VR Graph Visualization Project Manual

M14.2

December 19, 2022

Table of Contents

Table of Figures

User Manual

The VR Graph Visualization application was created in collaboration between Oregon State University and LPS. Its purpose is to visualize and analyze data generated from models and simulations of High-Performance Computing (HPC) System architectures and algorithms in virtual reality (VR). The Oculus Rift S and Oculus Quest 2 headsets are currently supported. The HTC Vive and other major headsets will be supported in the future. When discussing controllers, references will refer to the Oculus control mappings, but this will be updated to include other mappings when available.

VR Application Setup

The software is compatible with the Oculus Quest 1 and 2. It should work with other headsets, if necessary, but some inputs may not register due to differences in control schemes. This document will proceed to assume that an Oculus Rift S or Quest 1 or 2 is being used.

Prerequisite Software

- Windows 10 or 11
- Oculus Desktop Application

Setup

1. Download and extract the zip file to a convenient location.
2. Connect the Oculus Quest headset to the PC with a compatible link cable.
3. Enable Oculus Link in the Oculus Quest headset.
4. Double click the “VRGraphVisualization.exe” file in the extracted folder.

VR Application Startup

The application will load the dataset placed in input/graphs and will present the network graph in the middle of the scene, as well as the location of the controllers within the scene.

The graph is composed of three types of geometric entities: sphere, cube, or sprite game objects represent nodes, routers, and endpoints from the dataset and the lines represent the connections between them. Generically, the game objects are called ***nodes*** and the lines ***edges***.

VR User Interaction

This section goes into detail on features of this application from the standpoint of the user. It will go over graph interaction, teleportation, the selection feature, and UI menus.

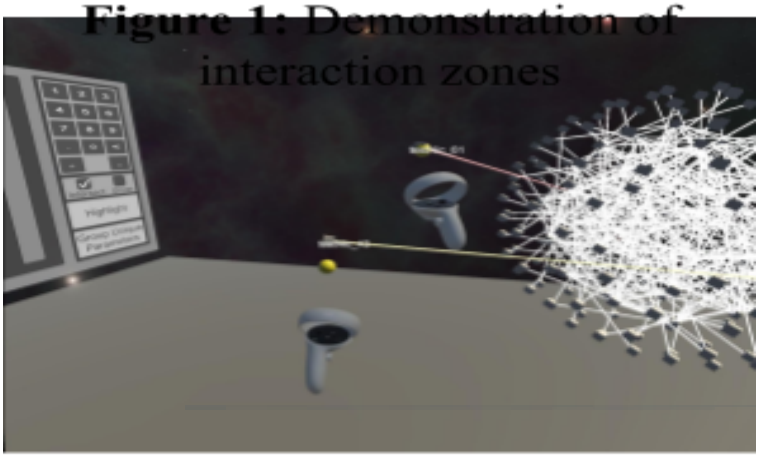
Basic Graph Interaction

At the end of the controllers, there are little yellow spheres; these are called the ***interaction zones*** (**Figure 1**). If one of these interaction zones is brought close to a node, the node will be highlighted yellow. If the interaction zones touch the node, the node highlights red. While the interaction zones are close or touching the nodes, it will display the name of that node from the dataset.

While the interaction zone on one of the controllers is overlapping a node, there are various inputs that can be used to manipulate that specific node. If the trigger on the corresponding

controller is held, the node will be grabbed, and will follow the controller as it is moved. The "X" or "A" button can be used to highlight a node targeted by the yellow zone for the left and right controllers respectively. This highlight will outline the node in blue and add it to a list of selected nodes that can be used by other functionalities in the application. If a highlighted node is grabbed with the trigger, all highlighted nodes will be moved at once. A highlighted node can be deselected in the same way that they are selected, using the "X" or "A" button. Additionally, holding the "X" or "A" button will deselect all currently selected nodes.

Figure 1: Demonstration of interaction zones



Two interaction zones interacting with nodes. The left is touching the node while the right is only near the node

The graph can also be moved as a whole. To do this, hold down one or both grip buttons on the controllers. Holding one grip button will match the graph's position and rotation to that of the controller. If both grip buttons are held, the graph's position will match the average location of both controllers, the rotation will be based on the angle between the two controllers, and the size of the graph can be modified by moving the controllers closer together or further apart.

Teleportation

For better graph viewing, the user may want to change their position in the environment. For this, there is a teleportation feature. If the "B" button is pressed and held, a dotted line will appear starting from the controller and follow a projectile arc (**Figure 2)**. At the end of this line, if there is a blue cylinder, this means that that position can be teleported to, and if the B button is released, the user's position will be updated to the position of the cylinder. If there is

Figure 2: Teleportation

On the left, a valid teleportation spot, on the right an invalid teleportation spot

no cylinder and the line is red, it is not possible to teleport to that location and releasing the B button will not do anything.

The Hand-Held UI Menu

When the “Y” button is pressed on the left controller, a UI menu will appear on top of the left controller. While using the menu a pointer is created on the right controller. This can be used to select options by hovering over the option with the pointer and pressing the trigger button on the right controller. Details for each page in the hand-held UI menu are below.

Figure 3: The home tab on the hand-held menu

Home

The home page (**Figure 3**) allows the graph’s layout and packet visualization method to be changed. The graph’s currently supported options are the various supplied layouts and for each of these layouts, an edge heatmap packet visualization and an object packet visualization. These will be further discussed in the anim section of this document. To change these options, select the desired layout and visualization using the dropdowns at the top and then select “apply” to reload the graph with the selected options. This reload might take a while. If the scene stops updating it hasn’t crashed, just wait for the application to resume execution.

Metrics

The metrics page (**Figure 4**) contains useful metrics to help evaluate the dataset. These metrics are as follows:

- Order: The number of vertices in the graph.
- Size: The number of edges in the graph.
- Maximum Size: The number of edges that would be in the graph if every vertex were connected to every other vertex.
- Minimum Degree: The smallest degree of any vertex in the graph.
- Maximum Degree: The largest degree of any vertex in the graph.
- Average Degree: The average degree across all vertices in the graph.
- Degree Variance: The variance of the degrees of all vertices in the graph.

