Android VirtuTrace Remote (VT Remote)

Design Document 2

Group May14-21

Kollin Burns

Tanner Borglum

Sheil Patel

Alexander Maxwell

Lukas Herrmann

Project Overview

The project that our team has undertaken is to create an Android application for the virtual reality simulator inside of Howe Hall on the Iowa State University campus. The simulator is known as the C6, and the application that compiles and builds the virtual environments and simulations, which are known as scenes, within the C6 is called VirtuTrace. The app that we are designing and implementing will interact with VirtuTrace to allow the user to control various environmental aspects of the a given scene that is running.

For example, the scene might have various objects in the scene, let's say an apple, and our app will allow the user to control properties of the apple, such as color, size, and placement in the scene amongst others. The app will also allow the user to debug the scene and watch objects in the scene to see how they change over time. The user will be using an Android tablet to interact with the Android app, so our team is building a tablet app.

Project Goals

On a high level, these are the the goals that we have defined for this project:

- The app should connect to VirtuTrace through a TCP/IP connection across the Iowa State wireless network.
- The underlying VirtuTrace instance (written in C++) should be able to receive data from the Android app (written in Java) and also should be able to send data back when necessary.
- The Android app should have an easily accessible and understandable user interface that allows the trained C6 technicians to use the app without having extensive underlying knowledge of the implementation of the app.
- The app should allow the user to view all the objects in the scene and allow them to choose which object they want to edit from an expandable list.
- The app should allow the user to "watch" objects and variables of their choosing.
- The app should allow the user to see a console view of the "watched" objects and see what properties of these objects change over time.
- The app should allow the user to manipulate at least the following properties of a given object: color, scale, location, and orientation.
- The app should allow for real time changes to the scene, and following this goal, VirtuTrace should be able to handle real time configuration changes to various object properties.
- The app should allow the user to save changes that they made to objects in the scene so that they can reload the reconfigured scene if necessary.
- The user should be able to get visual feedback of which object they have selected to edit as to not make mistakes in selecting incorrect objects (e.g. a highlighted bounding box).

Deliverables

Our main deliverable will the Android application that will run on an Android tablet. Secondary deliverables will be various documentation on how to use the app from within the C6 and also information about the implementation for the app so future developers can easily understand how to maintain and extend it. Additionally, we are providing some minor additions to the VirtuTrace code base.

System Requirements

- The user should be given a way to edit properties such as color, dimensions/size, placement in the scene, and rotation in the user interface.
- The system must provided the user with a listview or treeview that allows them to select objects in the scene.
- The system should provide visual feedback when a user has selected an object for editing.
- If the user makes changes to an object in the scene, the system must reflect the changes in real time (or with as little lag time as possible).
- The system should allow the user to save the current configuration if the user has made changes to the environment.

Functional Decomposition

VT Remote and VirtuTrace have several large functional units. VT Remote has a network management thread, a fragment to display the VirtuTrace environment structure, and a fragment to display properties of a single item in VirtuTrace. Our modules in VirtuTrace include a network management thread, an action queue, and an action listener.

The VirtuTrace network manager will look for an open port, create a TCP server socket on it, and wait for commands from the client currently connected to it. After it receives a start message macro, it will add the operation to an action queue after receiving the end message macro. The VirtuTrace action listener will wait until VirtuTrace is not rendering, then check for an action in the queue and lock rendering if there is. Once it has the lock, it will complete the action and update the queue front. If an action requires VirtuTrace to send data back, the network manager will wait for it to return the data.

The main interaction with VirtuTrace requires a network manager for VT Remote. This manager will respond to actions in other views and send messages to VirtuTrace. When an action needs to be performed, the manager will send a start macro, the operation, the data (if any), and an end macro through a TCP connection in an ASCII string or binary format depending on connection speed. Both formats use a native Open Scene Graph notation for data representation.

There are three main UI components to generate actions. The tab selector at the top of the screen switches modes; one for editing the scene graph and another for the property map. Both of these views represent the corresponding part of the VirtuTrace environment in a listview. When a user selects a tab, VT Remote queries VirtuTrace for the latest environment state and updates the view. When a user selects an item in the tree view, a second view on the right will query VirtuTrace for the current item property states and show them in the view. If a user changes one of the properties, VT Remote will send the change to the VirtuTrace which will update the simulation in real-time.

