Final Report

Expanse - Virtual Reality Workspace

Group 13
Allen Tung
Andrew Chan
Anthony Wong
Dmitriy Kozorezov
Joshua Chan
Marc Tobago
Yue Yang

05/04/2016
Software Engineering 14:332:452

Table of Contents

Individual Contributions

All team members contributed equally

Summary of Changes

Section 3
Requirement 8, priority 3, The user should be able to specify which applications get rendered into the VR scene, removed

Requirement 10, priority 3, The user should be able to use the system to interact in more depth with specialized applications, e.g., viewing a 3D model when working with CAD software, removed.

Requirement 11, priority 3, The user should be able to view a video player application in a panoramic view, removed.

Requirement 12, priority 1, The system should be low latency (<10 ms), changed to < 50 ms

Requirement 20, priority 4, The system must have a consistent look across all browsers and operating systems,removed.

Requirement 21, priority 3, The system should show a list of all the window feeds being sent to the rendering client,removed.

Requirement 23, priority 1, Windows should appear to be overlayed over real space and be unobstructed, changed to reflect virtual reality rather than augmented reality.

Tracking Device specified to be Kinect

Section 4

UC#4: Visualize 3D design removed. No longer part of our scope

UC#9:Allow for multiple Users removed No longer part of our scope

UC#12, Save/Load removed No longer part of our scope

Use Case diagram modified to reflect changes

Section 6

Large Changes made to Section 6: Domain Model, many functions merged or encompassed by other functions or unnecessary. ViewController and InputController merged with WebClient. FovTracker removed. BMPtoJPG and WindowsAPI merged with WindowCapture.

Concept Definitions
Since we have turned the Server into a publisher-subscriber system, FeedStreamer and GestureCommunicator are now all interpreted as ‘data’ events and there is no longer a specific function needed for each event.

Gesture Library removed; Gestures are no longer compared to a library but based on hand position on screen.

Invariants added to Object Constraint Language Contracts

Section 8

To reflect our changes in our gesture tracking algorithm and the addition of design patterns, major changes were made to the class diagram. Interface specification for gesture tracking has also undergone major changes to reflect the class diagram changes.

Section 11

Updated gesture tracking UI is portrayed in screenshot, highlighting movement away from library-based recognition and towards region-based detection

Test cases

Reflecting our changed use cases and requirements, some test cases have been removed or adjusted, and others added to take their place.

1. Customer Statement of Requirements

A. Problem Statement

We are an architectural company, that often works with a 3d drawing software and a heavy use of various office management tools. Our office is looking to undergo renovation to implement the newest virtual reality technologies to improve our working effectiveness. We made an agreement with Google to use their virtual reality hardware and are now seeking software developers to implement our new hardware with our offices. We would like this software to solve several problems:

In our company, we work with an abundance of programs. Our architects have to manage their 3d (modelling) drawing software, email (applications) software, and word processing software all on one screen. This causes our architects' digital workspace to become very cluttered and difficult to manage. The architects have to constantly go back and forth between their drafting documents and the building specifications. We would like to increase our throughput by simplifying application management.

Currently, a standard employee requires multiple displays to work efficiently, using two to six screens to simply view their modelling software efficiently, not to mention various other communication software. This proposed solution has limitations in both space and funding however, requiring various mounting equipment, new monitors, and increased power consumption from the copious screens in the building. A single multi monitor workspace set up potentially takes the space of two to three standard cubicles, requiring more and more rented office space.

Consulting with business engineers, we have devised a solution to improve our workflow. We would like a system that uses virtual reality that can give our employees more digital "space" to work with. This system must, in real time, display the applications present on the current desktop. The system must also allow for more natural input of commands, and manipulation of the virtual workspace, for example, by using hand motions to manage windows. As there are no currently available productivity tools designed with 3D spaces in mind, we would also like 3D support for existing applications, allowing the software we currently use to remain in use.

We have a limited budget for each employee's workspace, and would like a single solution to maximize efficiency given our monetary constraints. Ideally, the space required for each employee's workspace should also be used as efficiently as possible, to reduce the need for more office space rental. Despite the limited space, we would also like each employee to feel as though they have enough space to work with, and not be dissatisfied with their environment, so as not to reduce productivity. The product we are looking for should be scalable as well, so hiring more employees should not bring up an issue with procuring equipment.

Ideally, the necessary equipment should be easy to obtain, and the product should be simple to maintain and upgrade as necessary. The ever changing landscape of workspaces necessitates a product that is similarly morphable, and modular to meet the changing requirements of the future. We hope to purchase a product which we may simply add features to, without it being clunky or unwieldy. The design of the product should be made with modularity in mind, as it will be maintained regularly.

The product should also support various generic applications, such as AutoCAD, or provide some functionality which would make it easy to use these applications with the full functionality of the workspace. In line with the scalability requirement, we would like this product to have a long life span, which would surpass that of the current modelling programs we use. Thus, support for yet-to-exist programs should be somehow implemented, and the product should be very flexible and easy to understand.

We feel that using this technology would be a great solution for both our company and our employees. We would benefit from having less expenses on the workspaces of our employees, while they themselves would benefit majorly from being able to use this virtual workspace rather than the multi monitor setup they currently have. Some have complained about having to utilize many screens to complete their tasks. With the use of this virtual reality workspace, they would be able to use as much or as little space as they desire, without needing to expand their stations or have to look from monitor to monitor to find what they need. We feel that many industries such as ours would benefit from such tools in more ways than one.

Another important benefit of using these virtual reality headsets, is that we feel there will no longer be a need for cubicles for some of our departments. With these headsets, the individual is pretty much “walled in” with the headset. This also means less expenses for furnishing these cubicles. More desks can fit in the same amount of space, meaning more employees can fit as well. It also frees up our employees to work from wherever they please. Since the headset allows them to use gesture controls to work the desktop, they could be using this at a desk or wherever else they are comfortable and productive. This will allow them to work in comfort and put them in charge of where they want to be, leading to increased moral as well.

2. Glossary of Terms

Virtual Reality (VR) - Virtual Reality is an artificial environment generated by a computer that allows a user to interact with the environment.

Augmented Reality (AR) - Using digital or computer generated information and overlaying it over a real-time environment.

Personal Computer (PC) - a PC is typically a desktop or laptop computer that lets you run word processing applications, music players and browse the internet. It’s a term usually reserved for non-mobile devices, although laptops also fall under this category.

Workspace - In our context it is a collection of digital tools that is used to accomplish a task.

Window - not to confused with the operating system sold by Microsoft, a window is a rectangular, framed or frameless viewing area that is controlled by, and shows information from, an application on your computer.

Window Manager - A piece of system software that controls the placement and appearance of windows within a windowing system in a GUI.

Graphical User Interface (GUI) - A GUI is an interface which allows the user to interact with electronic devices through direct manipulation and interaction with graphical icons. In other words, by interacting with icons on a display, the user is given a way to interact with the machine hardware in an easier way.

Scene - A scene is a 3-dimensional digital VR environment, usually thought of as a variable-sized room that may or may not have objects in it

Render - The act of taking a digital environment (data and data structures) and turning it into graphical figures for display to the user.

Virtual Camera - Usually referred to as just a camera, it is a digital object that controls what the user actually sees (what is rendered) inside a scene.

Degrees of Freedom - The range of motion of a system described by roll, pitch, yaw. The human arm for example has seven degrees of freedom.

Head Mounted Display (HMD) - A device that you wear on your head the displays the VR/AR environment to you

Field of Vision (FOV) - The angle of degrees in a visual field one is able to see. A regular human's viewing angle, including peripheral vision, is 200 degrees

Head Tracking - A process of converting the positional information of one's head into the virtual system

Eye Tracking - A process of converting the positional information of one's eye into the virtual system

Hand Tracking - A process of converting the positional information of one's hand into the virtual system

Gesture Control - A process of converting the gesture information into some instructions.

Latency - The time it takes for the virtual environment to change in response to a user's physical action (i.e. moving his or her head). This can be thought of as lag time.

Simulator Sickness - this is nausea and/or anxiety caused by the brain rejecting an imperfect VR/AR experience. It usually happens when HMD has high latency (lots of lag) and/or the scene itself is visually disconcerting (intensely bright colors, flashing lights, etc.)

Refresh Rate - The frequency at which one's display device updates

Haptics - Tactile feedback the user gets from a system

3. System Requirements (or User Stories)

a. Enumerated Functional Requirements

REQ	PRIORITY	DESCRIPTION
REQ-1	1	The user must be able to view all his or her open windows in the VR scene rendered to the HMD
REQ-2	1	The user must be able to use their hands to drag, drop and resize the virtual windows in the scene
REQ-3	1	The user must be able to see changes in the scene and windows in real-time
REQ-4	2	The user must be able to click and interact inside the VR windows to

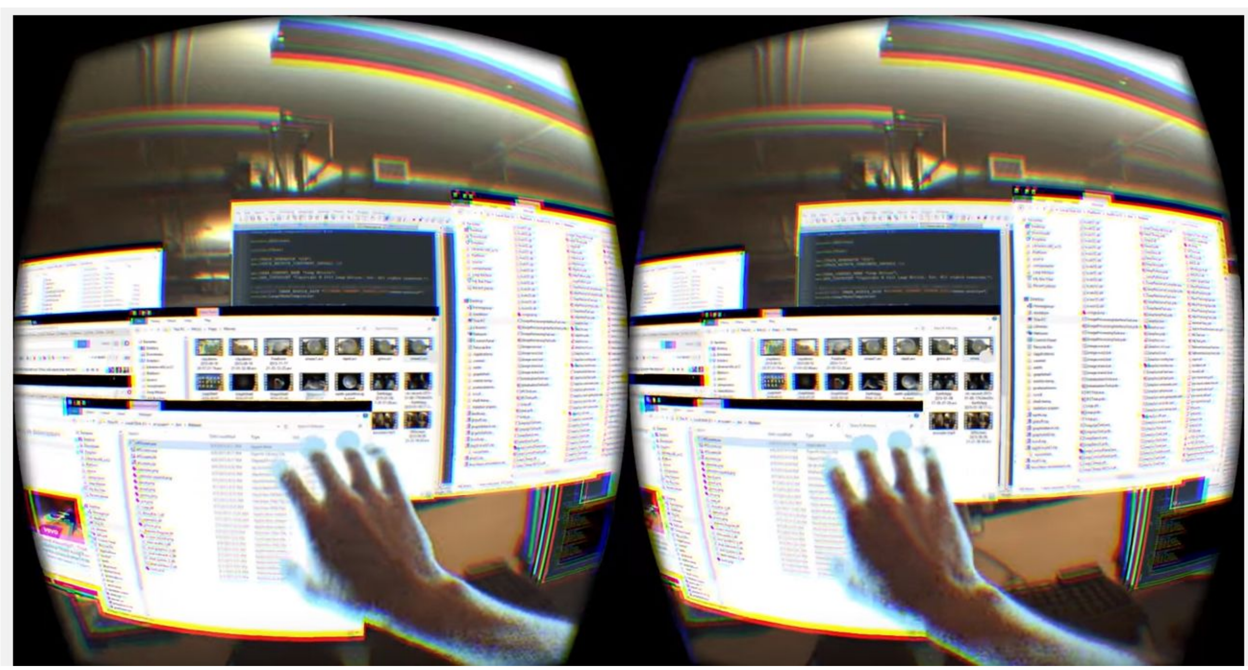
		affect an action in the corresponding desktop application window
REQ-5	2	The user must be able to use his or her phone with the Google Cardboard to act as an HMD
REQ-6	2	The user must be able to stop (close) applications in the VR scene which could optionally also close them on the desktop
REQ-7	3	The user should be able to start applications in the VR scene, which starts them on the desktop
REQ-9	2	The user should able to bring applications on the desktop to focus so it can receive input from the VR scene

b. Enumerated Non-functional Requirements

REQ	PRIORITY	DESCRIPTION
REQ-12	1	The system should be low latency (<50 ms)
REQ-13	1	The system should be divided into 4 major components: virtualization and rendering, window feeds, gesture tracking, and web server
REQ-14	2	The system should be able to recognize hands of different colors
REQ-15	2	The system should be able to recover the desktop state in the case of unexpected system or computer failure
REQ-16	2	The system should be able to run at a consistent refresh rate
REQ-17	1	The system should be able to maintain nearly 1:1 tracking with gesture controls
REQ-18	4	The user can control the brightness of the virtual scene

c. On-Screen Appearance Requirements

REQ	PRIORITY	DESCRIPTION
REQ-22	3	The system must have a consistent look across all screen resolutions
REQ-23	1	Windows should appear to be overlayed over virtual space and be unobstructed
REQ-24	1	Applications in virtual space should appear as they would on a desktop
REQ-25	1	View should be split to create a 3d effect



Similar VR Scene View

4. Functional Requirements Specification

Actors and Goals

Actor	Type	Goal
User	Initiator	Use his/her personal computer in a more dynamic way, with better spatial distribution of windows and special visualization schemes for different applications.
Smart phone	Participator	Serve as the virtual reality screen to be used in conjunction with the Google Cardboard. Also tracks screen position using accelerometer and gyroscope.
Google Cardboard	Participator	Uses special lenses and a cardboard structure in conjunction with phone to form the HMD.
Computer	Participator	Allow for storage and execution of documents and programs used by the phone running the interface.
Server	Initiator/ Participator	Allow user input from the HUD to be reflected on the computer itself. Render open windows in 3D.
Tracking Device (Kinect)	Participator	Capture User's physical movement and transfer as data

Use Cases

i. Casual Descriptions

ID	Name	Description
UC#1	**Display workspace in Virtual Space**	The desktop workspace will be shown in the virtual space and allow the user to interact with it just as on the computer.
UC#2	**Gesture Input**	Through using gestures it allows the user more freedom on where to sit and allows for less hardware to be used(mouse,keyboard).
UC#3	**Specify and adjust Windows**	Allow user to manipulate window size and position, as well as dictate which windows appear, and which do not.
UC#5	**Hardware Input**	Allow user to issue commands to the VR display application with a keyboard and mouse connected to the servicing computer.
UC#6	**Open/Close**	Allow user to start and terminate applications
UC#7	**Adjust Brightness**	Allow user to change the brightness of the display
UC#8	**Enable/Disable Gesture Input**	Toggle gesture tracking
UC#10	**Recalibrate gesture controls**	Allows user to tune the gesture tracking system
UC#11	**Allow Remote Access**	Allow the user to access the workspace from anywhere.