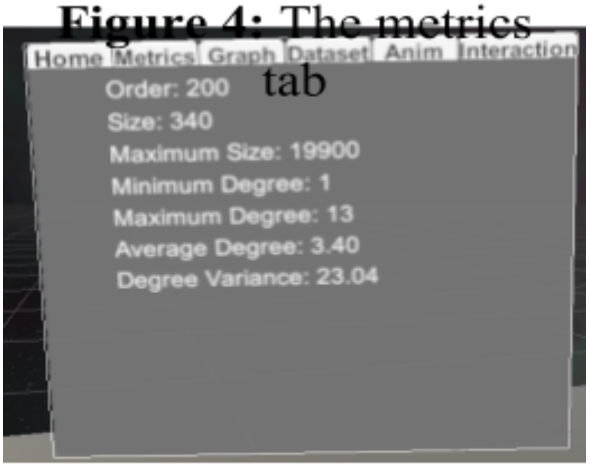


Figure 4: The metrics tab on the hand-held menu

Graph

The graph page (**Figure 5**) allows general settings about the graph to be modified. Currently, the two supported graph settings are to reset the graph’s scale back to the default scale and

reset the graph's position back to its origin. Additionally, the graph section of the handheld menu supports the ability to save states of the graph. These saves persist between uses of the application. Three instances of the graph can be saved, allowing the user to load the graph's state at any time. These slots are selected using the dropdown in the middle of the menu and then selecting the desired operation to be performed on this slot. This data is stored in the form of a json file which attempts to mirror the dataset.

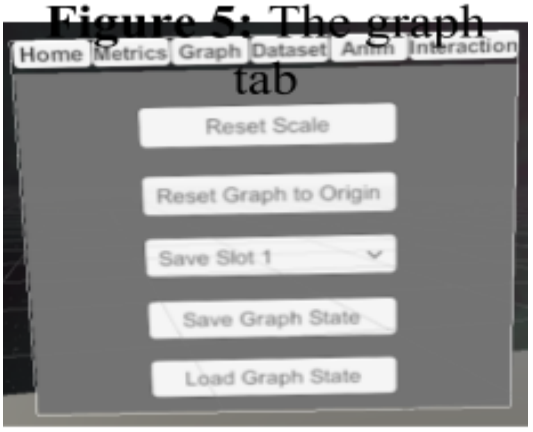


Figure 5: The graph tab

The graph tab on the hand-held menu

Dataset

The dataset page (**Figure 6**) allows the graph dataset to be changed to any of the other dataset files in the local folder used for storing them. All available dataset files are presented in a dropdown list. Once the dataset is selected, click the "Apply" to switch the visualization to the newly selected file.

Figure 6: The dataset tab

The dataset tab on the hand-held menu

Anim

The "anim" tab (**Figure 7**), short for animation, allows the parameters of the packet visualization animation on the graph visualization to be controlled. The animation starts paused at the first tick of the simulation. On the UI page, the central button is a play/pause button that halts or continues the animation of packets as they travel through the graph. The buttons alongside this play/pause button add or subtract speed at various intervals. This controls the playback speed of the packet animation. It's possible to have a negative speed, which will play the animation of the packets in reverse. There is also a reset button below this row of buttons, which when pressed will reset the speed and current tick of the animation to 0. We support two forms of packet visualization. The first being an edge heatmap, which turns the edges green when there are packets travelling upon that edge, becoming more opaque as more packets travel upon it. This visualization method is the more performant of the two. The second visualizes individual packets as sprites and shows them travelling on the edge.

Figure 7: The anim tab

The anim tab on the hand-held menu

Node Properties Viewer

The parameters of any node can be viewed by highlighting it. The most recently highlighted node

will have its properties displayed to the left of the hand-held menu. If the most recently selected node is a cluster node, it will display a list of all the different properties of each node. There is a scroll bar on the right of the page if needed.

Interaction History Menu

The interaction menu allows for undoing and redoing of interactions. Interactions are defined as any action performed by the user on the graph. Some examples include selecting or moving the node, moving the graph, filtering the graph, or changing node properties.

When an interaction is performed, it appears in the menu with its ID in parentheses on the left. If the same interaction is performed multiple times in a row, the interaction will be grouped together in the menu, getting rid of the ID on the left and replacing it with the number of interactions in the group in parentheses on the right. Additionally, a sideways arrow button will appear to the interaction's left.

If this arrow is clicked, it will expand the interaction grouping into the individual sub-interactions. From here, it is possible to undo individual interactions instead of undoing the entire collection of interactions.

To undo an interaction, click the box to the left of the interaction on the menu. This will create a checkmark and highlight it a new color. If there are any other interactions that also change to that color, undoing that interaction will undo all the following interactions of the same color. This is determined based on which nodes are affected by the interaction. **Figure 8** demonstrates how this may look in practice.

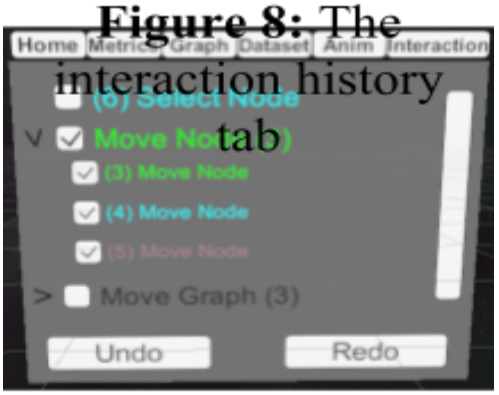


Figure 8: The interaction history tab

The anim tab on the hand-held menu

Clicking the "Undo Interactions" button to the right of the menu will undo any selected interactions as well as their dependencies. Additionally, the undo button can be clicked while no interactions are selected and it will undo the most recent interaction. The redo button will redo the most recently undone action when clicked. The menu can store up to 20 interactions at any given time. After the menu has reached maximum capacity, it will remove the bottom-most interaction.

