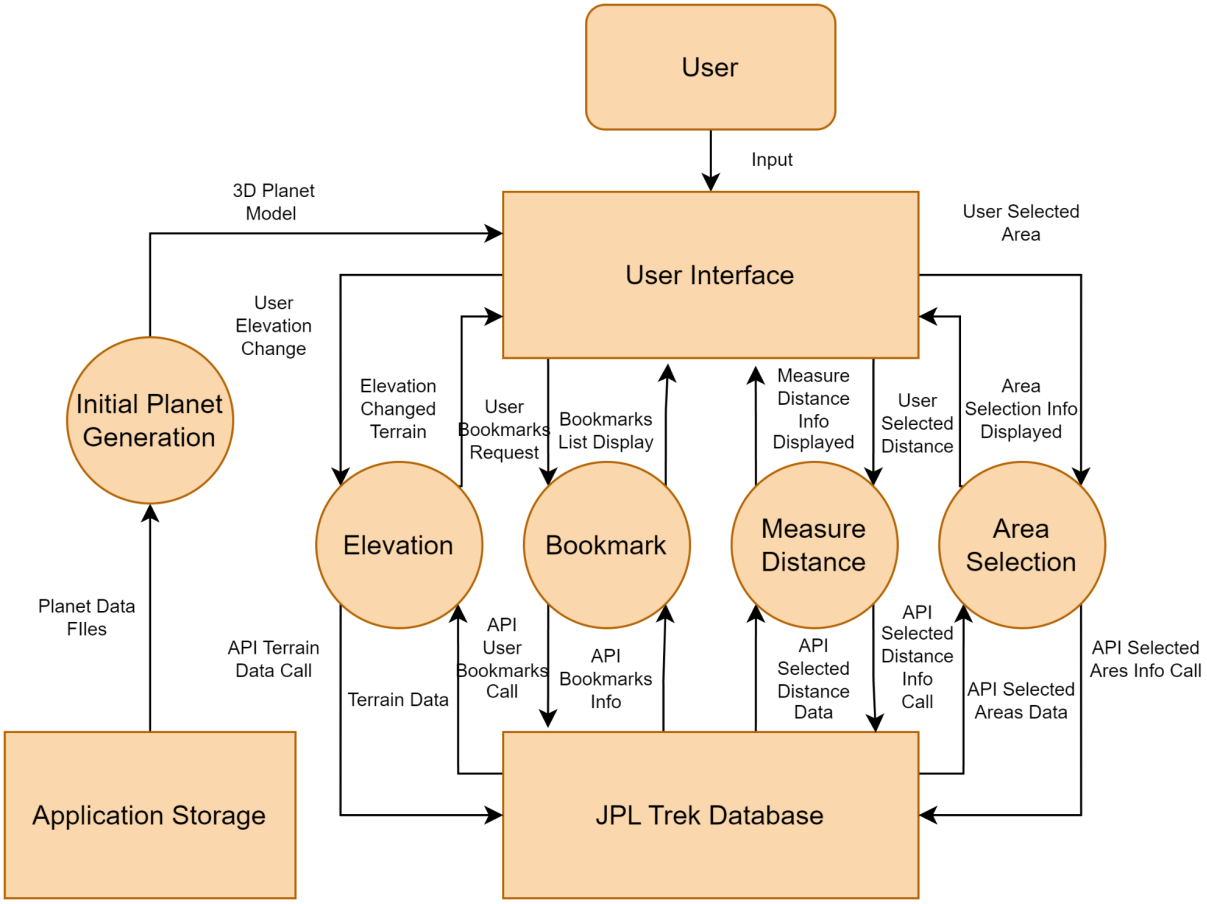
- Display demanded planet
- Display demanded planet data

4.3. NASA Database

4.3.1. This is where all the information is held, where the previous entity is able to grab any data required



Level 1 DFD



The software has two primary functions, namely, to exhibit the planet that the user has requested in virtual reality and to furnish data pertaining to the same. The user is presented with multiple options to choose from with regard to which planet he/she desires to view in VR or access the corresponding data. The software proceeds to execute the function that the user has requested and, subsequently, initiates an API call to the server of JPL to obtain accurate information.

5. Policies and Tactics

5.1 Choice of which specific products used

5.1.1. IDE: Unity Hub and Visual Studio Code

5.2 Plans for ensuring requirements traceability

5.2.1. To ensure the maintenance of up-to-date requirements traceability, our approach entails documenting all meetings and decisions made. This will predominantly involve generating meeting minutes that succinctly capture the key events and outcomes from such meetings. Furthermore, we have established a plan to keep our liaisons and advisor apprised of the project's progress.

5.3 Plans for Testing the Software

5.3.1. The software in the project shall undergo preliminary testing using an HTC Vive device in conjunction with the Unity Game Engine. It is important to note that the JPL VR Trek project is exclusively compatible with the HTC Vive VR headset. Following the integration of OpenXR, the software shall be capable of functioning on alternative devices such as the Meta Quest 2. Subsequently, we shall conduct testing using the Meta Quest 2 device in tandem with the Unity Game Engine. Upon completion of the project, a beta test phase shall commence, utilizing the Meta Quest 2 and SteamVR to evaluate the software product.