System Analysis

Interdependencies

Functionally, VT Remote will require a connection to an instance of VirtuTrace since it is a tool specifically designed to augment a simulation in progress. However, the app is not dependent on VirtuTrace to run. It will essentially idle until it makes a connection.

For VirtuTrace, there are not any dependencies on the VT Remote. It will function as normal regardless whether it is connected to the Android app.

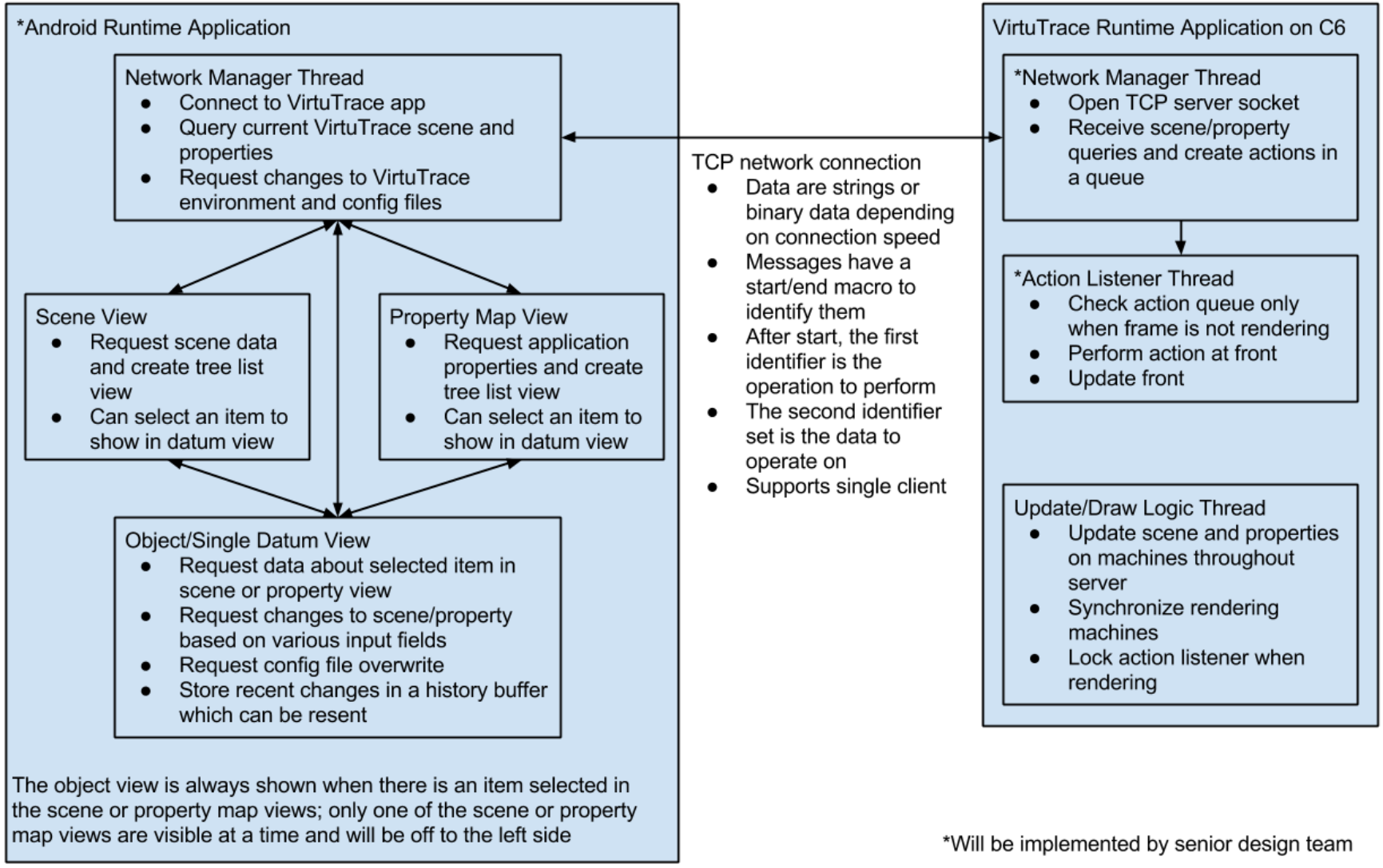
Potential Bottlenecks

The most threatening performance impact will come from the reliance on the wireless network for communication. In an effort to reduce this impact, we will focus on optimizing network packet size and minimizing the frequency of data transmissions.