Filter UI Menu

The filter UI Menu allows nodes to be selected by their specific parameters and parameter values provided in the dataset. The filter UI menu is separate from the hand-held UI menu, it is on a panel near the edge of the environment. Like with the handheld UI menu, the user can interact with the filter UI menu by aiming and clicking with the right controller and right trigger.

On this menu, filters are added by clicking the "+" button. This button is located on the left of the menu (**Figure 9**). This will allow a parameter to be selected using the dropdown. Once a parameter has been picked, a value and any other comparative operator that may be necessary can be selected using the dropdowns that appear next to it and the number pad next to the panel for number entry. This can be done by selecting the "Enter Text…" field and entering

the value using the number pad to the right. **Figure 9** shows what an example of entering a parameter may look like.

There is also a dropdown button which is denoted using this character: ∨. This button is located to the far right of the parameter entry (**Figure 9**). The button will bring up a dropdown containing all the possible values that have been seen for this specific parameter in the data set and will allow the user to select them.

Once parameter constraints have been configured as desired, press the “Highlight” button located under the number pad. This will select any nodes that fit the inputted constraints. Multiple constraints can be added together by adding additional filters using the + button. Additionally, these constraints can either be applied as the union of the constraints or the intersection of these constraints depending on the checkbox selected to the right of the filter menu.

The filter menu can also be used to cluster different groups of nodes. Each node belongs to an overarching hierarchy. For example, one node may be named “Group0.router1.testnic_0”. When selecting a parameter, you will have the option to choose different levels of the

Figure 9: The filtering menu

The Filter UI Menu example with a number parameter and an unselected parameter.

hierarchy (ie. “hierarchy[0]”, “hierarchy[1]”, etc). After selecting a level in the hierarchy, instead of specifying a specific identity to filter on, you can instead press the “Group Unique Parameters”. This will cause the application to cluster based on that hierarchical level. For example, if you grouped based on “hierarchy[0]”, then you will have 1 cluster for each unique name in the hierarchy at level 0 (Group0, Group1, etc.). These clusters will be named after the unique group that they belong to and will be colored automatically to tell them apart from one another.

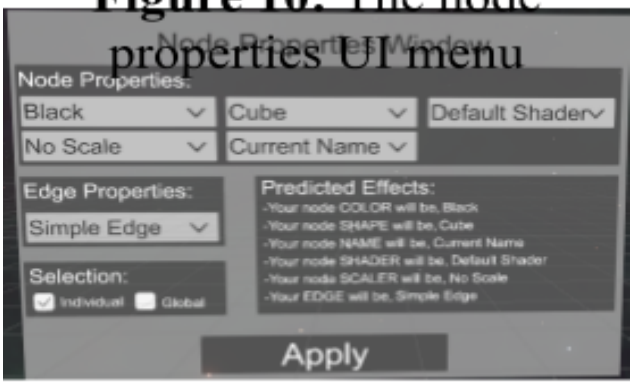
Node Properties UI Menu

The node properties menu (**Figure 10**) allows the properties of selected nodes to be edited. These properties include its color, shape, edge, name, scaling and the shader applied to the node. The currently supported shapes include cubes, spheres, sprite, fire, smoke, and clouds; for shaders, there is normal, wireframe, and steel; for edges there are electric, shudder, glow, and standard. It is located to the left of the filter UI menu. Furthermore, the menu can be utilized to either apply only to nodes you have manually or selected, or to apply the selected properties to all nodes on the graph depending on whether the individual or global checkbox is checked. On the bottom right of the node editor, dynamically updated strings exist to predict the effects upon the graph if the user were to click the apply button.

Visual Effect Nodes/Edges

In the shapes dropdown of the node properties window, as well as in the edges dropdown of the node properties window, a collection of new node/edge varieties can be seen. Many of these options, such as the fire, cloud, smoke, and electric edge all utilize the Unity Visual Effect Graph system to create dynamic and highly customizable particle-based effects. Properties of these graph-created effects can be exposed to allow real time editing based upon data at runtime, such as with the new size/color scaling options of the node properties window. Some of these effects however, such as the glow and shudder line effects utilize the unity line renderer system with modifications such as materials and shaders.

Figure 10: The node properties UI menu

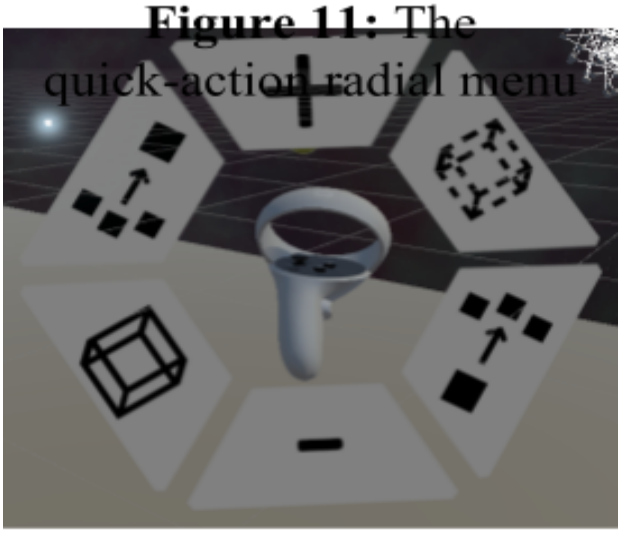


An example of editing a node's properties

Quick Action Radial Menu

Via pressing down the left joystick of the controller, the user can open the new "quick action menu". This menu serves to provide the user rapid and easy access to common functions used to manipulate the graph. Currently, the menu has access to cluster, uncluster, hide, reveal, and scale up/down interaction zone size. Once the menu is open, the user can use the direction of the joystick to select the action they would care to perform, at which point they can use the left controller trigger to select the action. A visual highlighting system also exists to help the user know which action is currently being "aimed at" by the joystick.