5.4 Coding Guidelines and Conventions

To ensure adherence to coding best practices, we will implement the following coding conventions. Firstly, we will utilize comments to enhance the readability of the code by providing transparency on the functionality of distinct sections. Secondly, we will employ the Camel Case standard in line with C# specifications to promote uniformity and clarity in code appearance. Lastly, we will use designated folders when coding with OpenXR and Unity to facilitate the proper sorting of project assets, such as scenes, materials, and 3D models.

5.5. The protocol of one or more subsystems, modules, or subroutines

The implementation phase of the project will involve the adoption of an agile programming approach, characterized by a series of iterative and incremental coding "sprints". These sprints will encompass essential phases, including design, development, testing, deployment, and review. The iterative nature of these sprints will facilitate the completion of the project in a timely and efficient manner.

5.6. The choice of a particular algorithm or programming practice(or design pattern) to implement portions of the system's functionality

Utilizing an agile programming methodology, we aim to establish deadlines for each significant milestone in the project. Moreover, the team will have the opportunity to engage with the liaisons subsequent to every coding sprint to evaluate the quality of the code output.

5.7. Plans for maintaining the software

A significant impediment faced by numerous Unity projects pertains to the comprehensive organization of the project. Our strategy entails preserving the integrity of the software by implementing precisely labeled folders for distinct assets. Establishing a uniform file structure will enhance accessibility and facilitate the maintenance of the code and Unity assets. Additionally, adherence to these practices will assist forthcoming project members in locating code, assets, and pertinent information.

5.8. Interfaces for end-users, software, hardware, and communications

- Unity
- Visual Studio Code
- GitHub
- SteamVR
- PlasticSCM

5.8.1 Hierarchical organization of the source code into its physical components (files and directories).

The project is organized into distinct application functions, namely globe display, JPL server communication through API calls, and Unity environment behavior. The C# files are divided into three categories: Models, Services, and Utils. The Models section contains the data necessary for creating the Mars GameObject. The Services section comprises several classes designed to make API calls to JPL, while the Utils section contains classes that assist in communication with the JPL API calls. The VR controllers are located in the Unity section of the project. The Unity component is responsible for generating lights and facilitating user interaction with Mars.

5.9. How to build and/or generate the system's deliverables (how to compile, link, load, etc.)

The system's deliverables shall undergo compilation, linkage, and loading into a build utilizing the Unity Engine, which is equipped with a C# compiler.

6. Detailed System Design

6.1 Service(Module)

6.1.1 Responsibilities

The primary purpose of the Services module is to establish a connection with the Trek web services and retrieve its products, while simultaneously providing the capability to load the acquired data into a cache. A cache is useful because it stores frequently accessed data, allowing for faster retrieval of that data when it is requested again. This improves system performance by reducing data access times and minimizing the load on the system.

6.1.2 Constraints

This software component facilitates the interaction with the Trek web services through JPL's API. It is crucial to note that this component's proper functioning depends on a successful API call to the Trek web services.

6.1.3 Composition

The Service folder comprises various subcomponents, namely: The RasterSubset folder, utilized for the retrieval of products from Trek web services; the Search feature, which searches through the Trek Services index to locate the technical information required by the TrekVR application; and the Tools folder, intended for retrieving products from Solar Trek.

6.1.4 Uses/Interactions

Upon project instantiation, this component and its subcomponents will directly interface with the Trek web services.

6.1.5 Resources

The proper functioning of this component will require the provision of the endpoint of the API in use, as well as a storage system capable of accommodating the products being accessed.

6.1.6 Interface/Exports

6.1.6.1. TrekRasterSubsetWebService

- Retrieves numerous different data types from the Trek Web Services by JPL

6.1.6.2. TrekSearchWebService

- Searches through the Trek Services index to locate technical information the TrekVR application needs

6.1.6.3. TrekToolsWebService

- Retrieves information from the Solar Trek from JPL

6.1.6.6. ControlPanel

- Displays UI for an Angular application

6.2 XRInteraction

6.2.1 Responsibilities

The XRInteraction directory facilitates the development of virtual reality (VR) encounters that enable users to engage with virtual objects in a seamless and instinctive manner without being burdened by the intricacies of controller mechanics at the lower levels of abstraction.

6.2.2 Constraints

This particular component is designed for employment within a virtual reality environment featuring controllers equipped with trigger buttons, touch pads, and grip buttons. Its core functionalities and attributes are tailored to these types of input and may not be well-suited for integration with VR environments utilizing alternate input methods or controllers.

6.2.3 Composition

The XRInteraction system comprises several subcomponents, including the Terrain folder, which facilitates various terrain interactions, including bounding box selection, distance measurement, and navigation to specific locations on the globe. Additionally, the UI Elements folder provides a mechanism for displaying text labels in a virtual reality environment, while the User Interface folder is designed for the display and interaction with web browsers within a virtual reality setup.

