Software Design Document
for
Trek VR Room

Version .1

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

November 15, 2022

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

<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 font size 20.
1.2.2. The subtitle for each section is Times New Roman, bold with font size 14.
1.2.3. The body and bullet points for each section is Times New Roman, with font size 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, NASA JPL liaisons. The recommended sequence for reading is start with the introduction then move 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 intended for the use of VR headsets. The software retrieves scientific data from Jet Propulsion Laboratory (JPL) TrekVR database. TrekVR database is a database that stores data of the terrain of celestial bodies. With the implementation of VR, the software will provide a ground perspective of these details. This shall be done through implementing a VR user interface through which the user shall be able to select a point of interest (POI) in the celestial body. Once POI is selected, users shall be displayed data pertaining to this POI. As well as the function to observe data on paths through the celestial body. This will be done through the use of a VR headset that will allow the user to turn head and face in different directions as well as control component hands.

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
- Operating System must be Window 10 or above

- If 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 group is using the Agile process, we conducted biweekly scrum meetings by ourselves but weekly with our liaisons. Our main focus was to make it so that features could be added without too much restraint from the waterfall method. We wanted to create working features first and then build up the project from there.

2.4 Development Methods

2.4.1. Our group is using the Agile process, we conducted semi weekly scrum meetings by ourselves but weekly with our liaisons. Our main focus was to make it so that features could be added without too much restraint from the waterfall method. Since there exists a possibility of some new features. Therefore with agile, we would be able to be more flexible in those types of 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 Cod

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 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