Figure 11: The quick-action radial menu



The radial menu with no option selected

Cluster Nodes

On the radial menu, the top left button, “Cluster Nodes” will group all the currently selected nodes into a new “Cluster Node”. This node has all the same behaviour as other nodes; however, it will contain all the edges and packets and properties of the original nodes. To split a cluster node back out into the individual nodes associated with it, highlight the cluster node and press the bottom right button or the “Uncluster Nodes” button.

Remote User Functionality

This section will go over setting up the remote application and how to use it. This feature allows users to connect to the application using the internet. They are able to examine the graph as well as annotate their own notes onto the graph, which will become visible to the local user.

Connecting

To connect to the VR Graph application as a remote user to collaborate with the user using the application in VR, follow these steps:

1. Ensure the VR Graph application is already running on a machine, with someone using the application.
2. Open a web browser (Chrome and Firefox have been verified to work) and open the URL http://147.182.229.16:8888/.
3. Press the play button on the gray stream viewer in the middle of the page.

At this point, a video stream from the VR application will be visible and can be interacted with using the interactions listed below.

Movement

The remote user can move around the application environment to observe the graph visualization and all UI elements present in the VR application. The movement controls are different for each type of supported input device:

- Mouse and Keyboard:
    - To move around, use the WASD keys to move forward, left, backward, and right, respectively. Additionally, the space key can be used to move upward, and the shift key can be used to move downward.
    - To rotate the camera, hold down right click within the video stream, and while holding the button move the mouse.
- Drawing Tablet and Keyboard:
    - To move around, use the WASD keys to move forward, left, backward, and right, respectively. Additionally, the space key can be used to move upward, and the shift key can be used to move downward.
    - To rotate the camera, hold down the side button on the drawing pen, and drag the pen across the drawing tablet.
- Touchscreen:
    - To move around, press and hold on the joystick zone at the bottom left of the screen. This allows you to move forward, backward, left, and right, depending on the finger position in the joystick.

    - To rotate the camera, first make sure that the application is in camera rotate mode by pressing the mode swap button. Once it is, press the finger onto any part of the screen without UI, and drag the finger to rotate the camera.

Highlighting Nodes

The remote user can select a node to highlight it in a color unique to their user. This highlight is visible by the VR user, and any other remote user connected to the application. To highlight a node, the process is slightly different for each input device:

- Mouse and Keyboard:
    - Left click on a node in the scene.
- Drawing Tablet and Keyboard:
    - Tap on a node in the scene with the drawing pen (without holding down the side button).
- Touchscreen:
    - Tap on a node in the scene.

If the node isn't immediately highlighted, try getting closer to the node in the scene using the movement inputs and trying again. To change which node is highlighted, press the "Q" key on the keyboard or the "remote highlight" button in the touchscreen UI to un-highlight the currently highlighted node, then another node can at this point be highlighted in the same manner as before.

Annotations

The remote user can annotate drawings onto the graph visualization. These annotations are attached to the currently highlighted node, so to begin annotation the user must first highlight a node. Once a node is highlighted, instructions for annotating vary based on input device:

- Mouse and Keyboard:
    - Press and hold the left mouse button and move the mouse to draw a line. When done with the current line, release the left mouse button.
- Drawing Tablet and Keyboard:
    - Tap and hold with the drawing pen (without holding down the side button) and move the pen to draw a line. When done with the current line, raise the pen from the tablet.
- Touchscreen:
    - First, make sure the application is in draw mode. If it's not, press the "mode swap" button in the touchscreen UI. Once it is in draw mode, press a finger anywhere on the screen where no UI elements are present, and drag the finger to draw a line. When done with the current line, raise your finger from the screen.

To clear all currently present annotations that have been drawn, press the "C" keyboard button, or the "clear annotations" button in the touchscreen UI.

Mixed Reality Capture

Mixed reality capture can allow you to take photos of yourself inside the application, this can be helpful for applications such as demonstrations. This feature is currently only available on remote builds that run untethered on an Oculus Quest 2.

Camera Placement and Hardware

The first thing you need is a capture area. This area should be large enough for the user to comfortably use VR in, usually an 8x8 square is sufficient. The backdrop for the play area should ideally be a green screen, but any solid color will work. The goal is that the background of the camera capture should be relatively monotone so that the masking performed by OBS results in a clean picture.

You will also need a camera. Most webcam cameras will work, but quality will be somewhat dependent on the camera you use. The example images included in this section were captured using a low-cost Logitech camera. Whichever camera you choose, you should disable autofocusing, if possible, as it will make greenscreen filtering a bit more difficult. The camera should be stationed to have a good view of the user and the capture area. The camera should stay as stationary as possible, as significant movement might require you to recalibrate the camera.

Software and configuration

Next you will need to install a few pieces of software:

- The Mixed Reality Capture tool released by Meta which can be found here.
- The Mixed Reality Capture app on the Oculus device. This can be done through AppLab.
- OBS version 24.0.3 which can be found here (A little over half the way down the page).

You will also need to verify that your computer and the Quest 2 are **on the same network**, and that they are above to stream data over the network.

The next steps are a rephrasing of the documentation provided by Meta, which can be found here.

1. Launch the Mixed Reality Capture tool on your computer and follow the configuration prompts.
2. Start the Mixed Reality Capture app on the device.
3. When prompted by the desktop MRC tool to select a VR device, select Quest 2.
4. Start the Mixed Reality Capture app on the device before pressing “search for device”
    a. For wireless builds “search for device” sometimes does not work. In this case, enter the IP of the device into the provided field. The IP is automatically displayed by the MRC app on the device.
5. Start a new configuration, the tool will have you position the controller at different points in the camera’s view and record its position.

Once you have completed the configuration, it will save it to the device and allow you to download an OBS configuration .json file.