Security

No sensitive data will be sent between the VT Remote and VirtuTrace. The only types of information that will be transferred are configurable properties of the simulation or objects and the scene objects themselves. Both sides will utilize Open Scene Graph, an open-source library, for serializing and parsing the transmitted data. The library does not encrypt the data as there is no need for it.

System Block Diagram



*Will be implemented by senior design team

I/O Specifications

VT Remote

Being an Android app targeting tablets, VT Remote will utilize the device's built-in touch input manager for handling user input. Based on user input, it may then send data over a TCP/IP wireless connection to VirtuTrace. Additionally, it will receive information, through the same connection, from VirtuTrace for populating the list of scene objects, gathering a list of variables available to "watch," etc.

VirtuTrace

VirtuTrace will send and receive wireless communication to and from VT Remote. It may also communicate with a variety of other I/O devices. However, that communication and those devices will not have any impact on VT Remote.

Interface Specifications

VirtuTrace - VT Remote Communication Interface

Scenes within VirtuTrace are built upon an existing open-source library called Open Scene Graph. It provides the implementation and models for each scene object as well as many useful operations on those objects. VirtuTrace will utilize this library for communication with VT Remote since it has functions for serializing and deserializing the scene objects.

On the other side, VT Remote will use a SWIG (Simplified Wrapper and Interface Generator) Java wrapper library that wraps around Open Scene Graph to provide the same serialization benefits within the Android app.

To keep the interface consistent, we have defined a set of keywords to indicate the beginning and end of a message, as well as a delimiter for easy separation of the elements within the message. Other than the initial handshaking, VT Remote will initiate all communication and VirtuTrace will only respond. The message syntax is shown below.

Field 1 - Start keyword

Field 2 - Operation

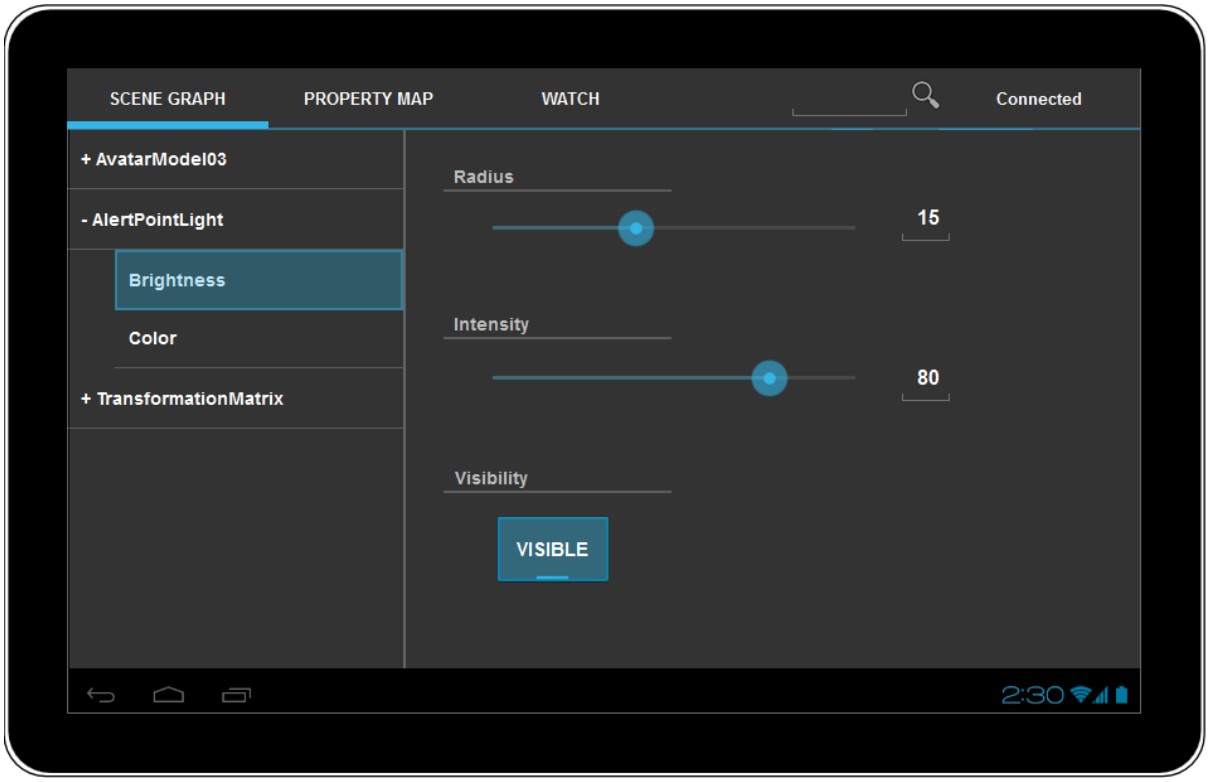
Possible operations are:
- GETS - get data for a specific scene graph node
- SETS - set data for a specific scene graph node
- GETP - get the value of a specific property map field
- SETP - set the value of a specific property map field
- STATUS - get the status of the last operation