6.2.4 Uses/Interactions

This component defines objects that can be interacted with using VR controllers. It also provides support for grabbing and rotating the globe.

6.2.5 Resources

The module in question requires minimal resources, albeit contingent upon possessing appropriate VR hardware, comprising a VR headset and VR controllers, upon facilitating interactive engagement with objects in a virtual reality milieu.

6.2.6 Interface/Exports

6.2.6.1. XRInteractableObject

- Provides a common interface and default behavior for responding to different types of controller input.

6.2.6.2. XRSubscribableCollider

- Enables the creation of clickable, interactable objects in a VR environment that a controller's buttons can trigger.

6.2.6.3. XRInteractableGlobeTerrain

- Allows developers to add features such as grabbing and rotating the globe, navigating to specific locations, and displaying coordinate lines and labels.

6.2.6.4. XRInteractableTerrain

- Provides access to a number of common features, including bounding box selection, height profile measurement, and sun angle computation.

6.2.6.5. XRBrowser

- Intended to be used to display and interact with an embedded web browser in a VR environment.

6.2.6.6 XRController

- Enable interactive functionality for XR controllers, enabling the utilization of typical VR triggers.

6.2.6.7 XRInteractableTerrainActivity

- The XRInteractableTerrain defines constants that represent various terrain activities.

6.3 TerrainModel

6.3.1 Responsibilities

The TerrainModel directory comprises all essential models for generating the Mars Model. It encompasses a wide range of planet attributes, including but not limited to radius, shadows, and terrain data. The TerrainModel folder serves as the fundamental cornerstone of the application, as it leverages the aforementioned model to execute the JPL API call.

6.3.2 Constraints

Upon the instantiation of this component during the application's launch, all pertinent data of the Mars model will be generated, thereby precluding any further modifications to said data. This preemption of additional data alteration results in seamless execution of the application, devoid of any loading concerns.

6.3.3 Composition

The TerrainModel is comprised of several subcomponents, each serving a distinct function in the creation of the Mars Globe for VR devices. Specifically, the Globe Folder generates the game object, the Layer folder incorporates various layers and materials onto the globe, and the Overlay component generates the longitude and latitude values of the globe. Through seamless coordination, these components collaborate to generate the final output.

6.3.4 Uses/Interactions

This particular component operates in conjunction with the Service Component and the UI Component, thereby facilitating user access to a range of information, including but not limited to the distance between two given points, variations in height levels, and other relevant data. Additionally, the Service Component will draw on planetary information to execute API calls to JPL.

6.3.5 Resources

Minimal resources are required to enable the functionality of this module. During project initialization, mesh data is generated, and the JPL data is already stored as hard-coded files.

6.3.6 Interface/Exports

6.3.6.1. TerrainModelManager

- Creates game objects for the globe and the terrain of the globe

6.3.6.2. GlobeTerrainModel

- Processes data from files inside the project and calculates values from other classes to make the data necessary to create the game object.

6.3.6.3. TerrainConstants

- Generates the necessary constants to make Mars object

6.3.6.4. TerrainLayerController

- Creates the different layers on the Mars Globe object

6.4 Tiff

6.4.1 Responsibilities

The Tiff folder facilitates the processing of tiff images sourced from JPL by developers. Specifically, it employs a conversion mechanism to translate the Tiff images provided by JPL into Unity-compatible objects that can be seamlessly integrated into the Unity engine.

6.4.2 Constraints

If the TIFF image is too large, it may take a significant amount of time to load the application. Tiff images from JPL vary from a couple of megabytes to gigabytes.

6.4.3 Composition

The subcomponents of the Tiff include TiffEncoding, TiffImage, and TiffSampleFormat. All of these classes handle the tiff images from the JPL API calls.

6.4.4 Uses/Interactions

This component defines the TIFF images so that it can be used to generate the Mars Globe.

6.4.5 Resources

The resources needed to run this module are minimal, it requires a library called libTiff, which is a library that handles the tiff images.

6.4.6 Interface/Exports

N/A

6.5 ZFBrowser

6.2.1 Responsibilities

The ZFBrowser directory comprises a multitude of auxiliary functions that are intended for utilization within the control panel. This directory is of paramount importance, as it enables the transformation of the Angular UI into a game object, thus enhancing the functionality of the control panel.

6.2.2 Constraints

The ZFBrowser version must be compatible with the Unity version.

6.2.3 Composition

The ZFBrowser comprises several subcomponents, namely ZFBrowserConfig, ZFBrowserConstants, and FunctionSets. The FunctionSets enable seamless communication between the Unity and Angular applications, and within the FunctionSets module, the ZFBrowser also interfaces with the TerrainModelManager

6.2.4 Uses/Interactions

This component interacts with the control panel, the terrainModelManager, and the Angular application

6.2.5 Resources

To run this, we need the angular application that was built by Alvin Quach (The person who originally made JPL Vr Trek)

6.2.6 Interface/Exports

N/A

7. Detailed Lower level Component Design

7.1 TrekRasterSubsetWebService.cs

7.1.1 Classification

VR room component that retrieves products from the Trek web services

7.1.2 Processing Narrative (PSPEC)

The purpose of this class is to retrieve required and wanted products from the Trek web services.