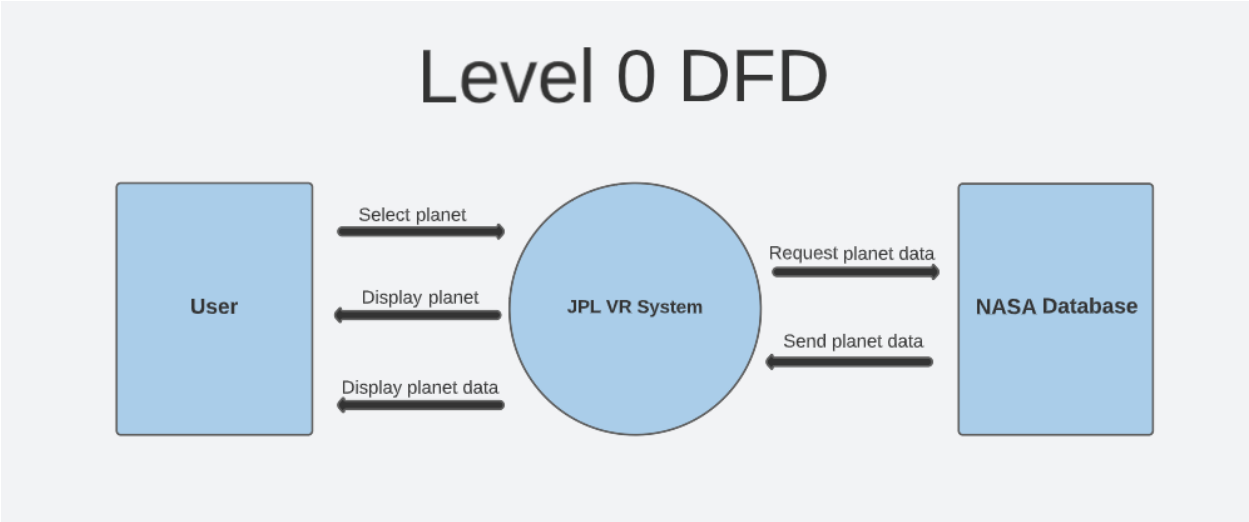
4.2. JPL VR System

4.2.1. The software itself will execute the wanted functions

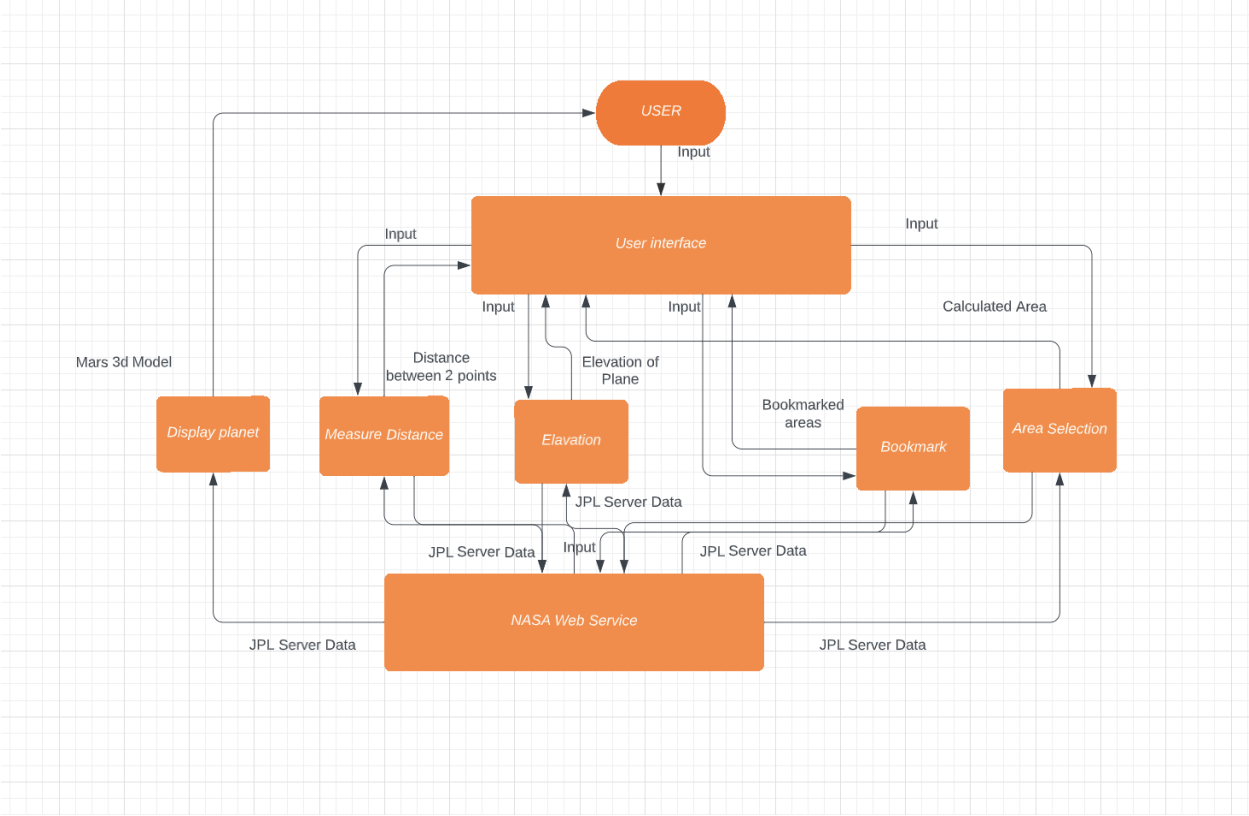
- Display demanded planet
- Display demanded planet data

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 two main functions of this software are to display the user requested planet in VR and to provide the data for the requested planet. The user will be given options as to which planet he would want to see in VR or the data behind it. The software will execute the function that the user requests and make the appropriate API call to JPL's server to get the correct information.

5. Policies and Tactics

Describe any design policies and/or tactics that do not have sweeping architectural implications (meaning they would not significantly affect the overall organization of the system and its high-level structures), but which nonetheless affect the details of the interface and/or implementation of various aspects of the system. Make sure that when describing a design decision that you also discuss any other significant alternatives that were considered, and your reasons for rejecting them (as well as your reasons for accepting the alternative you finally chose). Such decisions might concern (but are not limited to) things like the following (Must include 5.1, 5.2, and 5.3. The rest of these categories or custom ones can be added as needed.):

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. In order to ensure updated requirements traceability, we plan to keep a record of all meetings and decisions. This will mainly be done by creating meeting minutes highlighting events in meetings. In addition, we plan to keep our liaisons and advisor updated with the progress throughout the project.