Setting up OBS

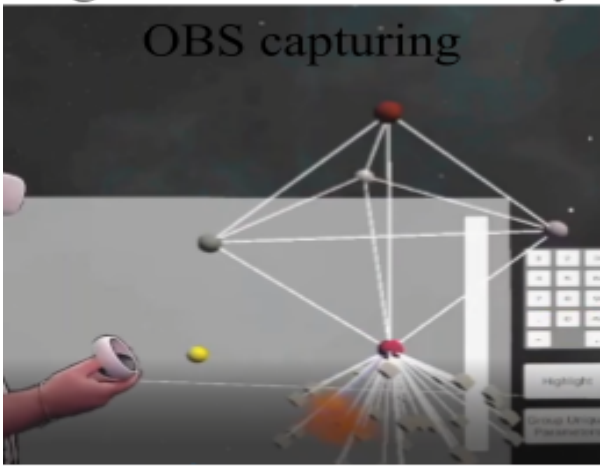
Next you will want to start an instance of OBS and complete the following steps:

1. Select “Scene Collection Menu” and “Import” to import the generated OBS json file from the Mixed Reality Capture Tool
2. At this point you will want to enter in the Chroma Key you are using to filter out the background.
    a. From the “Sources” panel, click “Video Capture Device” in the device layer to make sure your camera is turned on.
    b. After this right click on the “Video Capture Device” and select “filter”.
    c. Here you will be able to tune the Chroma key to fit your background. Once you are happy with this, you are ready to start the capture.
3. Double click the green “background” and enter in the IP address of the headset into the provided field. The IP is automatically displayed by the MRC app on the device.
4. With the MRC app open on the device, click “connect to MRC-enabled game running on Quest”.
    a. You should be able to see in OBS the gray background of the app, raise your controllers to ensure that they are being correctly tracked by the system (as shown in **Figure 12**). If they are not, then it is likely due to incorrect calibration while using the MRC tool. You can update the calibration and re-import the OBS configuration file if this is the case.
5. Disconnect from the Quest by selecting “Disconnect from Quest game”.
6. In the “Sources” panel, drag the “Foreground” component to the bottom of the list.
7. Start the Oculus build of the application. Once the application loads, select “Connect to MRC-enabled game running on Quest” again. The background of the application should now be visible in OBS (as shown in **Figure 13**).
    a. In the headset, you will see an additional blue floating box titled “VR Player” that represents the Camera that OBS is accessing.
8. Click “Start Recording” in OBS to begin capturing video!

Figure 12: Mixed reality controllers

An example of controllers correctly tracking in mixed reality.

Figure 13: Mixed Reality OBS capturing



Example of what Mixed Reality should look like in OBS.

Technical Manual

This section of the document goes into detail on the inner workings of the application and how certain areas of features are implemented. It will discuss how the application initializes and creates the initial graph, how player input and graph interaction is handled, and then it will go over the UI and other advanced features.

Application Initialization

When the application is launched, the StandaloneFileBrowser package opens the specified JSON held in the input folder. The data is then parsed through the GraphBootstrap script into an ExportedGraph SSTV object and converted into an undirected graph QuickGraph object.

The GraphBootstrap script then determines what the selected layout currently is, creates an instance of the respective layout, and calls SetupData() and SetupGraph(). The SetupData function just brings the data over from the GraphBootstrap script to the layout script, the SetupGraph function however handles all the initialization of Unity GameObjects to bring the graph into the application.

The currently supported layouts include Circa, Alt Circo, Two Pi, Random, Semantic, Simple, and Force Directed. These all have their own separate scripts associated with them that inherit from the Graph script. This allows the use of polymorphic principles to call an individualized SetupGraph function for each graph layout. This function instantiates all the SST nodes and edges as node and edge prefabs into the scene with their positions being handled by the layout. The nodes and edges have their own scripts associated with them, SimpleNodeScript (which is a child of NodeScript) and SimpleEdgeScript (which is also a child of EdgeScript). Both of these scripts are initialized by the SetupGraph function. The SetupGraph function tells the NodeScripts which EdgeScripts are attached to it and vice versa for the EdgeScripts. It will also fill a static dictionary called NodeDict in GraphScript with the SSTComponent as the key and the NodeScript instance as its value. This dictionary is widely used throughout this project. At this point all the nodes and edges have been initialized into the scene and the graph is ready to be viewed.

Player Input and Graph Interaction

Player input is handled using Unity's XR Interaction Toolkit package. Unity handles initialization of the headset and controllers, but the controller models are determined in the HandPresence script. The HandPresence script identifies what headset the user is using and looks for the associated model.

User inputs are handled using an event system and functions can be registered to these events so that whenever they're triggered, the functions will be called. Most basic VR actions such as snap turning and teleporting are handled using Unity's own built in VR controls. The rest are custom built for the application specific GameObjects: graph grabbing is handled in the GraphBootstrap script, node grabbing is handled in the XRInteractable script, and UI menu activation is handled in the UI script.

Node Interaction

There is a built-in Unity component that enables grabbing of objects such as our node objects using VR controllers, but they require Rigidbodies. Rigidbodies require a lot of resources because they are handled by the Unity physics system. Instead of using this method, node grabbing is handled in the XRInteractable script where a trigger is used instead of a collider and Rigidbody. To grab the nodes, the node is set as a child of the player's hand and the node's local position is set to the zero vector. Then when the grab action is exited, the script resets it as a child of the graph object.

If the node that is grabbed is selected, it also "manually grabs" all other selected nodes. The node that is physically grabbed is handled just as normal. If it is indirectly grabbed by being selected, then instead of setting the node's local position to the zero vector, it sets it to the difference between the physically grabbed node's new position and original position. This updates all node's position vectors with the same transformation.

For node highlighting and labeling, the Node script has two triggers, a "reveal zone" and an "interact zone". When one of the user's hands enters the reveal zone, it highlights the node's edges yellow and activates the UI name tag for that node. When the user's hand enters the interact zone, the node's edges become highlighted red, and the user can grab the node.