7.1.3 Interface Description

Input: Base, subset, and search URLs
Output: N/A

7.1.4 Processing Detail

The class gets called every time the project is instantiated

7.1.4.1 Design Class Hierarchy
Parent class: IRasterSubsetWebService
Child class: TrekRasterSubsetWebService.

7.1.4.2 Restrictions/Limitations
Restrictions and limitations are based on the API call, whether it is available to access or not.

7.1.4.3 Performance Issues
N/A

7.1.4.4 Design Constraints
This service will only work given the proper JSON payloads.

7.1.4.5 Processing Detail For Each Operation
GetRasters: Retrieve products from the Trek web services in the form of a JSON file
GetRaster: Iterates over the list of objects and passes each one as a parameter to the callback variable. This allows the function to determine the behavior and action executed on each object without needing to know how GetRasters works.
SubsetProduct: Retrieves products from Trek web services and saves them to a file. If the file is already present; it will then be loaded instead.
SubsetProduct: Retrieves products from Trek web service and saves them to a file. If the file is already present, it will be loaded instead unless file redownloads are forced.
VerifyProductExists: Determines whether a product UUID of the TerrainProductMetaData class matches with any of the rasters
DeserializeResults: Instantiate and convert the given string to a ‘SearchResult’ object

7.2 TrekToolsWebService.cs

7.2.1 Classification
The purpose of this class is to retrieve necessary information from Solar Trek

7.2.2 Processing Narrative (PSPEC)
Retrieve and return the distance between points of interest(POI) and height of POIs from Solar Trek.

7.2.3 Interface Description
Input: Selected POI
Output: Distance between POIs, the height of POIs

7.2.4 Processing Detail
The class gets called when the tool is used.

7.2.4.1 Design Class Hierarchy
Parent class: IToolsWebService

7.2.4.2 Restrictions/Limitations
Restrictions depend on Solar Trek Web Service

7.2.4.3 Performance Issues
N/A

7.2.4.4 Design Constraints
N/A

7.2.4.5 Processing Detail For Each Operation
GetDistance: Returns distance between POIs by invoking a string representing a JSON file.
GetHeight: Returns height of POIs by invoking a string representing a JSON file.

7.3 GlobeTerrainCoordinateLinesController.cs

7.3.1 Classification
The controller is responsible for managing the coordinate elements on the globe.

7.3.2 Processing Narrative (PSPEC)
coordinate lines and labels are rendered on the selected globe. Coordinate lines and labels may not appear if the eye position is not within a certain distance of the globe.

7.3.3 Interface Description
Input: coordinate position, label position, material, and viewer eye position
Output: coordinate lines and labels on globe object

7.3.4 Processing Detail
Gets called anytime a globe object is rendered

7.3.4.1 Design Class Hierarchy
Parent Class**:** MonoBehaviour

7.3.4.2 Restrictions/Limitations
N/A

7.3.4.3 Performance Issues
N/A

7.3.4.4 Design Constraints

Coordinate information not viewable while globe texture is being rendered or if eye position is not near the globe

7.3.4.5 Processing Detail For Each Operation

UpdateVisibility: controls whether the coordinate lines are visible or not
InitMaterials: creates the materials used for coordinate lines, coordinate labels.
RemoveCoordinateLines: removes the coordinate lines from the globe object.
UpdateCoordinateLinesOpacity: changes the coordinate lines opacity based on the viewing distance

7.4 GlobeTerrainModel.cs

7.4.1 Classification

Globe component that generates the terrain data based on mesh data and data given by JPL. Generates a level of details of Mars based on a queue adding the necessary data for each terrain.

7.4.2 Processing Narrative (PSPEC)

The purpose of this class is to generate the proper terrain for Mars.

7.4.3 Interface Description

Inputs: Meshdata
Outputs: None, updates the globe GameObject or data for the globe GameObject.

7.4.4 Processing Detail

The class gets called everytime the project is instantiated

7.4.4.1 Design Class Hierarchy

Sealed class - cannot be inherited

7.4.4.2 Restrictions/Limitations

Cannot have children, must have valid inputs to access data such as having a valid productId,width, and height for function GenerateProductMetadata.

7.4.4.3 Performance Issues

N/A

7.4.4.4 Design Constraints

Must contain valid data from JPL to generate proper representation of Mars

7.4.4.5 Processing Detail For Each Operation

GenerateMesh: Generates Terrain model data from JPL files about Mars
PostProcessMeshData: Adds mesh data to the components of the GameObject given meshdata
CanRescaleTerrainHeight: returns true or false if we can rescale terrain height
RescaleTerrainHeight: rescales the height given a scale input
GenerateProductMetadata: Generates all info currently into metadata in the form of a

TerrainProductMetadata object.

7.5 TextureUtils.cs

7.5.1 Classification
Texture component that generates accurate texture sizes that in return, allow the application to render and appropriately display.