5.3 Plans for testing the software

5.3.1. Initially, we will test the software in the project using an HTC Vive device and Unity Game Engine. The JPL VR Trek project only operates on the HTC Vive VR headset. After implementing OpenXR, the project will be runnable on devices like the Meta Quest 2. We will test the software using a Meta Quest 2 device and Unity Game Engine at this stage. Finally, we will beta-test the product once the project is complete using Meta Quest 2 and SteamVR.

5.4 Coding guidelines and conventions

In order to keep our code neat and accurate, we will apply the following coding conventions. First, we will use comments on different sections of the code to provide transparency on the functions of the code. Next, we will use the C# standard, Camel Case, to provide a uniform look to the code. Finally, when coding with OpenXR and Unity, we will use designated folders to properly sort project assets like scenes, materials, and 3D models.

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

The team will be using an agile programming approach during the implementation phase of the project. The agile programming approach will consist of coding “sprints.” Each sprint will include a design, development, test, deploy, and review phase. The sprints will repeat until the completion of the project.

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

By using an agile programming approach we will be able to create deadlines for every major milestone in the project. In addition to deadlines, the team will also be able to consult with the liaisons after every coding sprint to reflect on the quality of the code output.

5.7. Plans for maintaining the software

An obstacle for many Unity projects is the organization of the project as a whole. We plan to maintain the integrity of this software by using accurately named folders for different assets. Creating a uniform file structure will make the code and Unity assets easier to access and maintain. These practices will also help future workers on this project locate code, assets, and information.

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

- Unity
- Visual Studio Code
- GitHub
- SteamVR

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

The project will be separated by the applications functions in displaying the globe, calling the JPL server through API calls, and creating the behavior of the Unity environment. The C# files were divided into: Models, Services,and Utils. In the “Models” we have created the data necessary to create the Mars GameObject. In Services, there are multiple classes made to call the API from JPL. Finally for Utils, the classes inside this folder serve the purpose to help communicate with the JPL API calls. The controllers of the VR are located in the Unity section of the project. The Unity component is used to generate the lights and interaction between the User and Mars.

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

The system’s deliverables will be compiled, linked and loaded into a build through the Unity Engine. The unity engine has a C# compiler.

6. Detailed System Design

Most components described in the System Architecture section will require a more detailed discussion. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

This is where Level 2 (or lower) DFD's will go. If there are any additional detailed component diagrams, models, user flow diagrams or flowcharts they may be included here.

6.1 Service(Module)

6.1.1 Responsibilities

The main objective of the Services module is to connect to the Trek web services and access its products while also being able to load the data as cache.

6.1.2 Constraints

This component is used to access the Trek web services via JPL's API. This component will not function correctly without a successful API call to the Trek web services.

6.1.3 Composition

The subcomponents of the Service folder includes: The RasterSubset folder is used to retrieve products from Trek web services, the Search will search through the Trek Services index to locate technical information the TrekVR application needs and Tools folder will retrieve products from Solar Trek.

6.1.4 Uses/Interactions

This component and its subcomponents will directly access the Trek web services upon the instantiated project.

6.1.5 Resources

This component will require the endpoint of the API being used, as well as a storage system to store 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

6.2 XRInteraction

6.2.1 Responsibilities

The XRInteraction folder allows developers to create VR experiences where users can interact with objects in a natural and intuitive way, without having to worry about the low-level details of how the controllers work.

6.2.2 Constraints

This component is intended to be used in a VR environment that uses controllers that have trigger buttons, touch pads, and grip buttons. This component contains properties and methods related to these types of input, and may not be suitable for use with VR environments that use different types of controllers or input methods.