Actor	Type	Goal	Use case name
User	Initiator	View currently open windows on desktop through the HMD	Display (REQ-1)
User	Initiator	Use gestures to drag, drop, and otherwise manipulate windows.	Gesture Input (REQ-2)
User	Initiator	Use hardware peripherals to manipulate windows and issue commands	Hardware Input (REQ-4)
User	Initiator	Open or close windows	Open/Close (REQ-7/6)
User	Initiator	Specify what applications to display on the HMD	Specify windows (REQ-8)
User	Initiator	Adjust brightness	Brightness (REQ-18)
User Interface/ Phone	Participator	Allows the user to request windows to be displayed, and manipulate windows	Display (REQ-1)
User Interface/ Phone	Participator	Matches and responds to user exploring the workspace in real time.	Gesture/Hardware Input (REQ-2/4)
Server	Participator	Render the open windows in 3D and send the data to the phone to display	Display (REQ-1)
Computer	Participator	Perform requested operations, and sends open window data to server.	Display (REQ-1)
Computer	Participator	Reflect changes on desktop, send data to server to render.	Gesture/ Hardware Input (REQ-2/4)
Computer	Participator	Filter only the specified applications to display based on user input	Specify windows (REQ-8)

ii. Use Case Diagram

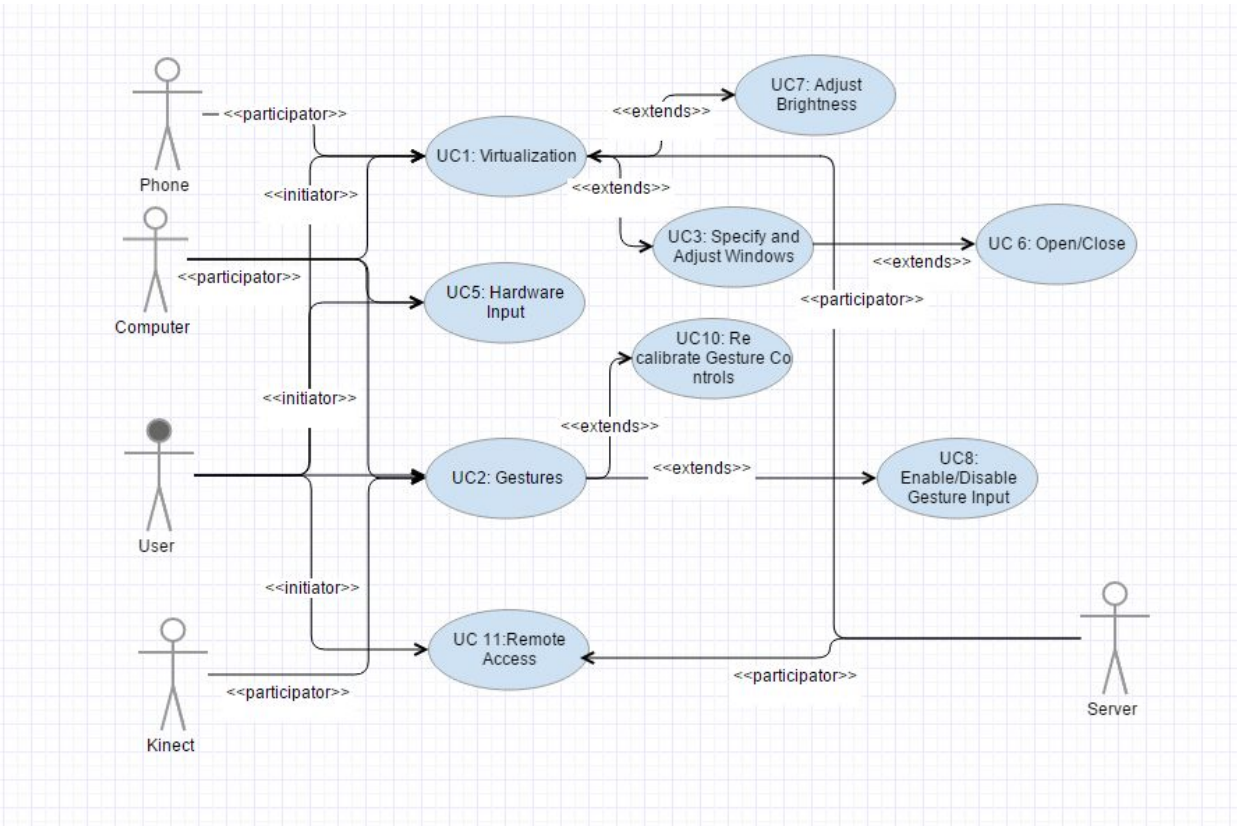


Fig II- Use Case Diagram

iii. Traceability Matrix

	UC-1	UC-2	UC-3	UC-5	UC-6	UC-7	UC-8	UC-10	UC-11
REQ-1	x								
REQ-2		x	x					x	
REQ-3	x		x						
REQ-4	x								
REQ-5	x								
REQ-6					x				
REQ-7					x				
REQ-9			x						
REQ-12	x	x	x						
REQ-13	x	x							
REQ-14		x						x	
REQ-15									
REQ-16	x								
REQ-17		x						x	
REQ-18						x			
REQ-19	x								
REQ-22	x		x						
REQ-23	x		x						
REQ-24	x								

System Sequence Diagrams Located in Section 7: Interaction Diagrams

5. Effort Estimation using Use Case Points

$$UCP = (UAW + UUCW) * TCF * ECF$$

UUCW

Use Case	Name	Weight (Points)
UC-1	Display workspace in Virtual Space	15
UC-2	Gesture Input	15
UC-3	Specify and adjust Windows	10
UC-5	Hardware Input	10
UC-6	Open/Close	5
UC-7	Adjust Brightness	5
UC-8	Enable/Disable Gesture Input	5
UC-10	Recalibrate gesture controls	10
UC-11	Allow Remote Access	10

$$UUCW = 85$$

UAW

Actor	Complexity	Weight
User	Complex	3
Phone	Average	2
Computer	Average	2

Server	Average	2

$$UAW = 9$$

TCF

TCF	Technical Complexity Factor Weight	TCF perceived complexity	TF
Distributed system	2.0	5	10
Response time/performance objectives	1.0	4	4
End-user efficiency	1.0	4	4
Internal processing complexity	1.0	4	4
Code reusability	1.0	2	2
Easy to install	0.5	2	1
Easy to use	0.5	4	2
Portability to other platforms	2.0	4	8
System maintenance	1.0	3	3
Concurrent/parallel processing	1.0	0	0
Security features	1.0	0	0
Access for third parties	1.0	0	0
End user training	1.0	1	1

$$TCF = 0.6 + (TF/100) = .98$$

ECF

ECF	Environmental Complexity Factor Weight	ECF perceived complexity	EF
Familiarity with development process used	1.5	3	4.5
Application experience	0.5	3	1.5
Object-oriented experience of team	1.0	3	3
Lead analyst capability	0.5	0	0
Motivation of the team	1.0	4	4
Stability of requirements	2.0	4	8
Part-time staff	-1.0	0	0
Difficult programming language	-1.0	2	-2

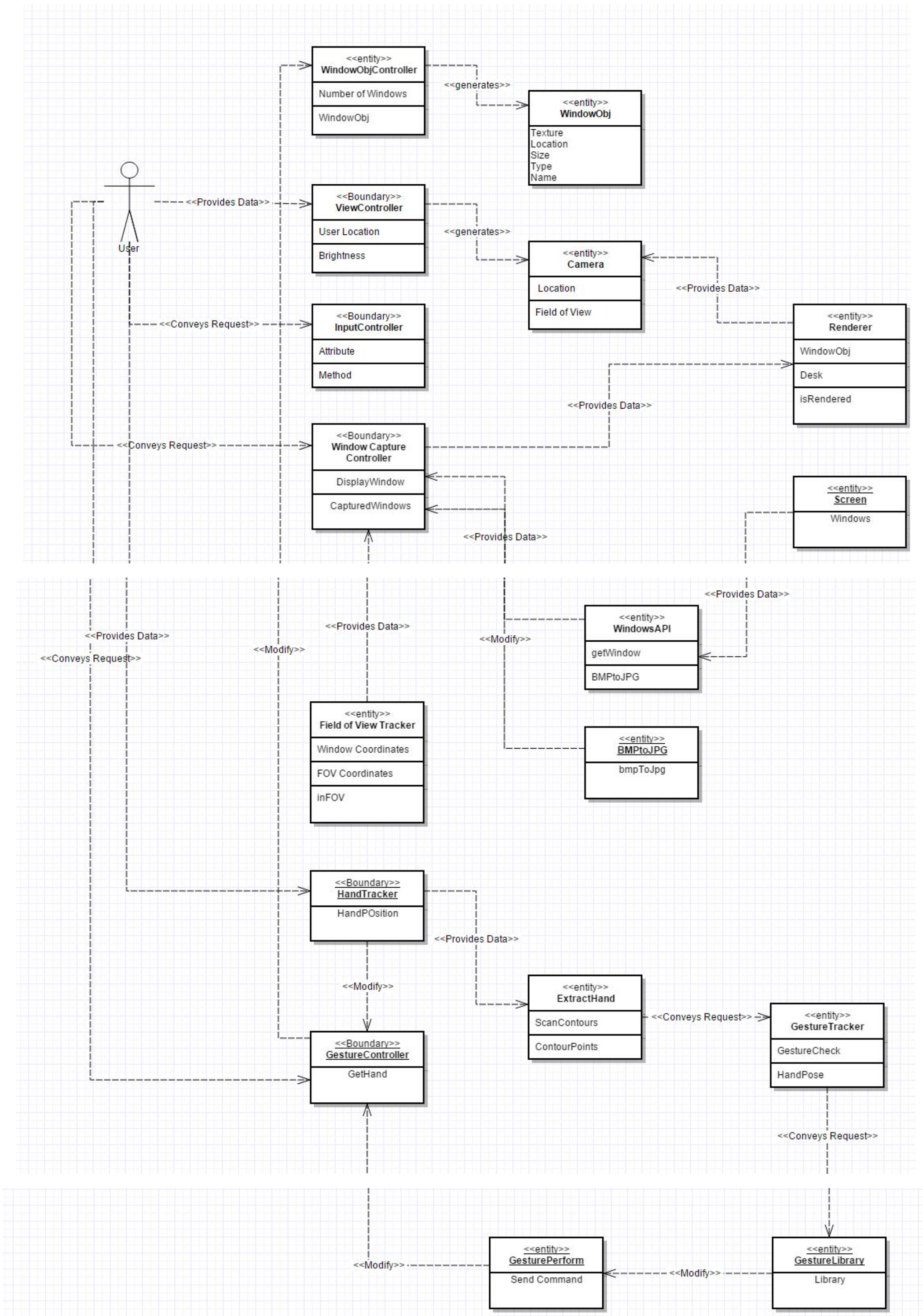
$$ECF = 1.4 + (-0.03 \; x \; EF) = .83$$

Total Estimates

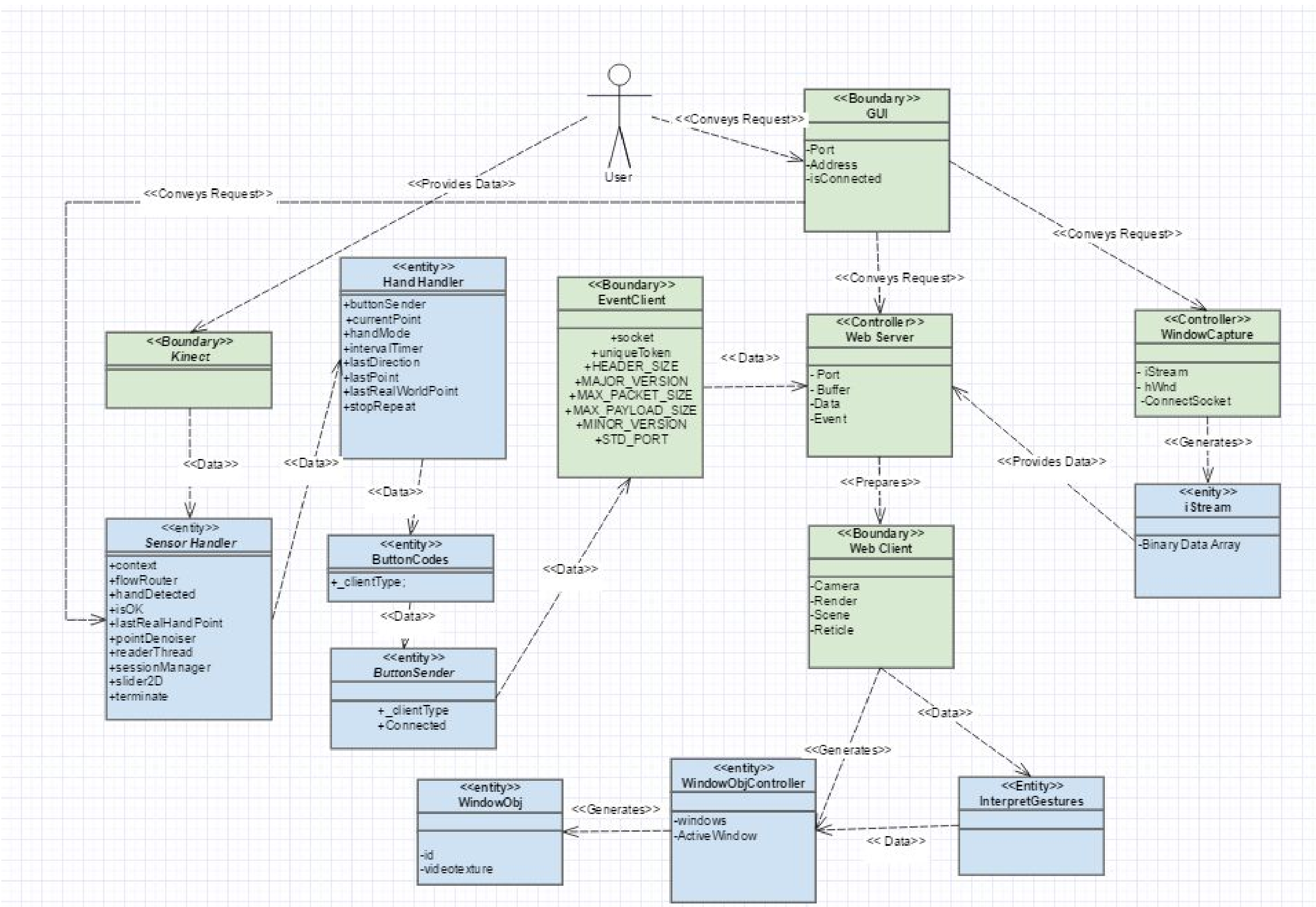
$$UCP = 69.1; \; PF = 28$$
$$Estimated \; Duration = 69.1 * 28 = 1935.9 \; Hours$$

6. Domain Modeling

First Iteration of the Domain Model



Final Iteration of the Domain Model



a. Evolution of Changes:

We can see from our domain model many concepts merged together; this was because we underestimated the power of webgl and three.js for our virtualization subsystem. Many of the concepts such as camera and renderer were automatically handled in the web client. On the other hand, we required a major overhaul of the gesture subsystem because of a change in algorithm. Instead of using a contour-fingertip tracking algorithm we used a regional-hand based tracking interpreted gestures based on the location of the entire hand instead of fingertips. This is reflected in the domain model. No longer is there a need for a gesturelibrary to look up gestures but instead buttoncodes; to send to the server based on the location of the hand. Another critical concept that we did not include in the first domain model was the concept of the web server subsystem. This subsystem was needed for integration of all our subsystems.