Graph Interaction

Graph grabbing is handled by the StartGrabGraph, GrabGraph, and EndGrabGraph functions in the GraphBootstrap script. The StartGrabGraph first determines whether the left controller is grabbing, the right controller is grabbing, or both at the same time. If it is the left or right hand, the graph is set as a child of the player's hand, allowing it to be moved based on the position and rotation of the player's hand. If it is both, the GrabGraph script determines the distance between the player's two hands, the midpoint of their positions, and the graph's current transform. Then, while the player is grabbing with both hands, the script determines the new position, rotation, and distance of the player's hands and applies that to the graph. If the distance between the player's hands is larger, the graph scales up. If the distance is less, then the graph scales down. Once the player releases with either hand, EndGrabGraph cleans up the variables to reflect the change.

Clustering and Unclustering Nodes

When a node is clustered, the ToCluster function in the NodeConversion script is called. This creates a new node, the cluster node, and initializes it with some basic values. Then, it will loop through each selected node and move its edges to the cluster node, add all the node's keys to the cluster node's keys, and set each SSTComponent's value in the NodeDict to the cluster node.

After this is all done, it will go through and destroy all the original nodes. A similar process is performed when converting from a cluster node to a simple node. It will do the same thing for each SSTComponent, creating a new individual node for each, and randomly dispersing them within a 0.25 meter range from the original cluster node's position.

Filtering System

The filtering system's goal is to create a list of nodes that meet a certain requirement. For this, first a parameter object is created. This allows there to be a list of all possible parameters

to search for nodes by. This list is dynamically generated from the dataset. The parameter object is abstract; its children include the number parameter, the unit parameter, the string parameter, and the array parameter. The number parameter is made for parameters that are expressed using only a numeric value. The unit parameter is very similar; however, it also includes the ability to have a unit attached. A string parameter is a parameter with a value that is a string. Finally, the array parameter is a parameter expressed using an array of doubles.

A constraint is the next type of abstract object necessary for this system. It is derived by the number constraint, string constraint, unit constraint, and array constraint objects. All of those objects have a value to constrain nodes by and functionality to determine if any given node meets the constraint.

The FilterManager script is responsible for the master list containing all the parameters and their possible values that have been seen in the dataset. When the user creates a list of constraints in the UI, it is parsed into a list of constraints in the ConstraintMenu script, then sent to the FilterManager. From there, the FilterManager determines which nodes are affected by these constraints and selects them when the user clicks the “Highlight” button.

In the event of the user clicking the “Group Unique Parameters” button, the FilterManager will first check that there is a parameter selected in the left drop-down, and then create a list of constraints in the ConstriantMenu script for each unique value for that parameter. It will then go through each constraint and cluster and color the nodes selected by that constraint. If there are multiple filters selected, it will only consider the first filter.

Node Properties

Node properties are handled using getters and setters in the NodeScript. Getting these properties returns their values. Setting them will call a function that changes these properties on the node as well. The color and shader properties change the node gameobject’s color and material respectively. When the shape property is updated, the NodeScript calls the OnShapeUpdated function in the Graph Script. Nodes come with a child game object for each supported shape. When the shape changes on the node, the respective child is set as active while all other children are deactivated. Currently nodes can take the following shape: a cube, a sphere, fire, smoke, clouds and a “hidden” node. The hidden node is a node prefab with no activated child object. This allows it to appear “hidden” while saving its properties. While a node is hidden, its original shape is saved in a list of tuples containing a NodeScript and its original shape. This list is named HiddenNodes and is stored in the GraphScript. Whenever nodes are revealed, the script runs through this list and sets all their shapes back to the shape saved in the list. Any nodes that are hidden are also automatically deselected.

Shape options such as fire, smoke, and cloud, as well as the electric edge option all utilize Unity Visual Effect Graph. The effects are disabled and reenabled in the same manner as the other available shapes via a prefab containing multiple visualizations of the same node.

The prefab for a node’s edge is very similar to the prefab for a node itself; each edge has a gameobject holder known as “MultiEdge” with a script known as MultiEdgeScript which is responsible for controlling the enabling/disabling of the various different visualization types of the same edge that live under the MultiEdge gameobject.

Packet Visualization

The packet visualization system is responsible for managing the packet animation that can be played on top of the graph visualization. The visualization system supports multiple visualization styles, through a modular system very similar to graph layouts. There is a visualization where each packet is represented by a small sprite traveling across the graph's edges. The other supported visualization style colors the graph's edges according to the number of packets currently active on that edge.

Parsing

The information on the visualization of the packets and how each one travels through the graph is stored in a text file, similar to how the graph datafile is stored. The datafile is organized by ticks as the unit of time, a tick representing an arbitrary but regular period of time. A packet is sent from a node at a specific tick, and then received by a neighbouring node at a later tick. When the application starts up, this file is loaded and parsed into data structures that ensure all operations including hiding and clustering nodes as well as general playback of the animation are as fast as possible. These data structures include three dictionaries: two dictionaries that store the packets according to the ticks they start and end on, and a dictionary that stores a packet according to which nodes they interact with (originate from and sent to).

Visualization

When the visualization is active (controlled by the handheld UI menu, described in more detail later), every frame the visualization identifies the new current tick given the current speed of the visualization. What happens next depends on the visualization selected. There are standard functions defined for visualization styles, one of them is executed every frame that the visualization progresses.

In the case of the more complicated visualization technique, where each packet is represented by a game object, the behavior is as follows. Every packet that should appear in this frame is identified and instantiated in the scene. These packets can be identified because their initial tick (or end tick, if the visualization is running backwards) is between the previous tick and the current tick. These new packets are added to a list named "ActivePackets", which is a list of every packet currently present in the scene. All the packets in ActivePackets then have their positions updated in the scene, to accurately reflect their progression through the edges. Once a packet reaches its target node, it is destroyed in the scene and removed from the list of active packets.