6.2.3 Composition

The subcomponents of the XRInteraction include: The Terrain folder which allows the user to interact with the terrain in various ways, such as selecting a bounding box, measuring distances, and navigating to specific locations on the globe, the UI elements folder that is intended to be used to display text labels in a VR environment, and the User Interface folder that is meant to be used to display and interact with a web browser in a VR 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 resources needed to run this module are minimal, however, appropriate VR hardware is required, such as a VR headset and VR controllers, in order to be able to interact with objects in a VR environment.

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 can be triggered by a controller's buttons.

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.

6.3 TerrainModel

6.3.1 Responsibilities

The TerrainModel folder contains all of the Models necessary to create the Mars Model. It also includes other components of the planet such as the radius, shadows, terrain data, and a lot of other information. This component is the basis of the application, since we use this model that was created in order to actually call the JPL API call.

6.3.2 Constraints

This component gets instantiated when the application starts. When the application is starting all of the data of the Mars model will be created and no more data about it will be altered. Therefore there will be no loading issues when running the application.

6.3.3 Composition

The subcomponents of the TerrainModel include: The Globe Folder which creates the gameobject, the Layer folder which adds all the layers and material to the globe, the Overlay which creates the longitude and latitude values of the globe. These components all work together to create the Mars Globe on the VR devices.

6.3.4 Uses/Interactions

This component is used alongside the Service Component and the UI component which allows the user to access information such as the distance between 2 points, the height levels, and more. The service component will utilize information from the planet to make API calls to JPL.

6.3.5 Resources

Not much resources are required to make this module function. Mesh data is generated throughout project initialization and data from JPL is already stored as hard coded as files.

6.3.6 Interface/Exports

6.3.6.1. TerrainModelManager

- Creates gameobject for the globe and the terrain of the globe

6.3.6.2. GlobeTerrainModel

- Processes data from files inside the project and calculated 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

7. Detailed Lower level Component Design

Other lower-level Classes, components, subcomponents, and assorted support files are to be described here. You should cover the reason that each class exists (i.e. its role in its package; for

complex cases, refer to a detailed component view.) Use numbered subsections below (i.e. “7.1.3 The ABC Package”.) Note that there isn't necessarily a one-to-one correspondence between packages and components.

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
Class gets called everytime the project is instantiated

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

7.1.4.2 Restrictions/Limitations
Restriction 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 it to a file. If the file is already present, it will then be loaded instead.
SubsetProduct: Retrieves products from Trek web service and saves it to a file. If the file is already present, it will be loaded instead unless file redownload is 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 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, height of POIs

7.2.4 Processing Detail
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 through invoking a string representing a json file.
GetHeight: Returns height of POIs through invoking a string representing a json file.

7.3 GlobeTerrainCoordinateLinesController.cs

7.3.1 Classification
Controller responsible for managing the coordinate elements on 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 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, mimpaps

Outputs: Textured width and height in pixels. Mimpaps 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, 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 mimpaps.

7.5.4.4 Design Constraints
The width and height must be of power 2 and texterures 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, 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 touch pad 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 go to sleep.

OnMenuButtonPressed:
Returns a response by opening an universal menu overlay.

7.7 PrimaryXRController.cs

7.7.1 Classification
This file handles the activity on the controller that is primarily operated by the user.

7.7.2 Processing Narrative (PSPEC)
The purpose of class is to identify the active/idle states of the controller and provide accurate responses based off of 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, 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
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 search index and stores them in cache.
GetFacetInfo: Gets facet counts from the search index and stores them in cache.
GetBookmarks: Gets bookmarks from search index and stores them in cache.
GetNomenclatures: Gets nomenclatures(search result details) from search index and stores them in 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

8. Database Design

This application will use the JPL TrekVR Database and API.

9. User Interface

9.1 Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback Information that will be displayed for the user. This is an overview of the UI and its use. The user manual will contain extensive detail about the actual use of the software.