A. Domain Model

i. Concept Definitions

Responsibility Description	Type	Concept Name
Virtualization		
RS-1: Shows the window feed and displays window information		WindowObj
RS-2: Coordinates and creates WindowObj; determines size and positioning of windows		WindowObjController
RS-3:Facilitates use cases related to the camera		WebClient
RS-4: Virtual camera that mimics user view in virtual space; sends visual data to user		WebClient
RS-5: Handles user input obtained		Interpret Gestures
RS-6: Renders windows given to it		WebClient
RS-7 Initiates virtualization of windows		GUI
Window Capture (Interface group)		
RS-8: Initiates window capture		GUI
RS-9: Decides which windows to capture		User
RS-10: Captures window textures and sends them to the render server		WindowCapture
RS-12: Facilitates the operation of each controller.		WindowCapture
RS-13: Given data concerning which windows are in the user's field of view, filter out windows that are not necessary to send to the render server.		WindowCapture/iStream
RS-15: Convert the BMP from the windows		WindowCapture/iStream

API to a smaller jpg file.		
Gestures		
RS-16: Initiates gesture control and gesture input		User
RS-17: Manages the gesture control objects and initiates hand acquisition algorithm		Sensor Handler
RS-18: Finds the contours of the hand and detects contour hull and defect points		Hand Handler
RS-19: Observes current hand positioning and signals when hand performs relevant gesture		ButtonCodes/Button Sender
RS-20: Relays the relevant user/computer action associated with a gesture and maintains execution of a single gesture at a time		EventClient
RS-22: Relays current position of hand as a single object in space		Hand Handler
Server		
RS-23: Serves the Webpage		WebServer
RS-24: Relays the window images taken from the Window Feeds subsystem		WebServer
RS-25: Receives remote event clicks from the app running on the client and sends them to the application windows		WebServer
RS-26: Relays hand and gesture data from the Gesture recognition system to the app client running on the HMD		WebServer

ii. Association Definitions

Concept Pair	Association Description	Association Name
User→GUI	User inputs commands to the GUI.	Conveys Request
GUI→WindowCapture	GUI sends a button press message to the capture program, which initiates window capture	Conveys Request
WindowCapture→iStream	The program saves each window feed into an iStream structure in memory, encoding as a jpg	Generates
iStream→WebServer	Each structure is read from memory, and sent over to the server in packets.	Provides Data
GUI→WebServer		Conveys Request
WebServer→WebClient	Sends the WebClient application files on demand	Prepares
EventClient→WebServer	Sends the gesture event details to the web server for parsing and sending to the virtualization clients	Data
WebClient→InterpretGestures	Parses gesture data to locate corresponding action	Data
WebClient→WindowObjController	Uses to generate WIndowObjs with references to the application scene and camera	Generates
InterpretGestures→WindowObjController	Performs an action on the activeWindow based on the gesture event	Data
WindowObjController→WindowObj	Creates the WindowObjs	Generates
User→Kinect	User positions hand in front of the Kinect.	Provides Data
GUI→SensorHandler	GUI reflects results of SensorHandler chain	Conveys Request

Kinect→SensorHandler	Kinect sends information feeds to SensorHandler	Data
SensorHandler→HandHandler	SensorHandler passes hand events to HandHandler	Data
HandHandler→ButtonCodes	HandHandler passes hand position to ButtonCodes, which designates a current region	Data
ButtonCodes→ButtonSender	Current region detected by ButtonCodes finds matching action sequence in ButtonSender	Data
ButtonSender→EventClient	ButtonSender sends combined ButtonCodes and resulting actions to EventClient	Data

iii. Attribute Definitions

Concept	Attributes	Description
GUI	Port	The network port to which the capture program will connect
	Address	The network address to which the capture program will connect
	isConnected	A Boolean value indicating connection status
WindowCapture	iStream	A memory structure to hold the encoded images
	hWnd	Unique identifiers for each window
iStream	Binary Data Array	The image encoded in a JPG format, in bytes.
Web Server	Port	The port(s) of the webserver to send data to
	Buffer	The container for the data handled by the web server
	Data	The data handled by the web server
	Event	Events that are received and sent by the server

Web Client	Camera	A camera object to control the viewpoint in the scene
	Render	An object that controls what is displayed from the scene
	Scene	A container for the 3-d object data
	Reticle	An object that selects and deselects the activeWindow
WindowObjController	windows	A hashmap of the windows that currently exist
	ActiveWindow	The id of the currently selected window (or null)
WindowObj	id	A unique identifier for windowObjs
EventClient	socket	Socket for passing events to local machine
	uniqueToken	Token representing current event
	HEADER_SIZE	Constant value for space left for definition of key fields
	MAJOR_VERSION	Constant int needed for header defs
	MAX_PACKET_SIZE	Constant value for maximum bytes of information sent
	MAX_PAYLOAD_SIZE	Constant value for maximum bytes of acting information
	MINOR_VERSION	Constant int needed for header defs
	STD_PORT	Constant value for intended local port
HandHandler	buttonSender	Passes region of hand to EventClient
	currentPoint	Current location of hand
	handMode	Verifies hand is currently being tracked
	intervalTimer	Value of how often hand region is updated
	lastDirection	Last region location of hand
	lastPoint	Last location point of hand used to define

		region
	lastRealWorldPoint	Last consolidated point of hand
	stopRepeat	Used to stop repeating signals conflicting
ButtonCodes	_clientType	Verifies client is currently active
ButtonSender	_clientType	Verifies client is currently active
	Connected	Verifies connection to EventClient
SensorHandler	context	Establishes initial sensor signal
	flowRouter	Establishes relay tracking of hand
	handDetected	Verifies hand is being tracked
	isOK	Verifies program activity is normal
	lastRealHandPoint	Last consolidated point of hand
	pointDenoiser	Cleans lastRealWorldPoint
	readerThread	Thread set aside to track hand
	sessionManager	Manages application while active
	slider2D	Defines regions of response
	terminate	Safe exit from use

iv. Traceability Matrix

Use Cases	**Name**	**Domain Concepts**						
		User	Window ObjController	Window Obj	ViewController	Camera	Renderer	InputController
UC-1	Display workspace in Virtual Space	x	x	x	x	x	x	
UC-2	Gesture Input							x
UC-3	Specify and adjust Windows		x	x				
UC-4	Visualizing 3D models		x	x	x	x	x	
UC-5	Hardware Input							x
UC-6	Open/Close		x	x			x	
UC-7	Adjust Brightness				x			
UC-8	Enable/Disable Gesture Input							x
UC-9	Allow Access for Multiple Users							
UC-10	Recalibrate gesture controls							
UC-11	Allow Remote Access							
UC-12	Save/Load							

Use Cases		Domain Concepts				
		WindowCaptureController	Screen	Windows API	BMP to JPG	FOV Tracker
UC-1	Display workspace in Virtual Space	x	x	x	x	x
UC-2	Gesture Input			x		
UC-3	Specify and adjust Windows	x	x		x	x
UC-4	Visualizing 3D models	x	x			x
UC-5	Hardware Input					
UC-6	Open/Close					
UC-7	Adjust Brightness					
UC-8	Enable/Disable Gesture Input					
UC-9	Allow Access for Multiple Users					
UC-10	Recalibrate gesture controls					
UC-11	Allow Remote Access					
UC-12	Save/Load	x		x		

Use Cases		Domain Concepts					
		Gesture Controller	ExtractHand	Gesture Tracker	Gesture Perform	GestureLibrary	HandTracker
UC-1	Display workspace in Virtual Space						
UC-2	Gesture Input	x	x	x	x	x	x
UC-3	Specify and adjust Windows						
UC-4	Visualizing 3D models						
UC-5	Hardware Input						
UC-6	Open/Close						
UC-7	Adjust Brightness						
UC-8	Enable/Disable Gesture Input	x					
UC-9	Allow Access for Multiple Users						
UC-10	Recalibrate gesture controls	x	x				x
UC-11	Allow Remote Access						
UC-12	Save/Load						

B. System Operation Contracts

Name	Visualizing Virtual Windows
Preconditions	1. Computer must be running window capture program 2. Window capture program must be communicating with Server 3. Phone must be communicating with Server 4. Phone must send a request to Computer
Postconditions	1. Workspace is displayed/updated in virtual space 2. WindowObj instances update, if necessary 3. NumberOfWindows updates, if necessary

Name	Gesture Tracking
Preconditions	1. User has compatible gesture tracking equipment connected to the system 2. User performs movement/gesture recognizable by the system 3. SendCommand is not currently in action 4. GestureController ceases hand acquisition
Postconditions	1. GestureController updates all gesture objects 2. GestureController resumes hand acquisition

Name	Specify certain windows to display.
Preconditions	1. There must be valid input, either from a mouse or from tracked hand gesture controls 2. Requisites for Visualizing Virtual Windows must be fulfilled 3. NumberOfWindows >= 1
Invariants	1. There must be one user. 2. There must be one server.
Postconditions	1. User specified windows are rendered and displayed on phone. 2. isRendered is updated for all WindowObj instances

C. Mathematical Model

Virtualization

In virtualization, we are creating an algorithm to always position a window perpendicular to a user (in actuality, the camera view-center) when it's being re-positioned. We have the camera's position as a point, as well as the window's original and current position. Considering movement in one direction, either horizontal or vertical, we treat a window's position as following the circumference of a circle, with the position of the camera as a fixed point and the center of the circle.

$$\begin{gathered} Let: \\ P_C = position\ of\ camera\ =\ (x_C, y_C, z_C) \\ P_i = original\ position\ of\ window(center\ point) =\ (x_i, y_i, z_i) \\ P_f = current\ position\ of\ window(center\ point) =\ (x_f, y_f, z_f) \end{gathered}$$

We can derive two key items from those definitions - a **vector from the camera to the original position**, and a **vector from the camera to the current position**.

$$\begin{gathered} Let: \\ \overline{A} = vector\ from\ camera\ to\ original\ window\ position \\ \overline{A} =\ < x_i - x_C, y_i - y_C, z_i - z_C > \end{gathered}$$

$$\begin{gathered} \overline{B} = vector\ from\ camera\ to\ current\ window\ position \\ \overline{B} =\ < x_f - x_C, y_f - y_C, z_f - z_C > \end{gathered}$$

From there, we can easily calculate the **sweep angle -** the inner angle between the two position vectors, relative to the direction of movement - using the dot product.

$$\begin{gathered} cos(\theta_S)\ = \frac{\overline{A} \bullet \overline{B}}{|A||B|} \\ \theta_S = cos^{-1}\left(\frac{\overline{A} \bullet \overline{B}}{|A||B|}\right) \end{gathered}$$

By the *Corresponding Angles Theorem* the sweep angle is also the **offset angle** we need to add to the window's original angular position.

$$\theta_f = \theta_i + \theta_S$$

During normal operation, however, a user won't just be moving a window in one dimension. More than likely, she will be moving the window in a diagonal manner. This solution is also easy to generalize to a multi-dimensional repositioning by introducing an extra step and projecting the distance vectors (from camera to windows) into their component x, y, and z vectors, treating those one-dimensionally, and updating separately.

$$\overline{A_x} = \ < x_i - x_C, y_i - y_C, z_i - z_C > \ \bullet \ cos(\frac{\bar{A} \bullet <1, 0, 0>}{|A|})$$

Gestures

The most important application of mathematics for gesture tracking will be maintaining an accurate representation and model of the hand in order to properly assess its location and orientation, so as to make sure any gestures performed by the user are registered accurately. To this effect, we can take the captured image of the hand and given background O and compare it against the best computer-generated hypothesis of the hand's current positioning, resulting in a discrepancy measurement E:

$$E(h,O) = D(O,h,C) + \lambda_k \cdot kc(h),$$

where D is

$$D(O, h, C) \ = \frac{\sum min(|od - rd|, dM)}{\sum(os \vee rm) + \varepsilon} + \lambda \ (1 - \frac{2\sum(os \wedge rm)}{\sum(os \wedge rm) + \sum(os \vee rm)});$$

the first term is the clamped depth difference between the observed O and the hypothesis h, and the second term accounts for discrepancies between the color of the hand and the color of the visual tracking model. The formula kc

$$kc(h) = \sum_{p \in Q} -min(\varphi(p,h),0)$$

keeps implausible hand configurations in check by accounting for radian angle differences between fingers across all three pairs of adjacent fingers, excluding the thumb.
λ and λ_k are normalization factors, dM is maximum clamping depth, C is the camera calibration information, and rd and rm are the resulting depth map (detailing the position of the hand in 3D space) and binary map (comparing the projected estimate of the hand location with its actual positioning).

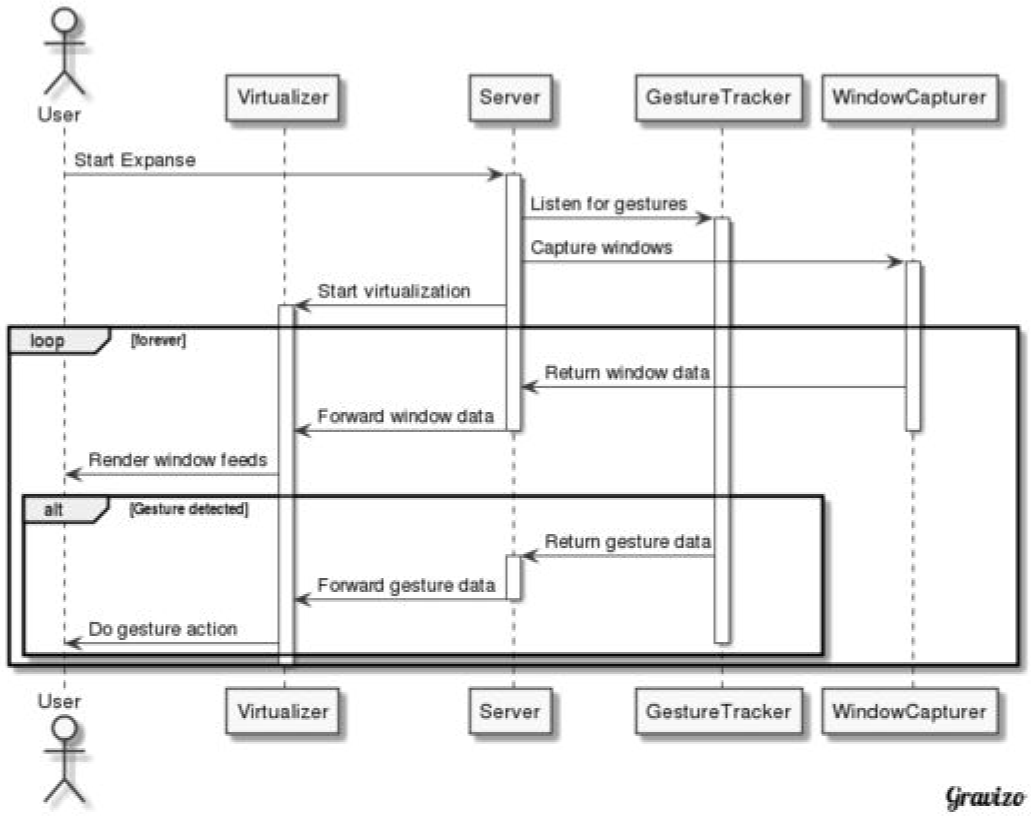
7. Interaction Diagrams

We chose to create two separate programs: One, to capture the window textures from the computer, and two, a server, that will send the window images (and other data) to the client HMD.

Because we have two programs to do the capturing and sending, respectively, our design follows the Expert Doer principle, somewhat akin to the Unix philosophy, in which one component is responsible for, and does, one major thing. Our design also follows the High Cohesion principle, in which the computational effort each program requires is distributed fairly equally.

Having multiple controllers means that the Low Coupling principle is followed; rather than having one central controller, some complexity in the communication network is added to increase the number of components in charge of communication, and thereby spread the load of each component's communication responsibilities across a larger number of components. The alternative would be to merge all the controllers into one central dedicated controller, which simplifies the diagram, but forces the controller to handle a very large amount of communication.