7.5.2 Processing Narrative (PSPEC)
The purpose of this class is to compute the expected texture sizes in bytes. Other functions are to compute mipmap dimensions, compute mipmap size, and to generate mipmaps.

7.5.3 Interface Description
Inputs: Width, height, format, mipmaps
Outputs: Textured width and height in pixels. Mipmaps dimensions.

7.5.4 Processing Detail
This class gets called in the TextureUtilsTest.cs

7.5.4.1 Design Class Hierarchy
This is a public class, and can be accessed through the whole project.

7.5.4.2 Restrictions/Limitations
Input data such as width and height must be of power 2. Textures must be in an uncompressed fashion.

7.5.4.3 Performance Issues
Performance issues could result in incorrect width and height values or issues in rendering due to textures and mipmaps.

7.5.4.4 Design Constraints
The width and height must be of power 2 and textures must be uncompressed.

7.5.4.5 Processing Detail For Each Operation
ComputeTextureSize: Receives width and height. Then return the size calculated by making sure the width and height are within the numerical range of 1.
ComputeMipmapDimensions: Checks to see if the dimensions are of power two. If they are, check the width and height are of size 0 then the mipWidth and mipHeight is 0. Else return the mipWidth and mipHeight by using clamp method.
ComputeMipmapSize: Create TextureCompressionFormat object. Throw an exception if there aren't uncompressed textures. Then call the ComputeMipmapDimensions to compute the mipWidth and mipHeight. Returns computed mipmap size by multiplying mipWidth with mipHeight and bits per pixel.
GenerateMipmaps: Receives an RGBImage and gets the width and height. Create a new method of GenerateMipmaps with the size, bytes, image size, and level.

GenerateMipmaps: Receives an RGBImage. Iterates mipWidth and mipHeight and sets the pixels for mipImage. Return from the method when mipWidth and mipHeight are 1, else rerun this method and reduce those values.

7.6 XRController.cs

7.6.1 Classification
This file registers all activity/events that occur with the VR Headset controllers.

7.6.2 Processing Narrative (PSPEC)
The purpose of this class is to accurately map out "Event Actions" when the user interacts with the controller device. This class takes into account all buttons pressed and sensors that are triggered while using the controller device.

7.6.3 Interface Description
Inputs: VR Headset Controllers
Output: Virtual Reality Responses: Movement, Laser Pointer, Menu Selection, Controller Vibration

7.6.4 Processing Detail
This class is accessed whenever the user interacts with the controller.
Sleep State ⇒ Awake State ⇒ Sleep State

7.6.4.1 Design Class Hierarchy
This is a public class, and can be accessed through the whole project.

7.6.4.2 Restrictions/Limitations
In order for this class to be accessed, the controllers must have sufficient batteries to operate.

7.6.4.3 Performance Issues
If the controller batteries are low, then the controllers may become unresponsive.

7.6.4.4 Design Constraints
User activity is limited to the battery health on both controllers.

7.6.4.5 Processing Detail For Each Operation
OnTriggerClicked/OnTriggerUnclicked:
Returns a response when the user clicks down on the controller trigger.
OnPadClicked/OnPadUnclicked/OnPadTouched/OnPadUntouched:
Returns a response when the user interacts with the touchpad on the controller.
OnGripped/OnUngripped:

Decides whether the user is actively using the controller. If the controller is ungripped for a long time, the device will sleep.
OnMenuButtonPressed:
Returns a response by opening a universal menu overlay.

7.7 PrimaryXRController.cs

7.7.1 Classification
This file handles the activity on the controller that the user primarily operates.

7.7.2 Processing Narrative (PSPEC)
The purpose of the class is to identify the active/idle states of the controller and provide accurate responses based on those states.

7.7.3 Interface Description
Inputs: VR Headset Controllers
Output: Virtual Reality Responses: Movement, Laser Pointer, Menu Selection, Controller Vibration

7.7.4 Processing Detail
This class is accessed whenever the user interacts with the controller.
Sleep State ⇒ Awake State ⇒ Sleep State

7.7.4.1 Design Class Hierarchy
This is a public abstract class that can be accessed through the whole project.

7.7.4.2 Restrictions/Limitations
In order for this class to be accessed, the controllers must have sufficient battery to operate.

7.7.4.3 Performance Issues
If the controller batteries are low, then the controllers may become unresponsive.

7.7.4.4 Design Constraints
User activity is limited to the battery health on both controllers.

7.7.4.5 Processing Detail For Each Operation
GrippedHandler/UngrippedHandler:
Send information regarding the active/idle state of the controller.
MenuButtonPressedHandler:
Sends a response when the menu button is clicked to open the menu user interface on the headset.
Update:
Whenever the user interacts with a key in the application, this function will update the frames viewed on the VR Headset in response to the button/key pressed.

7.8 TrekSearchWebService.cs

7.8.1 Classification
This class serves as a call to the TrekSevices API.

7.8.2 Processing Narrative (PSPEC)
The purpose of this class is to search through the Trek Services index to locate technical information the TrekVR application needs.