The other simpler edge coloring visualization technique is much simpler, just updating the base color of each edge according to the number of packets actively present on the edge currently. As packets are sent along an edge, there is a count for that edge that gets incremented, and decremented when those packets reach their destination. For the shudder and glow edge, the packet visualization strategy is highly similar but produces a different visual effect. Instead of edges changing in shade as packets load themselves onto the edge, the width of the line expands to indicate greater traffic on the edge.

Interaction History System

Interactions are defined using the interaction class. It contains the type of interaction it is, whether it's been undone, what other interactions it is dependent on, what nodes it affects, the function necessary to undo it, the arguments to call the function with, and an ID. This ID is its index in the InteractionHistory property of the InteractionManager.

The InteractionManager acts as the means through which the rest of the application can interact with the interactions created by the user. It contains a private list of all interactions ever performed in this instance of the application, a reference to the interaction UI menu, and a reference to the prefab created in the UI menu when an interaction is performed. The interaction manager has methods for creating new interactions, undoing interactions, and getting all dependencies of a given interaction. The rest of the application accesses this script via a static property present in the GraphBootstrapScript.

Interaction History UI Menu

The interaction UI menu contains the interaction prefab, a list of font colors, and list of lists of interactions. The interaction list is a list containing groups of interactions of the same type. Typically, the list will only contain a list of one interaction, but if multiple interactions are performed in a row, it will be all contained in the same list.

Font colors are defined using a class that contains a color and boolean that represents whether it is currently being used. They are used to define dependencies between interactions in the UI. Font colors that are currently being used should not be used again so as to not confuse dependencies, so when an interaction is selected, the script finds font colors that are marked as not currently being used.

When an interaction is performed by the user, the InteractionManager will call a method in the UI script that creates a new interaction prefab in the UI and adds it to its interaction list. When multiple interactions are performed in a row, it is added to the most recent list that was created.

When an interaction is selected, the interaction manager will perform a depth first search on the interaction's dependencies and the interaction UI script will change their font colors to the font color of the selected interaction. Similarly, when the undo button is clicked, the script performs a depth first search on the dependencies, performs them in reverse order, and then undoes the selected interaction.

Graph Saving and Loading

When a graph is saved, the SaveSystem script recreates the initial data used to create the graph using structs. After this, this struct is populated with the original provided data as well as our own data such as node position, node type, or node shape. This structure is saved as "VRG_Graph_Save_(save slot).json" where "(save slot)" is one of the three selectable save slots, 1, 2, or 3. This file is located where all other datasets are located.

When a save slot is loaded. The GraphBootstrap script loads the corresponding file as if it were one of the provided datasets. The dataset parser checks to see if there are any additional parameters such as node position, shape, or color and handles them accordingly.

Most of these extra parameters are trivial, the most notable parameter is the node_type parameter. Currently, there are two supported types, simple and cluster. Simple is handled like normal, however for clusters when saved, it is provided with an id; any SSTComponent within the same cluster is given the same id. When a save slot is loaded, the graph is loaded with all nodes being simple nodes and when it hits a clustered node, it saves it into a "pool" that is linked to its id. On the first frame after the graph is loaded, all of the SSTComponents within these pools are clustered together with the other SSTComponents present in the pool.

Handheld UI Menu

All UI objects within the player menu are children of a master UI menu object that contains the UI script. When the Y button is pressed, the UI script calls a function that activates all its children in the scene.

The UI Ray knows to turn on using a trigger surrounding the UI menu. There is a raycast being shot from the user's right controller at all times. If it hits the trigger while the user is not trying to grab something or teleport, it will turn on. This same system is used for all UI menus.

The Pages System

The pages system is handled by the TabButton and TabGroup scripts. The TabGroup is the grouping of tabs that should be able to access each other in a menu, while the TabButton is for the individual tab UI objects. TabButton starts with "subscribing" to whatever TabGroup is inputted in the tabGroup variable. TabGroup contains a list of TabButtons. When a TabButton subscribes to a TabGroup, the button is added to the list.

In the scene, the TabGroup is held in an empty object which is the parent to all the tab buttons. Directly below it, there is another empty object that is the parent to all the actual UI menus corresponding to the pages. Tab buttons and pages must be ordered the exact same. This is because when the user switches tabs or pages, the TabGroup script searches to see what the button's sibling index is, and then activates that specific page based on that number while deactivating everything else.

Home

The "home" group allows the player to change the graph visualization layout and packet visualization method used in the scene using a drop-down list and an "apply" button. When the "apply" button is pressed, a visualization enum is changed on the graph bootstrap object and the packet visualization manager object, and the entire visualization is reloaded using the ReloadGraph function in GraphBootstrap.

Metrics

When the graph is initially loaded, several calculations are done on the datafile used to calculate useful metrics on the graph. These metrics include: "Order", "Size", "Maximum Size", "Minimum/Maximum/Average Degree", and "Degree Variance". Once these are calculated, the information is listed in this tab.

Graph

The reset graph position button is handled using Unity button events and a function that sets the graph's position to the vector <0, 1, 0> (the "origin" or starting location of the graph). The reset graph scale button sets each node's scale back to the scale it was at the start of the

graph's lifecycle. This is stored by giving each graph a default node scale that it starts with, and then when the graph's scale increases or decreases, so do the nodes.

This tab is also home to the UI responsible for handling save slots. There is a dropdown to select the slot and then two buttons to load and save the selected save slot.

Dataset

This tab contains a drop-down list of all the available graph data files in the default dataset folder location. When the player selects another dataset and clicks the apply button, that file is parsed and loaded, and the visualization is reset using the newly selected file.

Anim

This tab is closely tied to the PacketVisManager. Every frame, it updates the text elements with correct information on the current speed and tick of the visualization. The buttons call functions on an AnimUI script, which identifies the speed that needs to be added or subtracted from the current speed of the visualization and manages the properties on the PacketVisManager object appropriately. The same occurs with the play/pause/reset buttons, when one of the buttons is called it modifies an enum on the PacketVisManager object that triggers the appropriate action to occur on the next frame.