A. UC-1

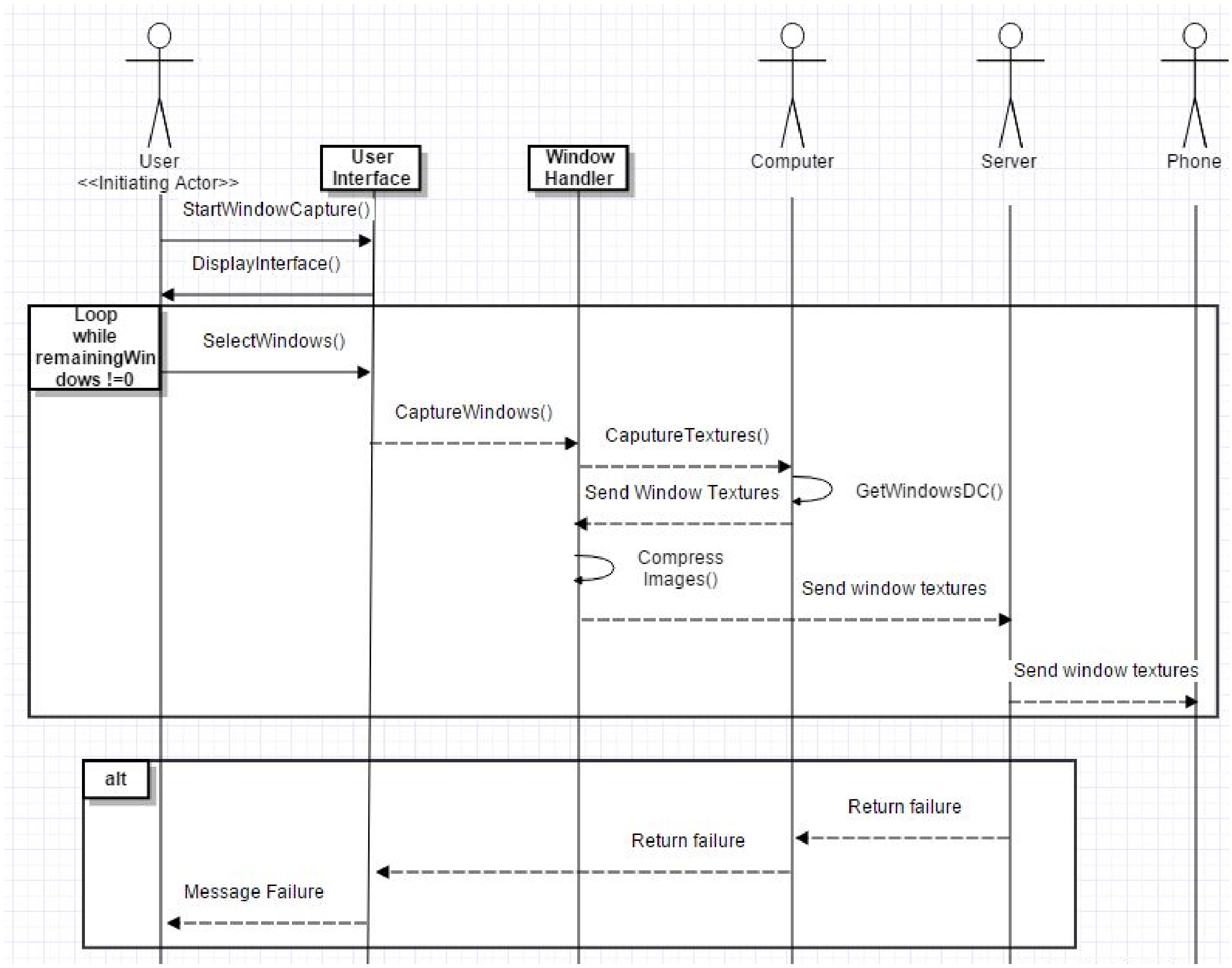


System Sequence Diagram for UC-1

The user starts virtualization by starting the overall program on the host computer. That program, the server, starts up the window capture, gesture tracking, and preps the virtualization subsystems. The user would then navigate to the app on her phone. The program on the computer starts capturing window screens and sending them to the phone, which renders the images in the 3d environment and displays them to the user. Additionally, when a gesture is detected from user the by the gesture tracker, it will notify the server which in turn will notify the virtualization engine to do the corresponding action.

There is a short period of time between the system receiving the initial request and displaying the window, but this should be in the order of a few microseconds. As a fundamental block of our program, this needs to execute flawlessly.

B. UC-1 & UC-3



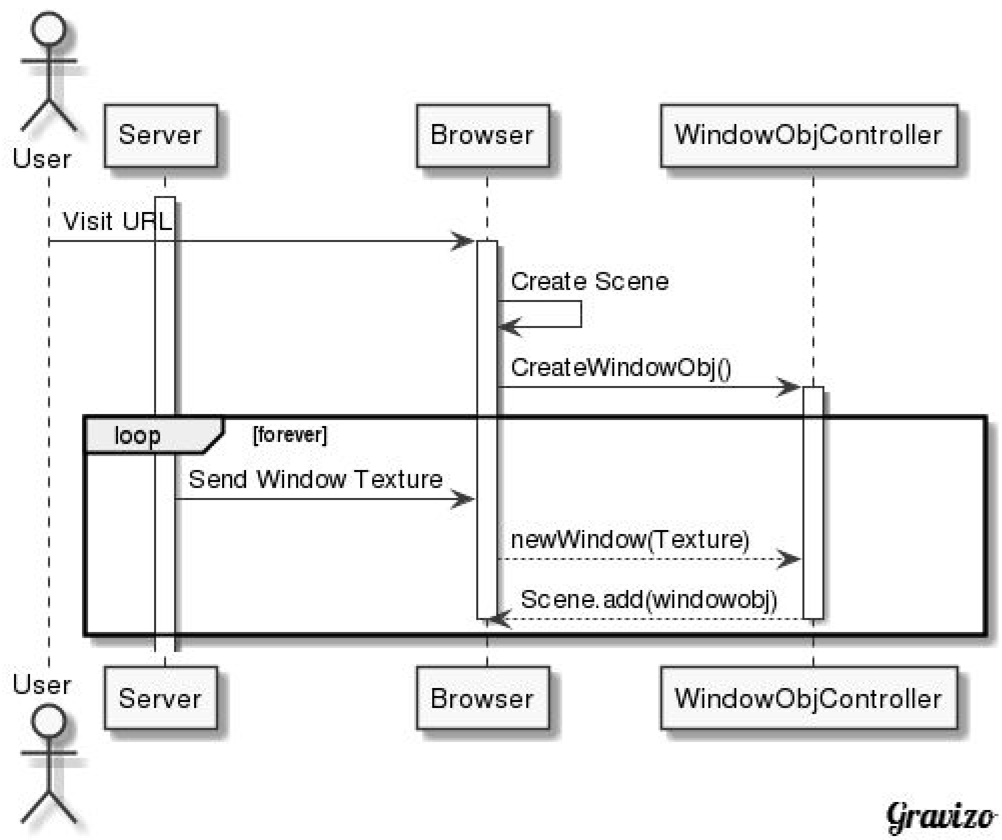
Detailed Sequence diagram part I

It begins with the User requesting to display windows from the desktop, which triggers a sequence of events ultimately leading to the windows hopefully displaying on the phone screen in 3D. At the end of this sequence diagram, the server will have sent all the window textures to the phone, which would process the textures and return a link to the user, which would automatically launch to display the 3D window renders.

As per the GUI that we specified in our RAD for Window Selection, the user has the option to change which windows are captured of all the applications running on the computer. The user would open the settings and navigate to the window that lists all the open programs.

Then, it would check on/off which windows will be grabbed. That command stops the window capture program from capturing those windows via an exception-list or blacklist, and the server will only send off whichever window images are received.

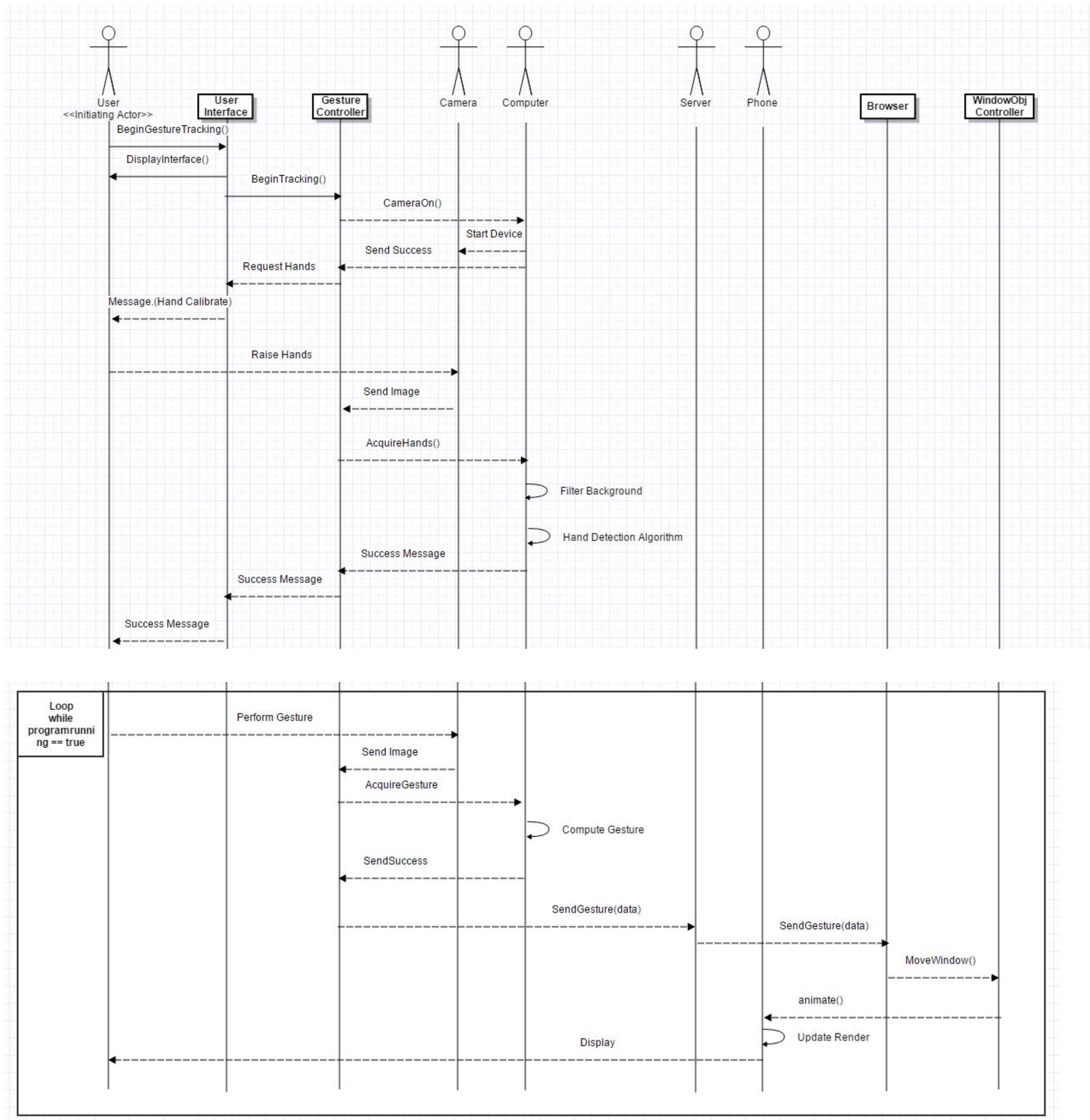
Analyzing this workflow, it's clear that adding this 'exception/selection' functionality does not change the server, as it doesn't need to know explicitly what images to ignore, and will only send what the capture program gives it. This does add extra computational burden to the capture program, but not much - just a block that stops the capture program from capturing all windows. In fact, one can argue that this might even relieve the capture program's computational effort, since it won't have to capture all windows anymore.



Detailed Sequence Diagram part II

The phone, having received the textures as detailed in the previous sequence diagram, will need to render the images in the browser, using webGL. The process by which this is done is detailed here. At the end of this sub case, the window object controller would display the virtual scene to the user, and the phone would simultaneously return a link to the corresponding renders.

C. UC-2



System Sequence Diagram for UC-2

As notable from the diagram, within the scope of the system, displaying changes to the workspace as a result of the user's actions can potentially take a long period of time. However, by maintaining both the High Cohesion and Low Coupling principles, the system should not be hung up at all during the gesture tracking process.

As an example, the tracking of the user's hand for gesture recognition and positioning purposes could have been within one module, but this would create an overreliance on one part of the implementation and would be likely to slow down the response of the system as a whole. In this case, the plan going forward is to create two modules of code, one for each of the duties outlined in the previous example. The diagram outside of the scope of the loop performs the hand tracking, while the diagram within the loop is set to perform any recognizable gestures and update all interfaces accordingly; having these similar ideas run individually of each helps to enforce the cohesion and coupling principles touched on previously.

Responsibility weights: None / Light / Medium / Heavy

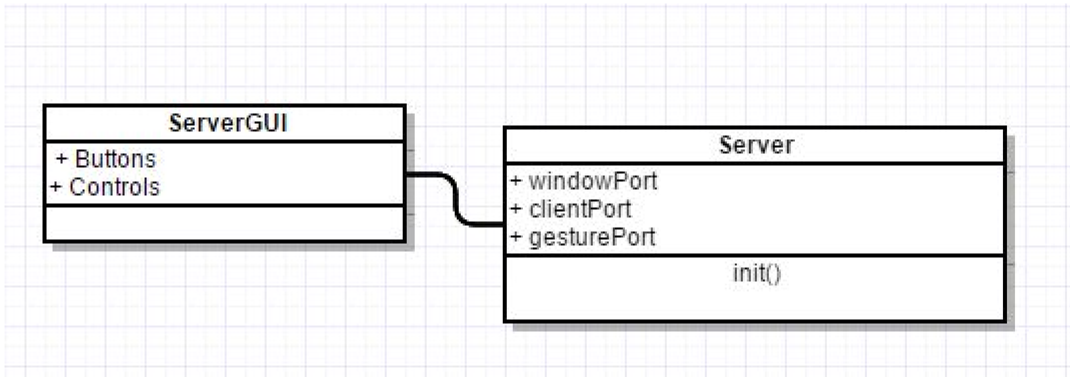
Responsibility Actor	Type 1 Knowing	Type 2 Doing	Type 3 Communicating
WindowObjController	Light	Medium	Medium
GUI	Light	Light	Medium
Interpret Gestures	Light	Medium	Medium
EventClient	Medium	Light	None
ButtonCodes	Medium	None	Heavy
ButtonSender	Light	Heavy	Medium
WindowCapture	Heavy	Light	Medium
Hand Handler	Light	Medium	Heavy
Sensor Handler	Light	Medium	Medium
Web Client	Medium	Medium	Medium
Window Obj	Light	Medium	Light
Web Server	Light	Medium	Heavy