When launching the program the user is put in a virtual reality room. The room will contain an interactive table at the center that will allow the user to use the Moon / Mars Trek Program. When interacting with the table, the user will get a 3D topology map. After selecting a location on the map, the user can transport themselves from the virtual room to the terrain.

9.2 Screen Frameworks or Images

Figure 9.2.1.

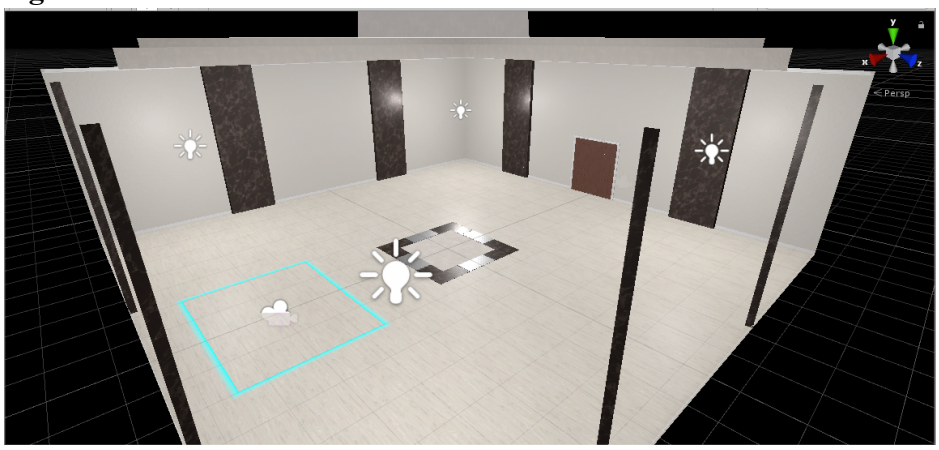


Figure 9.2.2.

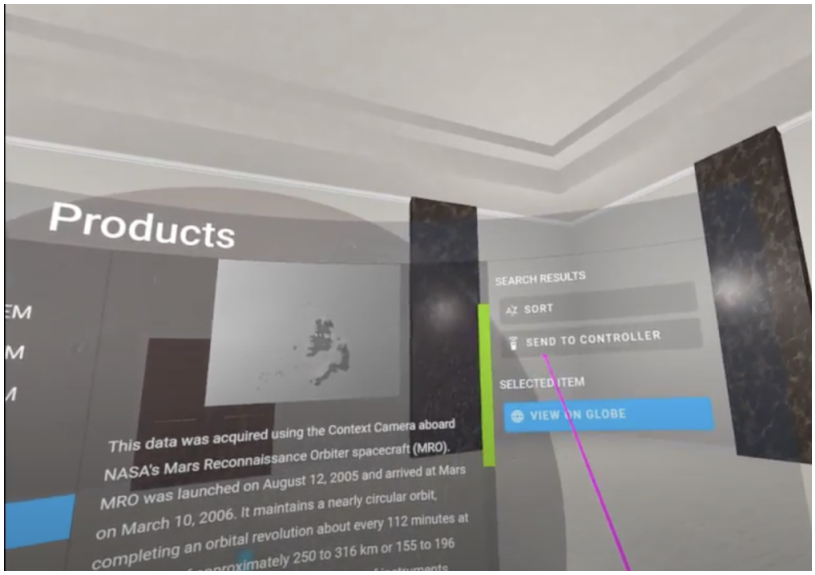


Figure 9.2.3.