7.8.3 Interface Description
Inputs: Search index
Outputs: Search results

7.8.4 Processing Detail
This class is accessed whenever the TrekVR application needs to call the TrekServices API for information.

7.8.4.1 Design Class Hierarchy
Parent Class: ISearchWebService

7.8.4.2 Restrictions/Limitations
The search is limited to 400 items.

7.8.4.3 Performance Issues
N/A

7.8.4.4 Design Constraints
N/A

7.8.4.5 Processing Detail For Each Operation
GetDatasets: Gets datasets from the search index and stores them in the cache.
GetFacetInfo: Gets facet counts from the search index and stores them in the cache.
GetBookmarks: Gets bookmarks from the search index and stores them in the cache.
GetNomenclatures: Gets nomenclatures(search result details) from the search index and stores them in the cache.
GetProducts: Returns product label from the web index.
Search: Gets Http response parameters for the keys and values and returns a Search Result
SearchResults: Takes a JSON response file and returns a Search Result

7.9 TerrainModelManager.cs

7.4.1 Classification
The manager creates a globe using JPL's TIFF DEM and the texture of Mars. There exists only one instance of this class in the life cycle of the project.

7.4.2 Processing Narrative (PSPEC)
The purpose of this class is to generate the globe of Mars.

7.4.3 Interface Description
Inputs: Meshdata
Outputs: None, updates the globe GameObject or data for the globe GameObject.

7.4.4 Processing Detail
The class gets called everytime the project is instantiated

7.4.4.1 Design Class Hierarchy
Sealed class - cannot be inherited

7.4.4.2 Restrictions/Limitations
Cannot have children, must have valid inputs to access data such as having a valid productId,width, and height for function GenerateProductMetadata.

7.4.4.3 Performance Issues
N/A

7.4.4.4 Design Constraints
Must contain valid data from JPL to generate proper representation of Mars

7.4.4.5 Processing Detail For Each Operation
InitializeMaterial: Generates Texture of a Globe given a shader.
InitializeGlobeTerrainModel: Utilizes DEM and triangulation to create the mars globe.

8. Database Design

This application will use the JPL TrekVR Database and API.

9. User Interface

9.1 Overview of User Interface

Upon initiation of the software, the user will be seamlessly immersed into a virtual reality environment replete with an interactive table located at its focal point, thereby affording facile access to the Moon/Mars Trek Program. By manipulating the interactive table, the user will be able to navigate and inspect a detailed 3-dimensional topographical map. From this map, the user will be able to select a desired destination, whereupon they will be promptly transported from the virtual environment to the corresponding terrain.

9.2 Screen Frameworks or Images

Figure 9.2.1.

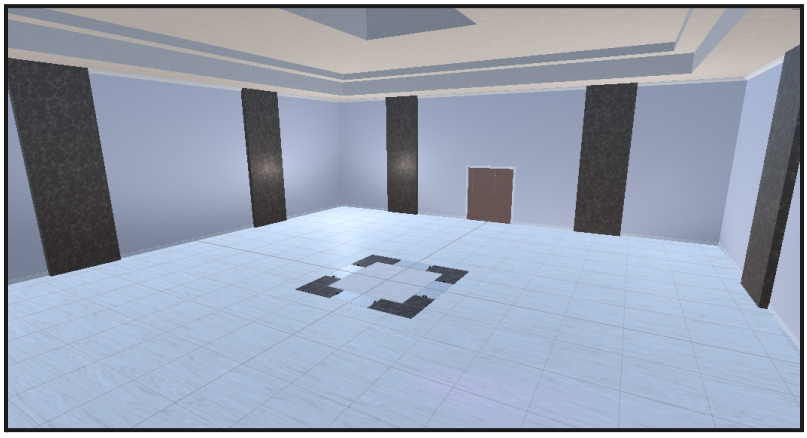


Figure 9.2.2.

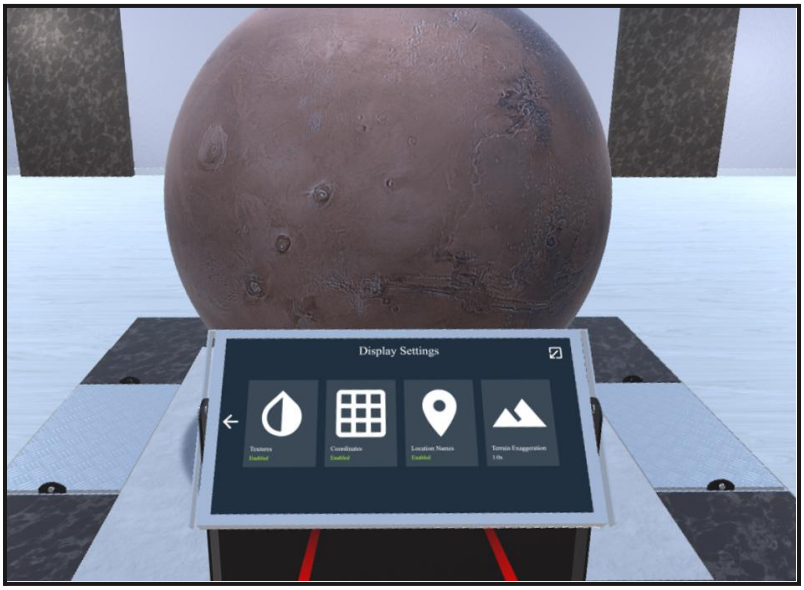


Figure 9.2.3.