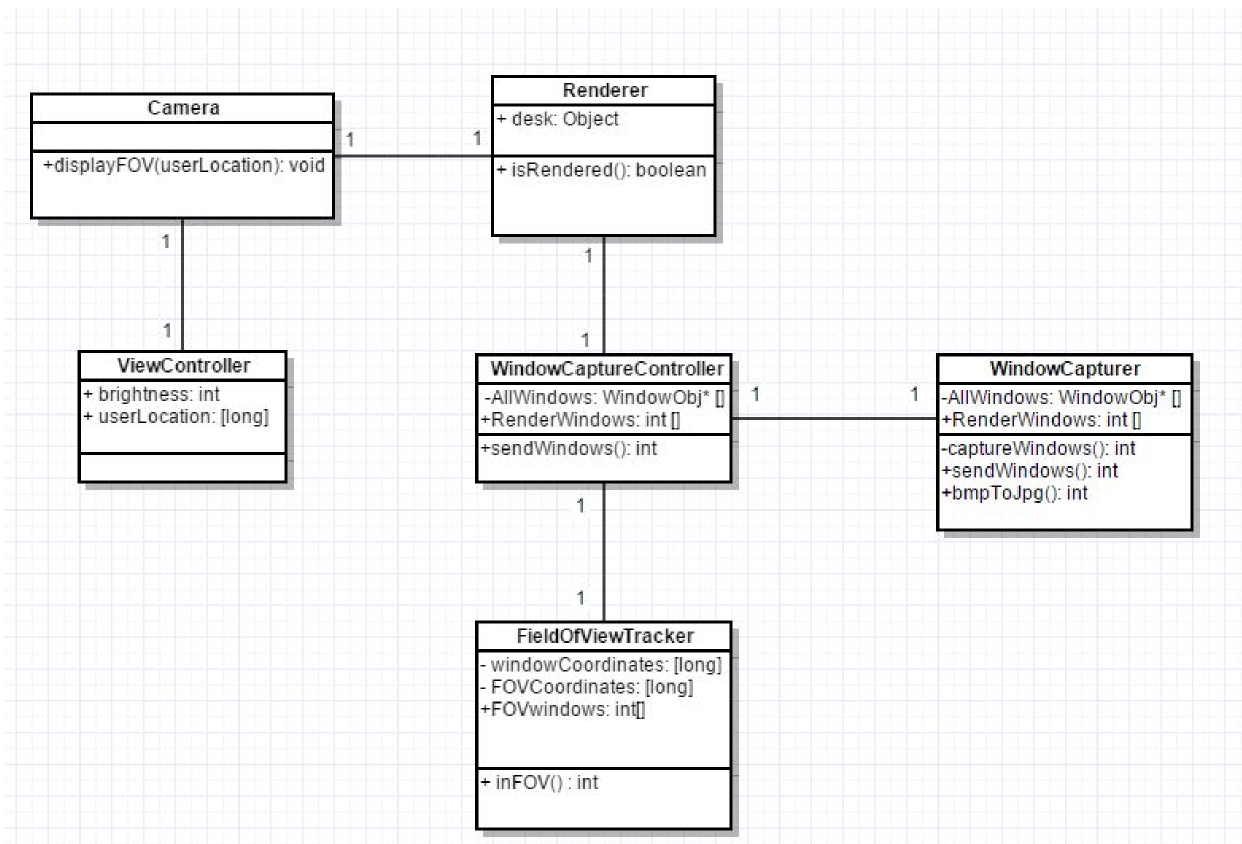
As we can see, the system was designed such that no actor would have two heavy responsibilities, and as much care was taken as possible to ensure that any one actor would not have too heavy of a responsibility. The key actors are those with both heavy and medium responsibilities, like the HandHandler or Sensor Handler, and there was very little that could have been done to reduce their weight since they require heavy computation. In general, work was divided among ‘expert’ components of the system, where each expert component would have a single task to accomplish well. This increases the amount of communication required somewhat, but reduces the burden of each of the first two types of responsibilities on each actor. The WindowCapture for example, only outputs data, and does not receive any data.

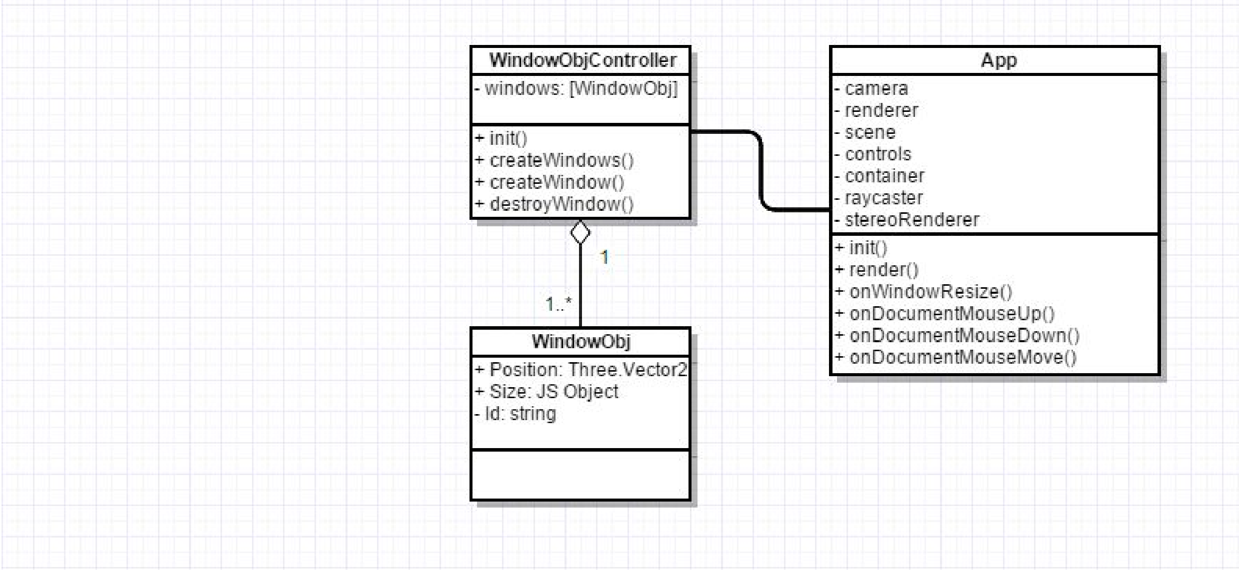
8. Class Diagrams and Interface Specification

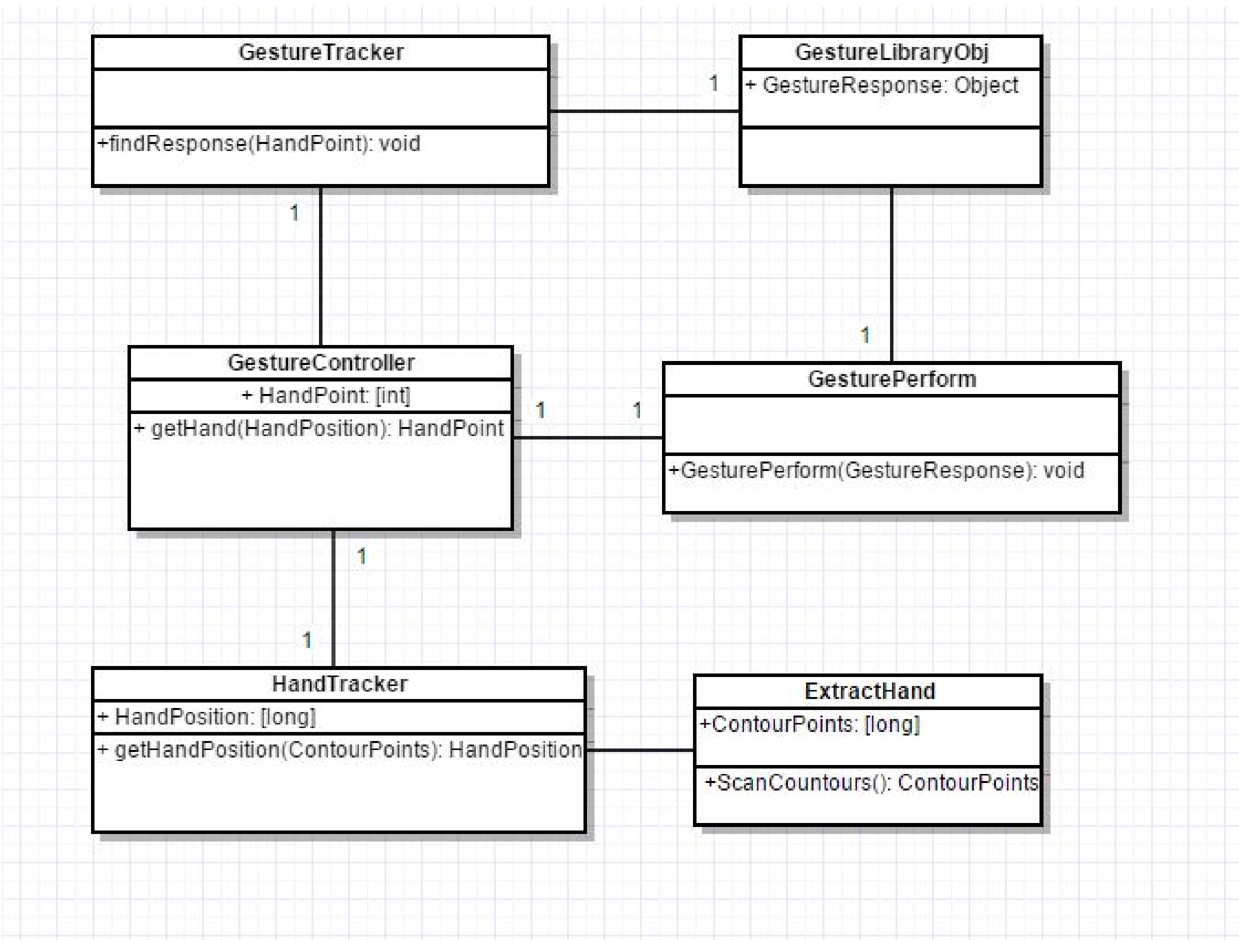
A. Class Diagram

Initial Class Diagram (in White)

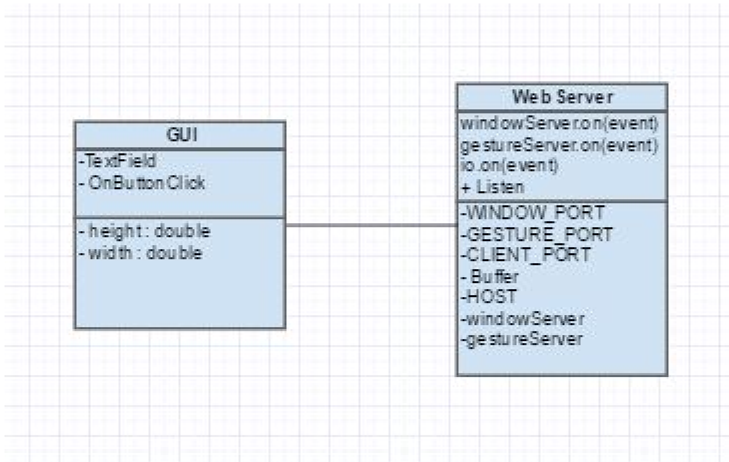


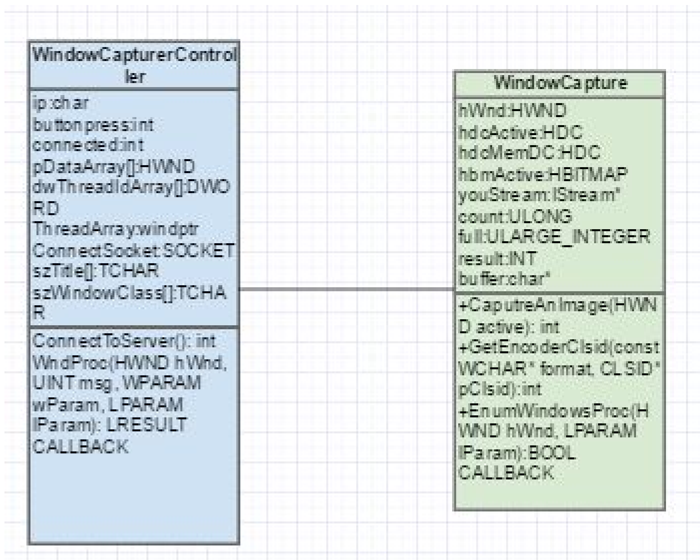


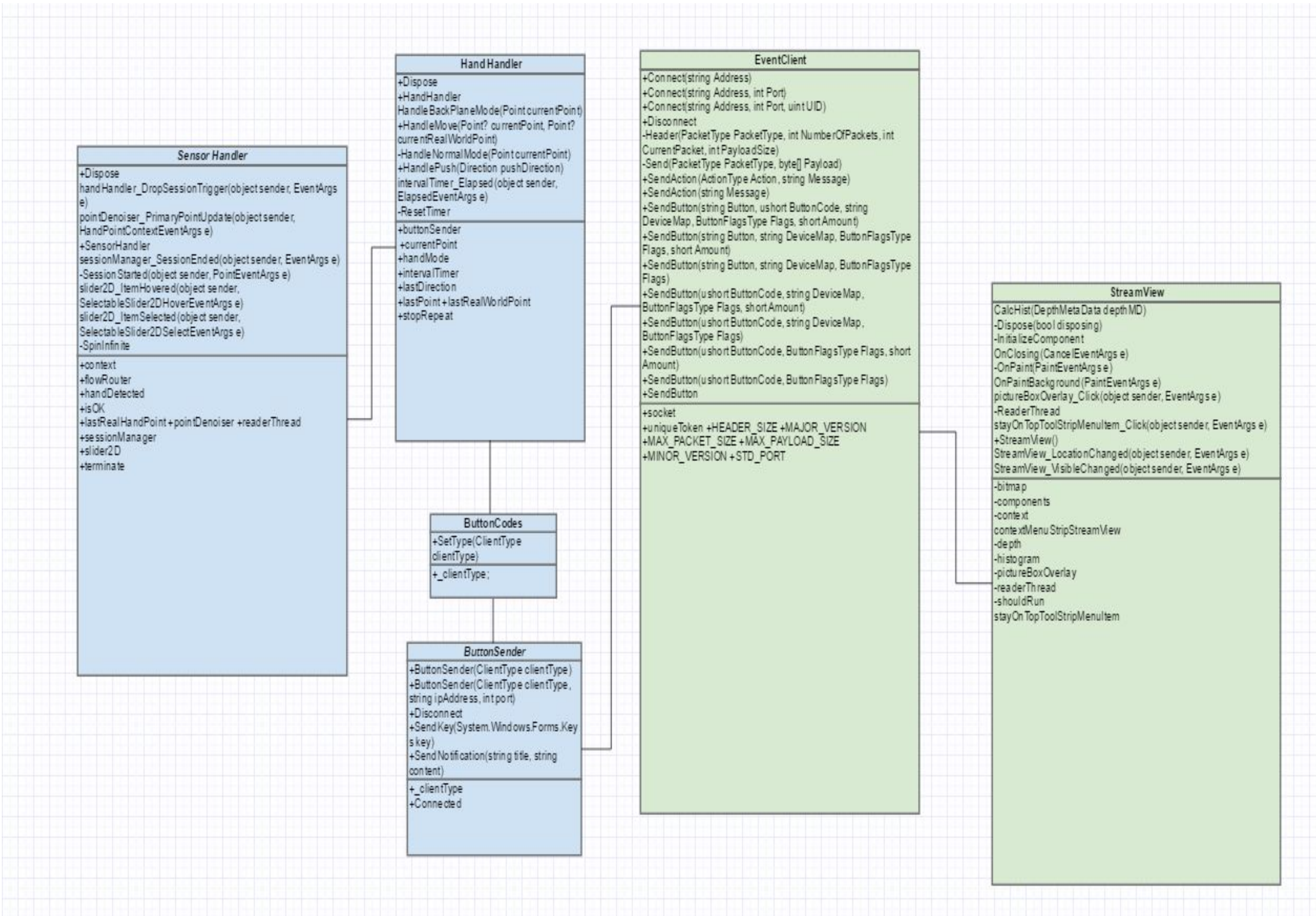

WindowObjController
- windows: [WindowObj]
+ init()
+ createWindows()
+ createWindow()
+ destroyWindow()
1
1..*
WindowObj
+ Position: Three.Vector2
+ Size: JS Object
- Id: string
App
- camera
- renderer
- scene
- controls
- container
- raycaster
- stereoRenderer
+ init()
+ render()
+ onWindowResize()
+ onDocumentMouseUp()
+ onDocumentMouseDown()
+ onDocumentMouseMove()