Field 3 - Operands
Scene Graph: The first operand is a unique identifier for the specific node. The following operand(s) depend on whether how many properties being affected, which varies depending on the node's data type, and whether they are getting set or retrieved.
Property Map: The first operand is the unique key for the specific property and the second operand is the corresponding value.

Field 4 - End keyword

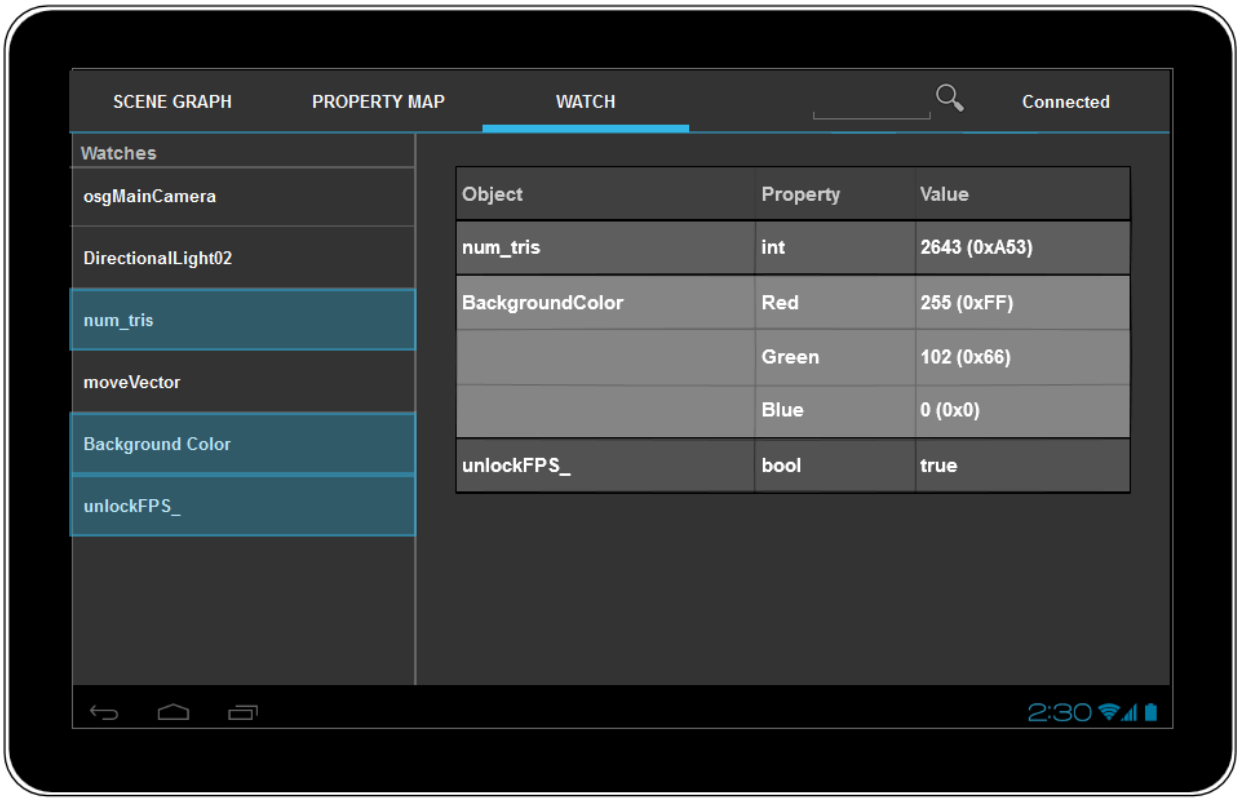
VT Remote - User Interface

The user interface for VT Remote is split into three main tabs: Scene Graph, Property Map, and Watch. The Scene Graph and Property Map tabs are very similar. They both have a listview on the left hand side of the screen that the user is allowed to traverse. Once the user selects an editable object in the list, its modifiable properties are displayed in the screen space to the right.



Scene Graph Tab Concept Rendering

The Watch tab allows the user to select a handful of variables, objects, and properties to observe how they change as the simulation runs.This feature is for mostly for debugging purposes.



Watch Tab Concept Rendering

HW Specifications

- Wireless Networking - 802.11a/b/g/n
- Screen Size/Resolution - at least 7.0” screen measured diagonally with at least 720p (1280x720) resolution

Software Specifications

- Operating System - Android 4.2 or higher

Testing Procedures and Specifications

The testing for our system falls into three main areas: the android application, the VirtuTrace functions, and the testing for the network managed between the two of them. Each area will be tested using unit tests whenever possible to maximize automation as well as manual testing as necessary.

VT Remote

Testing of the android will happen both under forced conditions and live conditions. The forced conditions will guarantee that the android application can request and receive data, display data in desired fashion, and then send changes. After testing under forced conditions, live standalone testing and C6 testing will show that there is all around proper functioning of the application. We also have a tool called UI Automator that allows us to test the dynamic construction of UI elements automatically.

VirtuTrace

Testing of the VirtuTrace functions can be mostly done through unit tests and standalone live testing. The unit tests will make sure that the functions perform as needed for the rest of the application. Live standalone testing makes sure that the functions are behaving as expected under working conditions.