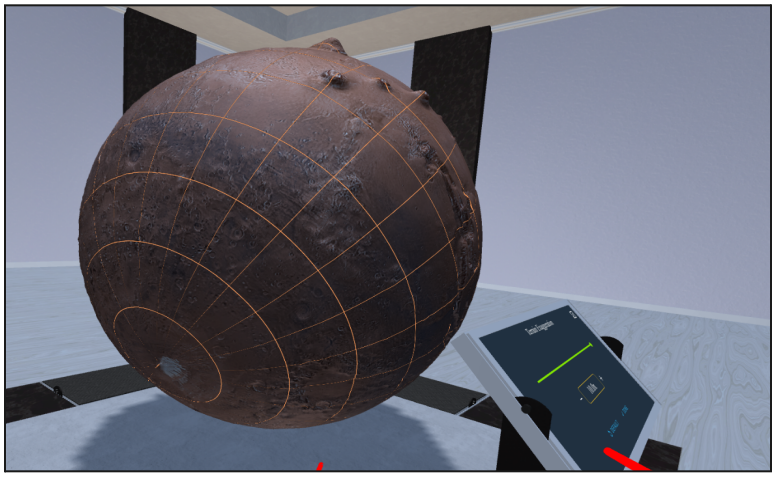


Figure 9.2.4.

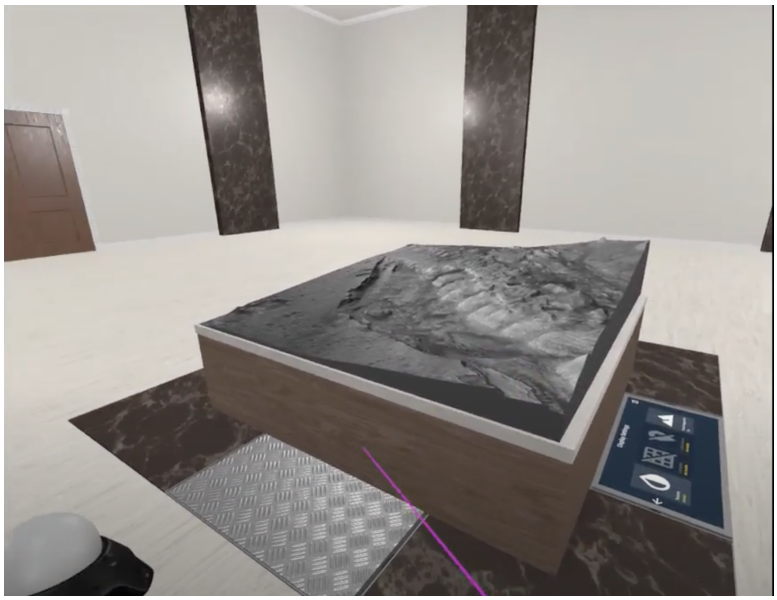
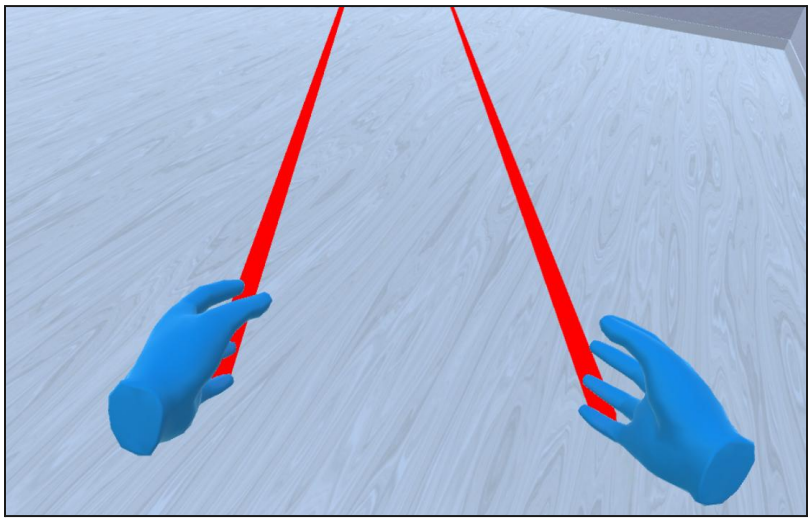


Figure 9.2.5.

Figure 9.2.6.