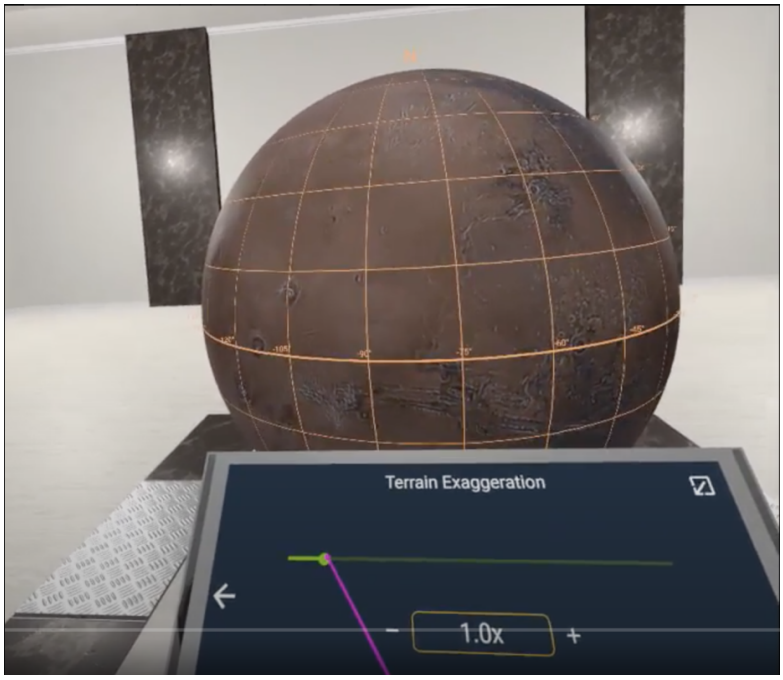


Figure 9.2.4.

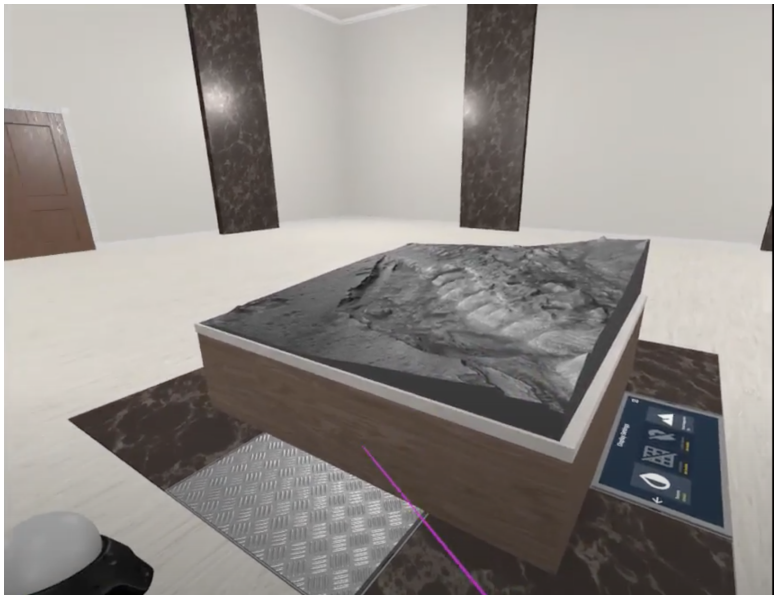
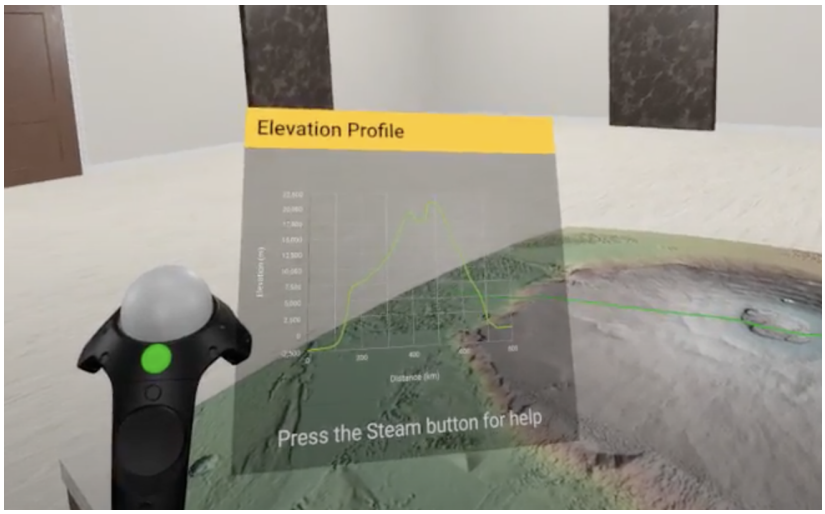


Figure 9.2.5.