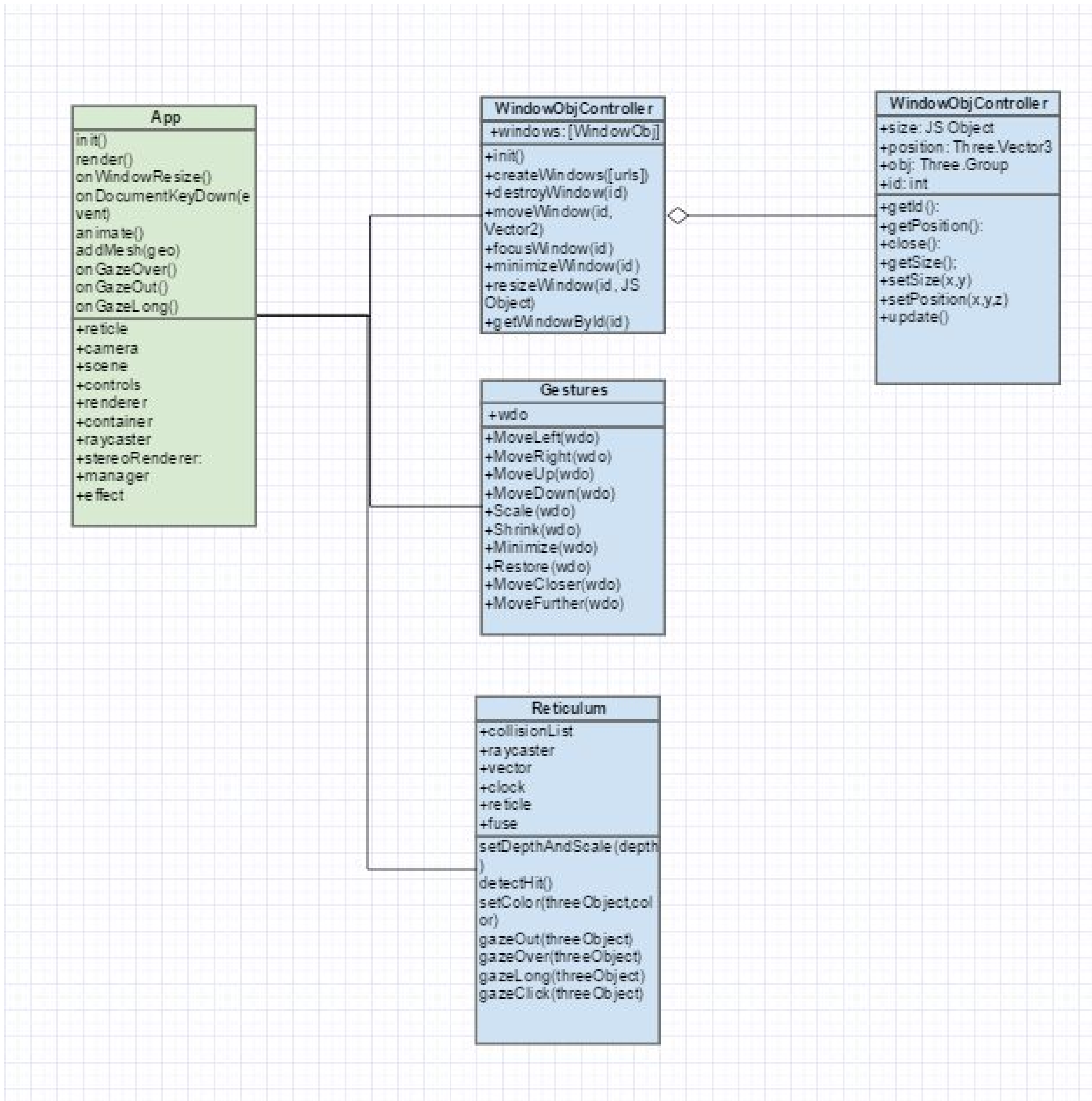

GestureTracker
+findResponse(HandPoint): void
1
GestureLibraryObj
+ GestureResponse: Object
1
1
GestureController
+ HandPoint: [int]
+ getHand(HandPosition): HandPoint
1
1
GesturePerform
+GesturePerform(GestureResponse): void
1
1
HandTracker
+ HandPosition: [long]
+ getHandPosition(ContourPoints): HandPosition
ExtractHand
+ContourPoints: [long]
+ScanCountours(): ContourPoints

Finalized Class Diagram






Sensor Handler
+Dispose
handHandler_DropSessionTrigger(object sender, EventArgs e)
pointDenoiser_PrimaryPointUpdate(object sender, HandPointContextEventArgs e)
+SensorHandler
sessionManager_SessionEnded(object sender, EventArgs e)
-SessionStarted(object sender, PointEventArgs e)
slider2D_ItemHovered(object sender, SelectableSlider2DHoverEventArgs e)
slider2D_ItemSelected(object sender, SelectableSlider2DSelectEventArgs e)
-SpinInfinite
+context
+flowRouter
+handDetected
+isOK
+lastRealHandPoint +pointDenoiser +readerThread
+sessionManager
+slider2D
+terminate
Hand Handler
+Dispose
+HandHandler
HandleBackPlaneMode(Point currentPoint)
+HandleMove(Point? currentPoint, Point? currentRealWorldPoint)
-HandleNormalMode(Point currentPoint)
+HandlePush(Direction pushDirection)
intervalTimer_Elapsed(object sender, ElapsedEventArgs e)
-ResetTimer
+buttonSender
+currentPoint
+handMode
+intervalTimer
+lastDirection
+lastPoint +lastRealWorldPoint
+stopRepeat
ButtonCodes
+SetType(ClientType clientType)
+_clientType;
ButtonSender
+ButtonSender(ClientType clientType)
+ButtonSender(ClientType clientType, string ipAddress, int port)
+Disconnect
+SendKey(System.Windows.Forms.Keys key)
+SendNotification(string title, string content)
+_clientType
+Connected
EventClient
+Connect(string Address)
+Connect(string Address, int Port)
+Connect(string Address, int Port, uint UID)
+Disconnect
-Header(PacketType PacketType, int NumberOfPackets, int CurrentPacket, int PayloadSize)
-Send(PacketType PacketType, byte[] Payload)
+SendAction(ActionType Action, string Message)
+SendAction(string Message)
+SendButton(string Button, ushort ButtonCode, string DeviceMap, ButtonFlagsType Flags, short Amount)
+SendButton(string Button, string DeviceMap, ButtonFlagsType Flags, short Amount)
+SendButton(string Button, string DeviceMap, ButtonFlagsType Flags)
+SendButton(ushort ButtonCode, string DeviceMap, ButtonFlagsType Flags, short Amount)
+SendButton(ushort ButtonCode, string DeviceMap, ButtonFlagsType Flags)
+SendButton(ushort ButtonCode, ButtonFlagsType Flags, short Amount)
+SendButton(ushort ButtonCode, ButtonFlagsType Flags)
+SendButton
+socket
+uniqueToken +HEADER_SIZE +MAJOR_VERSION
+MAX_PACKET_SIZE +MAX_PAYLOAD_SIZE
+MINOR_VERSION +STD_PORT
StreamView
CalcHist(DepthMetaData depthMD)
-Dispose(bool disposing)
-InitializeComponent
OnClosing(CancelEventArgs e)
-OnPaint(PaintEventArgs e)
OnPaintBackground(PaintEventArgs e)
pictureBoxOverlay_Click(object sender, EventArgs e)
-ReaderThread
stayOnTopToolStripMenuItem_Click(object sender, EventArgs e)
+StreamView()
StreamView_LocationChanged(object sender, EventArgs e)
StreamView_VisibleChanged(object sender, EventArgs e)
-bitmap
-components
-context
contextMenuStripStreamView
-depth
-histogram
-pictureBoxOverlay
-readerThread
-shouldRun
stayOnTopToolStripMenuItem



Discussion of changes:

Our finalized class diagram did not undergo major changes for the virtualization and server subsystems. For the WindowCapture subsystem our class diagram and operations were greatly simplified because we discovered that there was no need in having the WindowCapture program attached to the renderer and virtualization methods. The WindowCapture program could operate independently and decoupled from virtualization. Further, it was deemed more efficient that the WindowCapture program not take any input, following a simplized scheme where it would open only one outgoing connection to output streams. We also see in the WindowCapture program specific variables and functions only found in the Windows API. This is because we changed our scope from supporting all OSes to supporting only Window OS. This allowed us to use specialized Window API calls so we could easily capture windows. As for gestures the class diagram changes were a result of changing our gesture tracking algorithm from fingertip tracking to regional hand tracking; this of course changed the methods we needed to use and call.

B. Data Types and Operation Signatures

WindowObj

Description	
The WindowObj class is the object responsible for holding data pertaining to each unique window feed and is designed in such a way that each WindowObj is independent of another.	
Attributes	
+id: string	The unique identifier of each object; used to reference the object
+size: hashmap<string:int>	This is a hashmap with keys ‘width’ and ‘height’ that represent the width and height of the WindowObj
+position: Three.Vector3	The x, y, and z coordinates of the object represented as a vector. The center of the scene is the origin of the vector.
+obj: Three.Group	This is an a class defined by Three.JS that groups multiple 3D objects together. A 3D object is a geometry and a texture/material, where a geometry is a set of points that make up an object and a texture/material is how the space corresponding to a geometry should be styled. A group allows us to have multiple geometry & material pairs - one for the window images, one for a close button and one for a resize handle.
Methods	
+init(): WindowObj	Constructor for the WindowObj object. Instantiates the necessary Three.js objects and our needed attributes.
+getId(): string	Returns id
+getPosition(): Three.Vector3	Returns the position of the windowObj as a Vector3 object.

+close(): null	Destroys the window and collects garbage.
+getSize(): hashmap<string, int>	Returns the width and height of a WindowObj as a hashmap with width and height keys.
+setSize(x:int, y:int): null	Given x,y input resizes the windowobj
+setPosition(x:int, y:int, z:int): null	Sets the x, y, z of the windowobj
+update(imgData:Buffer): null	Updates the image of the WindowObj with the new imgData

WindowObjController

Description	
This class is responsible for managing and referencing all the WindowObj objects in the app. It controls the entire lifecycle of the WindowObj objects and, including the special activeWindow object.	
Attributes	
+windows: hash-map<string, WindowObj>	WindowObjController keeps track of the WindowObjs by keeping them in a hash-map, where specific WindowObjs are accessed by their id.
+activeWindow: Null or WindowObj	A reference to the currently active WindowObj, if there is one.
Methods	
+init(): WindowObjController	Constructor for the WindowObj Object.
+createWindow(): WindowObj	Creates a window
+destroyWindow(id): null	Closes a window with the given id, if it exists. Does not throw an error if the id is not found.
+getWindowById(id): Null or WindowObj	Retrieves a WindowObj with the given id, if it exists. Does not throw error if the id is not found, and instead returns null.

Web Client (App)

Description	
The Web Client (implemented as App in code) serves as the wrapper that controls the lifecycle of the virtualization engine running from the browser on the user's phone.	
Attributes	
+camera: Three.PerspectiveCamera	Camera object defined by Three.JS
+light: Three.HemisphereLight	Global lighting object defined by Three.JS
+scene: Three.Scene	Scene object defined by Three.JS
+controls: Three.Controls	Control object defined by Three.JS
+renderer: Three.Renderer	Render object defined by Three.JS
+stereoRenderer: Three.StereoEffect	Rendering Effect object defined by Three.JS
Methods	
init(): null	Constructs the App object. Initializes internal variables for the app, and sets up event handlers.
update(): null	This renders the entire scene, taking into account the camera, the renderer/effect, and the objects in the scene
onWindowResize(): null	This allows sets the controls for resizing the window through dragging a designated area of the object.
onDocumentMouseMove(event) onDocumentMouseDown(event) onDocumentMouseUp(event)	These functions allow for the scene to be manipulated, for the user to be able to pan the scene and move the objects.

Web Server

Description	
The web server stands as the main communication point between the Virtualization engine, the Gesture Tracker, the Window Capturer, and the GUI. It receives data streamed by the Gesture Tracking subsystem and Window Capture subsystems, and fires events and streams data to the Virtualization engine.	
Attributes	
+gesturePort: int	Port for the gesture subsystem to connect to.
+windowPort: int	Port for the window capture subsystem to connect to.
+clientPort: int	Port for the user's phone to connect to in order to access the application.
Methods	
+init(): null	Constructs and initializes the attributes of the server. Opens the sockets for the ports that are streamed to/accessed
+start(): null	Starts listening for connections and runs the app.
-windowServer.on(event): null	Called when an event (data sent, connected) is made on the window_port. Performs appropriate action based on event.
-gestureServer.on(event): null	Called when an event (data sent, connected) is made on the gesture_port. Performs appropriate action based on event.
-io.on(event): null	Called when an event (data sent, connected) is made on the client_port. Performs appropriate action based on event.