9.3 User Interface Flow Model

Search Tree for all scenarios:

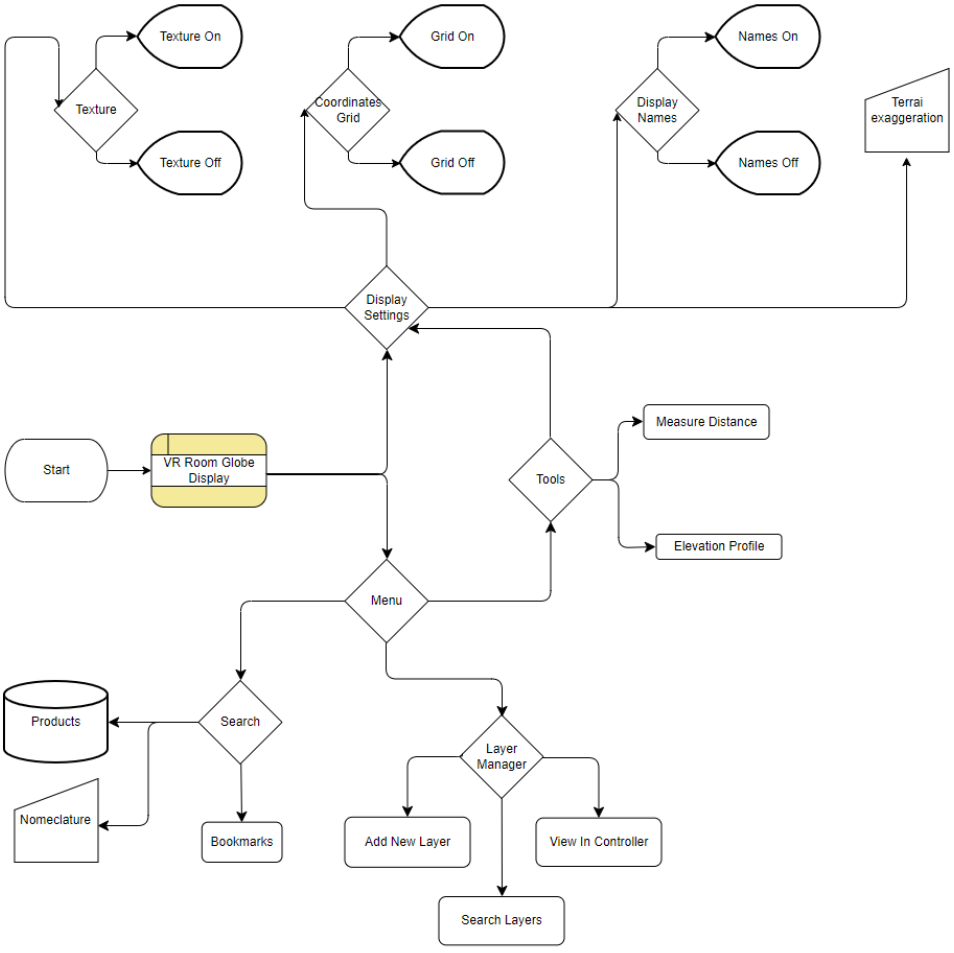
- Display
    - Return to Globe
    - Display Settings
        - Texture
            - On/off turns off or on texture
        - Coordinate
            - On/off to turn the grid off or on
        - Location names
        - Terrain exaggeration
            - Scroll bar gives more exaggeration to texture
        - Can turn off lights

- Activate flashlight

- Menu
    - Search
        - Bookmarks
            - Can see information about the product that was bookmarked
            - Send to controller
                - Products can be seen on the controller
                - Contains information about the product
                - Go back button
                    - Can see a list of products
                - View on globe
        - Nomenclatures
        - Products
            - Displays list of products
            - Select a product
                - See information about product
                - Can choose to see it on globe
            - Sort search results
            - Send to controller
                - Products can be seen on controller
                - Contains information about product
                - Go back button
                    - Can see a list of products
                - View on globe
    - Tools
        - Area selection
            - Display Settings
                - Textures
                    - Adjust resolution and/or brightness of selected object
                - Terrain Exaggeration
                    - Adjust elevation of selected terrain
        - Distance
            - Measure distance
                - Records distance of drawn area
        - Elevation Profile
            - Records elevation of selected area
            - Plots coordinates of the elevation on a graph
    - Layer manager
        - Add new layer

- Layer manager: select an item from the list and add it to layers.
    - Add to layer
        - Adjust the color of selected object
- Search layers
- View in controller

- Options



10. Requirements for Validation and Verification

Requirements	Testing method
Collaboration	To be tested
Chat	To be tested
Annotation	To be tested
Save state	To be tested
Waypoint	To be tested

11. Glossary

An ordered list of defined terms and concepts used throughout the document. Provide definitions for any relevant terms, acronyms, and abbreviations that are necessary to understand the SDD document. This information may be listed here or in a completely separate document. If the information is not directly listed in this section provide a note that specifies where the information can be found.

Acronym	Long Version
[1] SRS	Software Requirement Specifications
[2] SDD	Software Design Document
[3] JPL	Jet Propulsion Laboratory
[4] UI	User Interface
[5] POI	Point of Interest

12. References

Brad Appleton <brad@bradapp.net> http://www.bradapp.net

https://www.cs.purdue.edu/homes/cs307/ExampleDocs/DesignTemplate_Fall08.doc
TerrainModelManager