Networking

The network tests fall into two areas. The first area is proper serialization, deserialization, and transfer of data. The second area is speed testing which will be both virtual and live to account for shear volume of data and also actual handling of the C6 network constraints. It is important that our application maintain high speeds for all data volumes.

Implementation Issues and Challenges

The main implementation issues arise from the fact that we are interfacing with an already built system. The vastness of VirtuTrace as an application when compared to our small application means that our application must make the best use of built-in functionality. Though our application derives some strength from VirtuTrace it also runs into problems when desires of our application run contrary to what VirtuTrace can do. When this happens we must either write code that circumvents the problems or add code to VirtuTrace that gives us the desired functionality.

Design Decisions

The project called for us to build an Android app, so this is clearly the chief reason we chose to build the tablet app using Android. A big reason why Android is preferred in this project is because it is open source and based in Java, a language that is very well known at Iowa State and by most developers. This allows for any developers in the future to quickly be able to pick up the project. Another important design decision we made was the use TCP as the protocol for communication between the C++ server and the Java Android app. We chose to do this because TCP has built in

error checking and correction, so if there are any issues with sending data back and forth between the server and client, TCP can resolve most of them without the development team having to change the implementation. One downside to using TCP over UDP is speed. The UDP protocol allows data to be send much quicker over a network than TCP does, but because of the nature of the data we are sending, we felt that having the error checking capabilities of TCP was more useful to for this particular project than using UDP. We may revisit this decision as development progresses and further network stress testing is done, but our current design has clear benefits in using TCP over UDP.