Gestures (Virtualization)

Description	
Performs action based on a WindowObj gesture input.	
Attributes	
+wdo: WindowObj	A WindowObj
Methods	
+MoveLeft(wdo)	Moves window in the x direction by -0.1 units
+MoveRight(wdo)	Moves window in the x direction by +0.1 units
+MoveUp(wdo)	Moves window in the y direction by +0.1 units
+MoveDown(wdo)	Moves window in the y direction by -0.1 units
+Scale(wdo)	Increases window height by 0.02 and width by 0.015. If window height exceeds 6 units, the window is set to 6 units. If window width exceeds 8 units the window is set to 8 units
+Shrink(wdo)	Decreases window height by 0.02 and width by 0.015. If window height exceeds 0.3 units, the window is set to 0.3 units. If window width exceeds 0.4 units the window is set to 0.4 units
+Minimize(wdo)	Window is moved out of user view
+Restore(wdo)	Window is brought back to user view
+MoveCloser(wdo)	Moves window in the z direction by 0.5 units
+MoveFurther(wdo)	Moves window in the z direction by -0.5 units

Reticulum

Description	
Reticulum is an object that controls selection of the active windows	
Attribute	
+collisionList	List of objects that the reticle can interact with
+raycaster	Vector in three.js that calculates collisions
+vector	Vector object stores x,y,z
+clock	Timer
+reticle	3d reticle object
+fuse	Search helper function
Methods	
setDepthAndScale(depth)	Set depth and scale of reticle
detectHit()	If raycaster crosses over a collidable object
setColor(threeObject,color)	Changes the color of the object when called
gazeOut(threeObject)	Action to perform when reticle gazes on the object. Deselects the current WindowObj as the active window.
gazeOver(threeObject)	Action to perform when reticle gaze leaves the object. Selects the WindowObj as the activeWindow.
gazeLong(threeObject)	Action to perform when reticle gaze focuses on an object for more than a few seconds
gazeClick(threeObject)	Helper function not used in our implementation. When a click action is applied when reticle collides with an object

StreamView

Description	
Creates viewable object for user during use	
Attributes	
-bitmap	Writable, updating camera feed image
-components	Required designer variable
-context	Used for contexts needed within a method
-contextMenuStripStreamView	Cleans image in menu changes
-depth	Writable, updating depth feed image
-histogram	Array render of image for detecting hand pixels as they move
-pictureBoxOverlay	Static overlays
-readerThread	Thread for stream view to run in
-shouldRun	Denotes whether or not stream view is active
-stayOnTopToolStripMenuItem	Pins stream view to top
Methods	
-CalcHist(DepthMetaData depthMD)	Calculate histogram pixels
-Dispose(bool disposing)	Clean up resources on exit
-InitializeComponent	Initialize all StreamView components
-OnClosing(CancelEventArgs e)	Runs upon closing program
-OnPaint(PaintEventArgs e)	Runs upon updating view
-OnPaintBackground(PaintEventArgs e)	Runs upon updating background
-pictureBoxOverlay_Click(object sender, EventArgs e)	Creates overlay of user interface

-ReaderThread	Sets histogram pixels to be updatable
-stayOnTopToolStripMenuItem_Click(object sender, EventArgs e)	Pins stream view to top
+StreamView()	First instance of StreamView to run
-StreamView_LocationChanged(object sender, EventArgs e)	Updates StreamView if StreamView is suddenly moved from current location
-StreamView_VisibleChanged(object sender, EventArgs e)	Updates StreamView while StreamView is hidden from view

Event Client

Description	
Cues up events that will update in the stream view	
Attributes	
+socket	Socket for passing events to local machine
+uniqueToken	Token representing current event
+HEADER_SIZE	Constant value for space left for definition of key fields
+MAJOR_VERSION	Constant int needed for header defs
+MAX_PACKET_SIZE	Constant value for maximum bytes of information sent
+MAX_PAYLOAD_SIZE	Constant value for maximum bytes of acting information
+MINOR_VERSION	Constant int needed for header defs
+STD_PORT	Constant value for intended local port
Methods	

+Connect(string Address)	Connect the program to the localhost PC
+Connect(string Address, int Port)	Connect the program to the localhost PC
+Connect(string Address, int Port, uint UID)	Connect the program to the localhost PC
+Disconnect	Disconnect the program from localhost PC
-Header(PacketType PacketType, int NumberOfPackets, int CurrentPacket, int PayloadSize)	Fill in default info required to pass information from program to localhost PC
-Send(PacketType PacketType, byte[] Payload)	Send signals from program to PC
+SendAction(ActionType Action, string Message)	Send signals from program in payload form
+SendAction(string Message)	Initial SendAction method
+SendButton(string Button, ushort ButtonCode, string DeviceMap, ButtonFlagsType Flags, short Amount)	Send button name instead of its matching code
+SendButton(string Button, string DeviceMap, ButtonFlagsType Flags, short Amount)	Sends a button plane down signal
+SendButton(string Button, string DeviceMap, ButtonFlagsType Flags)	Sends a button plane up signal
+SendButton(ushort ButtonCode, string DeviceMap, ButtonFlagsType Flags, short Amount)	Sends current amount (if amount is stored within button)
+SendButton(ushort ButtonCode, string DeviceMap, ButtonFlagsType Flags)	Queues events when a button is triggered
+SendButton(ushort ButtonCode, ButtonFlagsType Flags, short Amount)	Queues “do not repeat” for last event triggered
+SendButton(ushort ButtonCode, ButtonFlagsType Flags)	Sends virtual button presses
+SendButton	Sends changes in button planes on axis

Button Sender

Description	
Cues up button signals when buttons are triggered	
Attributes	
+_clientType	Verifies client is currently active
+Connected	Verifies connection to EventClient
Methods	
+ButtonSender(ClientType clientType)	First instance of ButtonSender; establishes connection
+ButtonSender(ClientType clientType, string ipAddress, int port)	Sends new signal to eventClient whenever a button is triggered
+Disconnect	Disconnects from eventClient after signal sent
+SendKey(System.Windows.Forms.Keys key)	Sends a keypress whenever button is triggered, if set to do so
+SendNotification(string title, string content)	Sends a string signal on trigger, if set to do so

Button Codes

Description	
Changes button signals depending on plane of activation	
Attributes	
+_clientType	Verifies client is currently active
Methods	

+SetType(ClientType clientType)	Sets current plane based on hand location

Hand Handler

Description	
Handles user input as user moves hand	
Attributes	
+buttonSender	Passes region of hand to EventClient
+currentPoint	Current location of hand
+handMode	Verifies hand is currently being tracked
+intervalTimer	Value of how often hand region is updated
+lastDirection	Last region location of hand
+lastPoint	Last location point of hand used to define region
+lastRealWorldPoint	Last consolidated point of hand
+stopRepeat	Used to stop repeating signals conflicting
Methods	
+Dispose	Clean up resources on exit
+HandHandler	Initializes HandHandler, makes socket connections needed to properly run
-HandleBackPlaneMode(Point currentPoint)	Switches actions to back plane of activity, normally for terminating program
+HandleMove(Point? currentPoint, Point? currentRealWorldPoint)	Either updates lastRealWorldPoint with user movement, or makes moves to exit program
-HandleNormalMode(Point currentPoint)	Sends button signals based on hand position
+HandlePush(Direction pushDirection)	Recognizes changes in plane, sends signals for changes in plane of action

-intervalTimer_Elapsed(object sender, ElapsedEventArgs e)	Updates hand when “frame” timer elapses to keep feeds updated
-ResetTimer	Resets the interval timer

Sensor Handler

Description	
Handles activity of the sensor as it tracks hand	
Attributes	
+context	Establishes initial sensor signal
+flowRouter	Establishes relay tracking of hand
+handDetected	Verifies hand is being tracked
+isOK	Verifies program activity is normal
+lastRealHandPoint	Last consolidated point of hand
+pointDenoiser	Cleans lastRealWorldPoint
+readerThread	Thread set aside to track hand
+sessionManager	Manages application while active
+slider2D	Defines regions of response
+terminate	Safe exit from use
Methods	
+Dispose	Clean up resources on exit
-handHandler_DropSessionTrigger(object sender, EventArgs e)	Ends tracking session safely
-pointDenoiser_PrimaryPointUpdate(object sender, HandPointContextEventArgs e)	Cleans up hand location to a single consolidated point
+SensorHandler	First instance of SensorHandler, initializes all relevant components

-sessionManager_SessionEnded(object sender, EventArgs e)	Cues up dispose to exit activity safely
-SessionStarted(object sender, PointEventArgs e)	Activates upon hand detection by sensor; sends signals to update hand location
-slider2D_ItemHovered(object sender, SelectableSlider2DHoverEventArgs e)	Detects if the hand is hovering over an onscreen object
-slider2D_ItemSelected(object sender, SelectableSlider2DSelectEventArgs e)	Sends signal confirming that user is attempting to select a signal response
-SpinInfinite	Continuously updates the sensor, even during inactivity

WindowCapture

Description	
This class is responsible for capturing windows, then sending a byte stream of encoded images to the server	
Attributes	
hWnd:HWND	Pointer to the application window to be captured
hdcActive:HDC	Pointer to the display context (DC) of the current windows
hdcMemDC:HDC	Pointer to the display context in memory, which will be the same size as the current window DCs
hbmActive:HBITMAP	Bitmap datatype.
youStream:IStream*	Data Structure that is in memory used to stream data to a socket
count:ULONG	Long integer to store the number of bits actually sent, used to ensure every bit is sent over the network
full:ULARGE_INTEGER	Stores the size of the jpg.
result:INT	Stores success or failure of writing buffer into istream
buffer:char*	Buffer used to prepare data for sending over socket

Methods	
+CaptureAnImage(HWND active): int	Function to capture an image given a window handle
+GetEncoderClsid(const WCHAR* format, CLSID* pClsid):int	Converts bitmap images to jpg
+EnumWindowsProc(HWND hWnd, LPARAM lParam):BOOL CALLBACK	Enumerates through all windows and generates a thread for each unique window

WindowCapturerController

Description	
This class is responsible for initializing the window capture process and for generating threads for every window.	
Attributes	
ip:char	Stores address of server ip
buttonpress:int	Stores if the connect button was pressed on the gui
connected:int	Stores if client has connected
pDataArray[]:HWND	Multithreading data struct stores all window handlers
dwThreadIdArray[]:DWORD	An array of unique thread handles to access their pointers in constant time
ThreadArray:windptr[]	A pointer to an array of windptr stucts, which will contain a mapping of threads to window handles
ConnectSocket:SOCKET	Socket id
szTitle[]:TCHAR	Title of the gui
szWindowClass[]:TCHAR	GUI
Methods	
ConnectToServer(): int	Function that establishes connection to the

	server
WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam):LRESULT CALLBACK	Function that is called whenever any event on gui such as button press occurs.

C. Design Patterns

We chose the publisher-subscriber model design pattern for this system. The server will publish the data streams it receives from the GestureTracker subsystems to the virtualization client. The interpret gestures handler in the client will handle the data and send the appropriate action to the Windowobjs. This decouples the GestureTracker system to the virtualization client it does not need to interpret the gestures and call the specific function of that gesture.

D. Traceability Matrix

	Software Classes							
Domain Concepts	App	WindowObj	Reticulum	Window Obj Controller	Gesture	*Web Server*	Event Client	Button Code
WindowObj Controller				x				
WebClient	x		x					
GUI						x		
WindowObj		x						
Interpret Gestures					x			
WindowCapture Controller								
iStream								
Sensor Handler								

ExtractHand								
Hand Handler								
ButtonCodes								x
Button Sender								
EventClient							x	
WebServer						x		

	Software Classes					
Domain Concepts	Button Sender	Hand Handler	Sensor Handler	StreamViewer	WindowCapture	WindowCapture Controller
WindowObj Controller						
WebClient						
GUI						
WindowObj						
Interpret Gestures						
WindowCapture Controller					x	x
iStream					x	x
Sensor Handler			x			
Hand Handler		x				

ButtonCodes						
Button Sender	x					
EventClient				x		
WebServer						

As we can see from the figure above, several concepts were mapped directly to their respective classes since our domain concepts this time around reflected our final design more accurately.

E. System Operation Contracts

Name	**Visualizing Virtual Windows**
Preconditions	1. Computer must be running window capture program 2. Window capture program must be communicating with Server 3. Phone must be communicating with Server 4. Phone must send a request to Computer
Postconditions	1. Workspace is displayed/updated in virtual space 2. WindowObj instances update, if necessary 3. NumberOfWindows updates, if necessary

Name	**Gesture Tracking**
Preconditions	1. User has compatible gesture tracking equipment connected to the system 2. User performs movement/gesture recognizable by the system 3. SendCommand is not currently in action 4. GestureController ceases hand acquisition
Postconditions	1. GestureController updates all gesture objects 2. GestureController resumes hand acquisition

Name	**Specify certain windows to display.**

Preconditions	1. There must be valid input, either from a mouse or from tracked hand gesture controls 2. Requisites for Visualizing Virtual Windows must be fulfilled 3. NumberOfWindows >= 1
Postconditions	1. User specified windows are rendered and displayed on phone. 2. isRendered is updated for all WindowObj instances

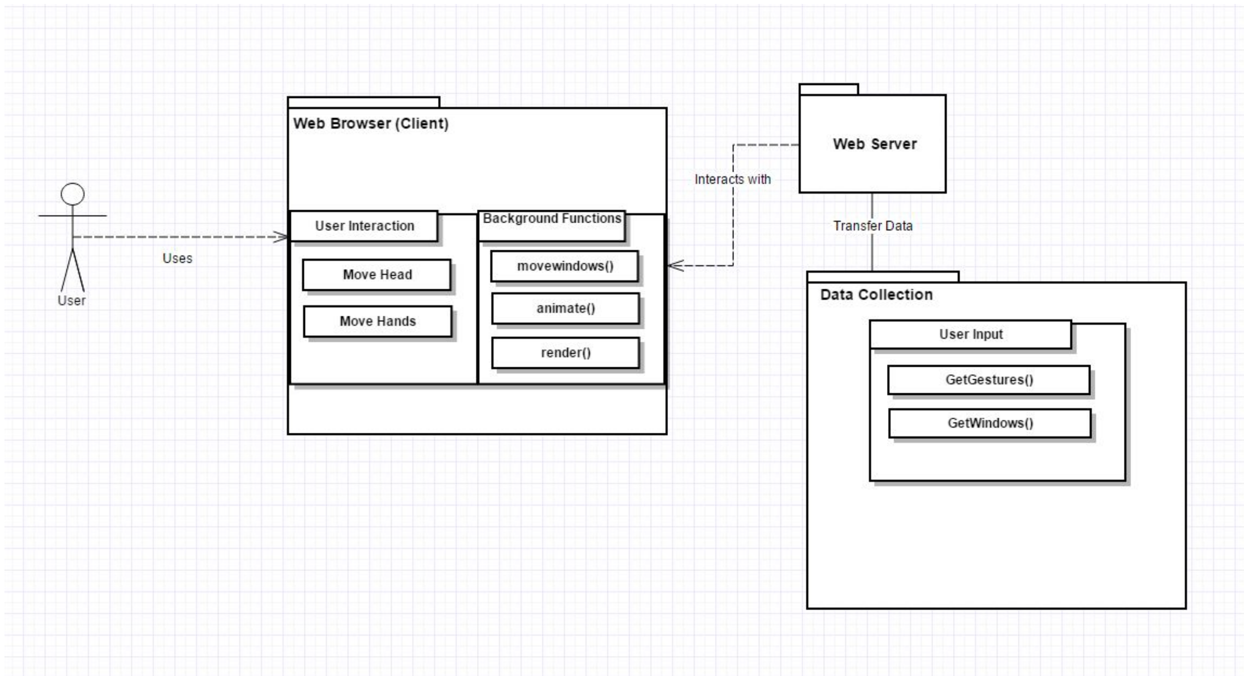
9. System Architecture and System Design

A. Architectural Style

The system uses a combination of mainly two architectural styles, the Model-View-Controller architecture and the Layered architecture. Starting from the user, the user's inputs are picked up by the gesture tracking hardware, which then communicates with several controllers that then interface with separate models for translating the user's gestures into actions which are reflected in the view, for the user, while in turn updating other controllers.

The Layered implementation comes through with the user and the hardware being in the first layer, which communicates with the models and controllers in the second layer, which in turn communicates with a server that serves as the third layer, bridging the models and controllers to each other, as well as sending updates all the way up to the first layer for the user.

B. Identifying Subsystems



C. Mapping Subsystem to Hardware

The system uses a client-server architecture structure; due to the simplicity of the architecture, the hardware hierarchy is simple. The server is run off of the user's computer, which transfers all relevant data related to gesture control and window capture to the client. The data collection is done by a Kinect which captures the user's hands; the window capture is done via software on the user's computer. The virtualization subsystem takes place on the client which is the user's smartphone. By utilizing this distribution of work, we can provide the user the most optimal experience.