Filter UI Menu

The filter UI menu is a stationary canvas that does not turn on or off. The filter UI Menu is handled by the ConstraintMenu script. At the start the menu only contains a "+" button. When this button is clicked, it adds a master Constraint UI Object prefab that contains all the UI objects that may be needed with all of them deactivated in the scene. A dropdown with all parameters present in the FilterManager is activated. It also activates an "x" button next to the parameter for removing it. When a parameter is selected from the dropdown, depending on what type of parameter it is, it activates all the necessary children of the Constraint UI Object for the user to articulate the filter they want to apply.

If the parameter is of type NumParameter, UnitParameter, or ArrayParameter, it will add a text entry box that can be written into using the numpad located next to the filter UI menu. This part of the UI is handled by the PinPad script. The text entry box is a button that when clicked "selects" it and resets the text in the button. When the buttons are clicked on the numpad, it updates the button's text to include what was clicked. That string is passed to the ConstraintMenu script which parses the string and creates a list of constraints from these values.

Remote User Collaboration

This section will describe the implementation of the remote user capabilities. It will first discuss the structure of this feature, and then it will follow this up with a description of the possible interactions and how they are implemented into Unity.

Services and Libraries

The remote user collaboration feature is built on several key technologies that help make the user experience of using the remote tool simple, and the process of programming the interactions on the application side straightforward. This includes the Unity Render Streaming library, which provides server software, video streaming tools, and input parsing

systems. This library uses the Unity WebRTC library to send the video stream from the application to the remote user, as well as sending the inputs from the remote user back to the application.

Signaling and HTTP Server

The server provides two important functionalities for the feature. The first is that it serves the web application files to any web browser clients that connect. This code is what is run on the remote user's machine to connect to the Unity application, stream the video, and process and send inputs. The second is to act as a signaling server, which is key to establish a WebRTC connection between the remote user and the Unity application. WebRTC is a peer-to-peer protocol, but generally when two peers wish to connect to each other they know the correct address to establish a data channel. A signaling server fixes this, by having each peer connect to a signaling server and broadcast their network addresses and supported types of data channels to all other peers. Once each peer has broadcasted its own address to the other, a direct WebRTC connection can be established, at which point the signaling server's function has completed.

Another important detail with this server is the presence of a TURN server. This server is utilized frequently in WebRTC systems and is used when two peers do not have a direct connection possible between each other. This can be due to network configurations on one or both of the routers each machine is connected to. To work around this, a TURN server is introduced, which both machines can communicate with, and will redirect any WebRTC data as a sort of passthrough. This ensures WebRTC can still be used in scenarios where a direct connection is not possible.

Input Processing

One important feature for the remote users to be able to participate in the Unity application themselves is the ability to send inputs to the Unity application. These inputs allow the application to implement interactions specifically for the remote user, as compared to remote users only being able to view a static stream of the scene. The remote user's web application registers for many input events. When one is detected, a packet is sent to the Unity application containing information on the device that triggered the input, and what that input specifically was. The Unity application receives this packet and processes it in such a way that Unity's built- in input system can handle the input as if it were a device plugged directly into the PC running the Unity application.

The remote collaboration system allows for multiple input devices to be used to control the system. By default, all remote users can control their avatar with mouse and keyboard. If a touchscreen device is detected on the remote user's device, a touchscreen control scheme will be enabled as well. This system interprets touch inputs based on what UI elements they are touching, or if they are just touching the screen outside of UI elements, and triggers rotations, movements, and other actions according to this processing.

Highlighting

When the remote user clicks in the video feed, the position of that mouse click is processed in the Unity application. A raycast is sent from the camera's position according to the position of this click. If the raycast collides with a node, a "highlight" object is placed on that node, and the node is saved for use in other features. This highlight object is translucent and is

colored according to the user's set user color. When the clear highlight button is pressed by the remote user, the highlight object is removed.

Annotation

When a node is highlighted, there is an invisible plane placed on the highlighted node. This plane is used by the annotation feature, to identify the location where the line being drawn should be placed. When the user clicks on the video stream after a node is already highlighted, a raycast is still done but instead of detecting node collisions, it detects the exact point that the ray collides with this invisible plane, and a line segment is drawn from the previous line segment to the new position. When lines are drawn in the scene, the initial position is dictated by the plane, but their positions are attached to the position of the node. This means that if a node is moved, for example by the VR user, the annotations will move with the node. These lines are also colored with the user's set color. Every line drawn by the user is saved in a list, so that when the user pressed the "clear annotations" button, all the lines can be removed.

Settings

The settings class is a static floating class that takes in settings from a JSON and imports it into a dictionary in the class. Currently, the setting class only supports changing the URL that the signaling server uses for remote collaboration.

UML Diagram Accompanying Text

These sections are written to accompany UML Diagrams uploaded alongside this document in a draw.io file. They provide additional context for the design decisions and architecture outlined in these diagrams. The header of each section corresponds with a tab label within that draw.io file.

Class Diagram

There are several notable hierarchies in this graph that provide important details for how the code is structured in the application to ensure modularity and good object-oriented design principles. The first is the GraphScript hierarchy. Organizing all methods of visualizing the graph as children of one parent class simplifies swapping between the visualizations as they all fit a standardized interface. The hierarchy around NodeScript and EdgeScript allows for a similar modularity, having all nodes and all edges inherit from one shared parent allows for different behavior to be built and dynamically switched between in the visualization. The final notable hierarchy is the group built around Constraint and Parameter. These allow the filter UI to support multiple kinds of parameters, from strings of text to numbers to units.

Remote User Protocols

This simple graph demonstrates the protocols used to communicate between each machine present in the remote collaboration feature. The remote browser PC and the PC running the Unity application both use websockets to communicate with the signaling server running on the Cloud VM. There is also a TURN server present, so two PCs that cannot communicate directly with each other can still communicate using WebRTC. Finally, the cloud VM also serves HTML, CSS, and JavaScript files to the web browser on the remote user’s PC, which serves as the client for the remote user to interact with.