Figure 9.2.6.



9.3 User Interface Flow Model

Search Tree for all scenarios:

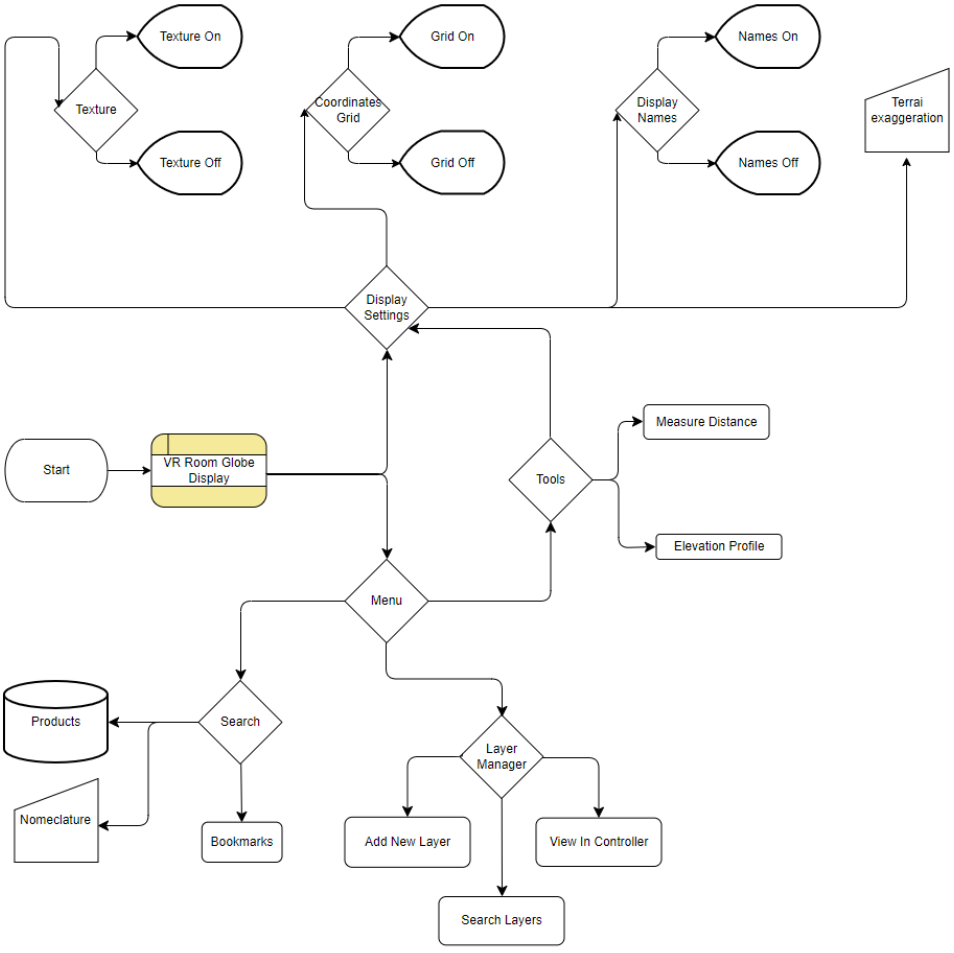
- Display
    - Return to globe
    - Display Settings
        - texture
            - on/off turns off or on texture
        - coordinate
            - on/off to turn grid off or on
        - location names
        - terrain exaggeration
            - scroll bar gives more exaggeration to texture
        - can turn off lights

- Activate flashlight

- Menu
    - search
        - bookmarks
            - can see information about product that was bookmarked
            - 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
        - 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 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

<List any other documents or Web addresses to which this SDD refers. These may include other SDD or SRS documents, user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

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

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