D. Persistent Data Storage

This system does not utilize persistent storage; each session is independent of any other session.

E. Network Protocol

The client-server architecture is supported by the Hyper Text Transport Protocol, commonly known as HTTP. The client loads the app from the web server using HTTP and standard network routing. We use HTTP/1.1, even though HTTP2 is much better at streaming data and handling concurrent data processing, because it's still not widely adopted and the laymen's documentation on HTTP2 is still very sparse.

The programs on the host computer (the server, window capture, and gesture capture programs) all communicate using internal TCP sockets. Both the Window Capture program and the Gesture Capture program captures their respective data, then stream the data to the server using a socket. Currently, the window capture program is designed to create a new thread for each window, keeping the main function free to handle input as it comes.

F. Global Control Flow

Execution Orderness
Expanse is generally a procedure-driven system. The user is required to launch the Expanse program on their desktop, select their desired windows, and calibrate their hands before they can view and interact with their windows virtually.

Time Dependency
The system will make use of timers to keep a real-time image of the applications that are being visualizing. Timers will also be used to check response times between the server and client.

Concurrency
Expanse is not set to utilize multithreading at this time. The Window Capture Program is designed for both single threading and multithreading, at different versions.

G. Hardware Requirements

Computer Specification

	Min	Recommended
OS	Windows 7/8	Windows 7/8
RAM	2 GB	4GB
Graphical Processor	Intel HD 4400(DX 11 supported)	Nvidia GTX 660(DX 11 supported)
Processor	64 bit processor Dual-core 2.66Ghz	64 bit processor Dual-core 3.1 Ghz or better
Storage	Any	Any

Kinect

Color Camera	640 x 480 @ 30fps
Depth Camera	320 x 240 @ 30fps
Audio	16 bit audio @ 16kHz
Max Depth Distance	3.5 meters
Min Depth Distance	40 cm in near mode
Horizontal Field of View	57 degrees
Vertical Field of View	43 +/- 27 degrees
USB Standard	2.0
Supported OS	Win 7, 8

Smartphone Specification

	Min	Recommended
OS	Android KitKat 4.4 or higher	Android 5.0 Lollipop or higher
RAM	1 GB	3 GB
Graphical Processor	Adreno 306	Adreno 330
Processor	1.4GHz Qualcomm Snapdragon 410	1.8GHz Qualcomm Snapdragon 808
Storage	1 GB or higher	1 GB or higher
Display resolution	1280 by 720 pixels	1920 by 1080 pixels

10. Algorithms and Data Structures

A. Algorithms

Virtualization

Although Expanse may seem like it requires intensive computation to adequately render and display the VR scene, a lot of the major graphical work is done for us by Three.JS. Three.JS has a well-defined application lifecycle that we take full advantage of, so we do not have to worry too much about creating a complex 3D rendering engine. However, we have decided to implement a few micro algorithms to improve usability of the application, pertaining to *Window Orientation* and *Layout/Distribution*.

Window Orientation

When a user repositions the window, its orientation (relative to the user) should change as well. It should stay perpendicular to the user's hand during the action, so that when a user pushes it off to the side, it won't just move linearly but instead emulate a circular path around the user. This function follows the mathematical model defined in the first report.

In pseudo-code:

```
def move(windowId, x, y, z):
    w = windowObjController.getWindow(windowId)
    w.setPosition(x, y, z)
    w.setOrientation(x, y, z)

// defined in WindowObj
def setOrientation(x, y, z):
    posVec = Vector(x, y, z)
    oldPosVes = Vector(self.x, self.y, self.z)
    theta = acos(posVec.xVec.dot(oldPosVec.xVec)/ ( posVec.length *
oldPosVec.length) )
    phi = acos(posVec.yVec.dot(oldPosVec.yVec)/ ( posVec.length *
oldPosVec.length) )
    w.setOrientation(theta + self.theta, phi + self.phi)
```

Window Creation & Distribution

When the application starts up, it is set to send the feeds of all the currently running applications on the host computer. This can clutter up the virtual workspace, so we decided to auto-layout the windows upon start-up and later creation. Windows will be created in a radial-manner, from the center.

In pseudo-code:

```
// defined in WindowObjController
def createWindows(windowFeeds):
    curLevel = 0
    For (i, feed in windowFeeds):
        If (i**2 > curLevel) then curLevel += 1
        w = createWindow(feed)
        R = Math.ceil(i / level^2)    // vector from center
        w.move(r.x, r.y, r.z)
```

FOV Tracking and Window Choice

This feature is not currently implemented, but may be in the future.

Given the center coordinates and size of the current field of view, as well as the center coordinates and size of each window, we have all the information we need to decide whether or not to render a window. For the sake of efficiency, since this calculation must be repeated many times a second, we will use a simple scheme, and find the difference between the x coordinates of the centers of each window as compared to the center of the field of view. Of those that have a smaller difference than half of the sum of the field of view width and the window width, we will further filter by their y coordinate. We will filter out those windows which have a larger difference in their y values by over half the sum of the field of view height and the respective window height, and thus be left with only windows that are in the field of view.

In pseudo-code:

```
//Returns 1 if in fov, 0 if not
int[] inFOV(windowObj[] inq){ // inq for in question
     int[] result;
     int diff_x;
     int diff_y;
     result = malloc(sizeOf(int) * lengthOf(inq));
     setToOnes(result); //function to make every value in result 1
     //result will be an integer array with length equal to
```

```
      for(int i = 0; i < sizeOf(inq); i++){
          diff_x = FOV.center_x + inq[i].center_x;
          diff_y = FOV.center_y + inq[i].center_y;
          if( diff_x < ((FOV.x + int[i].x)/2)){
                result[i] = 0;
          }
          elseif( diff_y < ((FOV.y + int[i].y)/2)){
                result[i] = 0;
          }
      }
     return result;
}
```

Frame-rate Tracking

We will make two global variables to keep track of the number of updates we can accomplish in every second. The first will be the displayed frame rate, and the second will be a number that counts up every second, and refreshes to 0 on the second, posting its previous value to the first global variable.

Gesture Tracking

As no gesture recognition algorithms exist natively within the Kinect for Windows SDK, it will fall upon the gesture tracking team to implement algorithms for every step of the gesture tracking process. To this end, we take some inspiration from the works of Daniel James Ryan of the University of Stavanger, who began an exploration into using the Kinect for purposes outside its intended use.

The gesture tracking process works by tracking the location of the entire hand in different regions of the screen. Based on the location of the hand, a corresponding buttoncode is sent to the server.

B. Data Structures

Three.JS exposes some primitive data structures that the Virtualization engine has inherited from to create the client-side WindowObj and WindowObjController. The behind-the-scenes implementation of WindowObj is really just a group of geometries and materials covering those geometries, both of which are well-defined by Three.JS. Building upon those, we added internal methods to make these windows interactive, manipulable, and overall functional. These additions include movement functions, to reposition and reorient the window, initialization methods for creation, destruction methods for garbage-collection, and a few other useful methods.

Originally, the windows capture team would share a WindowObj structure with the virtualization team, focusing on the CoordinateStruct associated with each WindowObj to manage the FOV tracking, and describing which windows to render that are currently in the user’s field of view. The WindowObj structs would beused to store the textures for each window, prior to their rendering. For quick access and ease of management, we would store the structures in an array, and use the indexes of that array to quickly manage which WindowObj structs we do and don’t render.

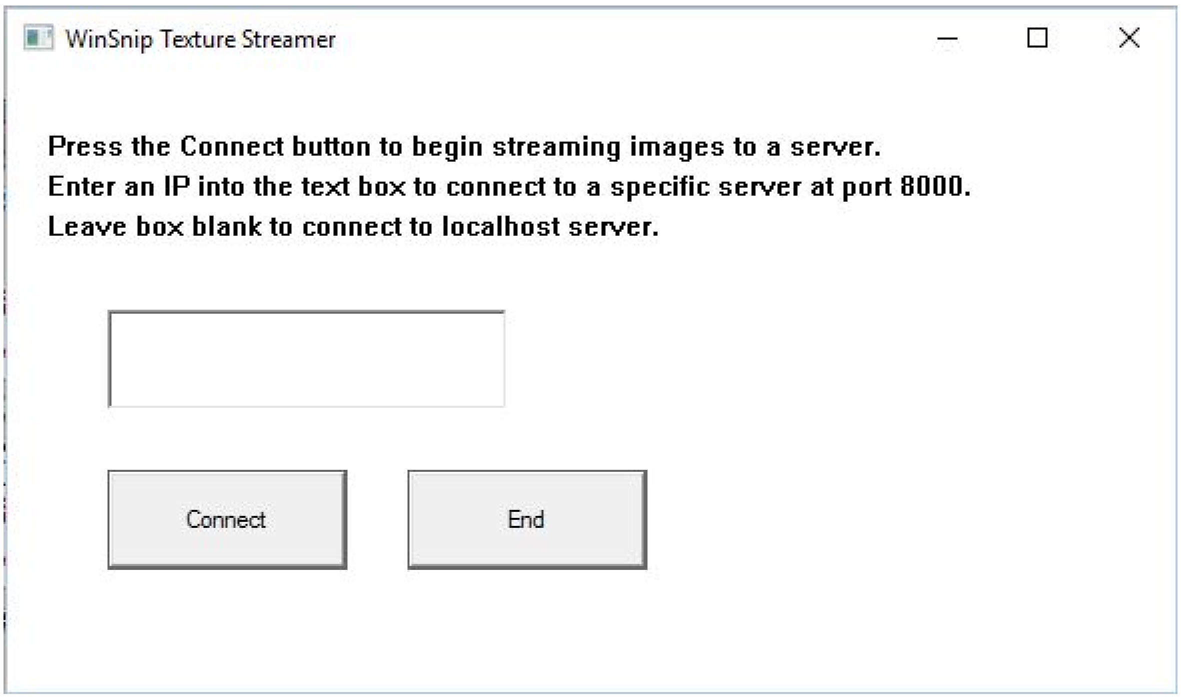
In the current implementation, storage is minimized on the window capture end. Rather than create arrays for texture storing, we thread once for each window, sending each feed in a continuous stream without the need for storage. FOV tracking is currently unimplemented as well, so there is no data structure scheme window capture follows at the moment.

The gesture tracking team uses the array as its primary data structure, storing lists of contour points from the input in an array, and comparing the resulting fingertip mappings to the gesture fingertip mappings stored as arrays within the system.

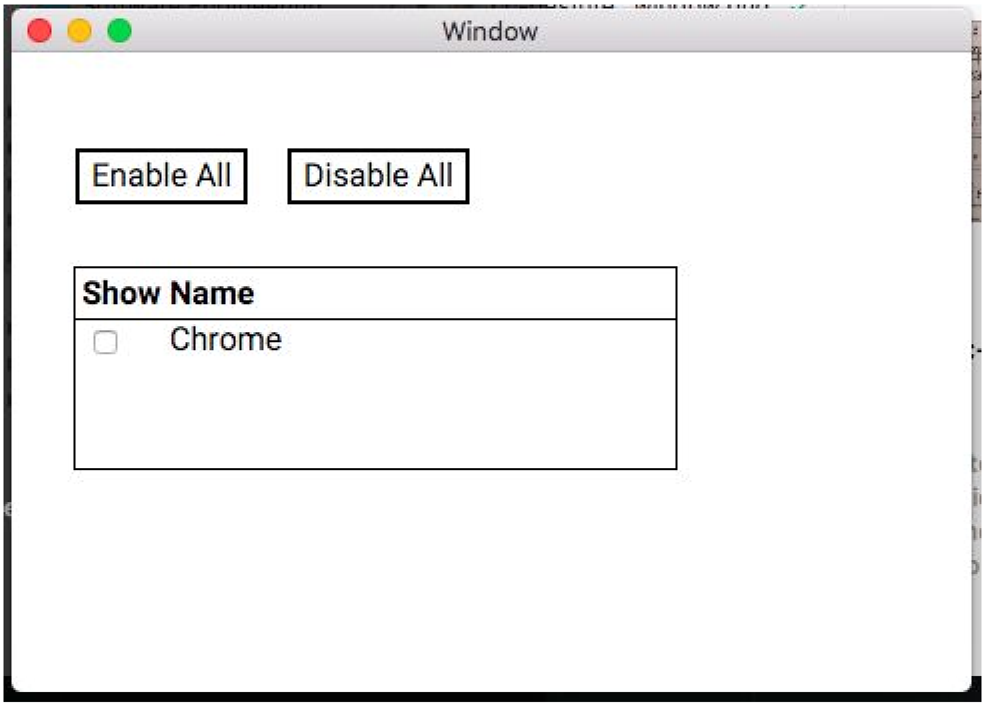
11. UI Design and Implementation

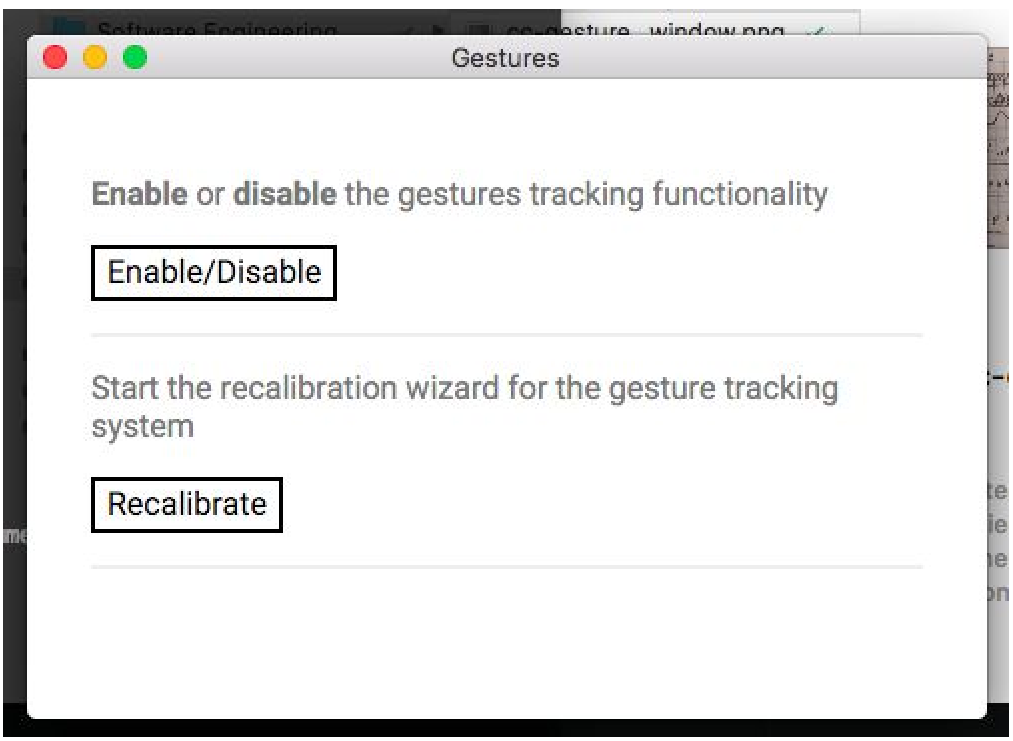
A. Review and Comparison to Initial Projections

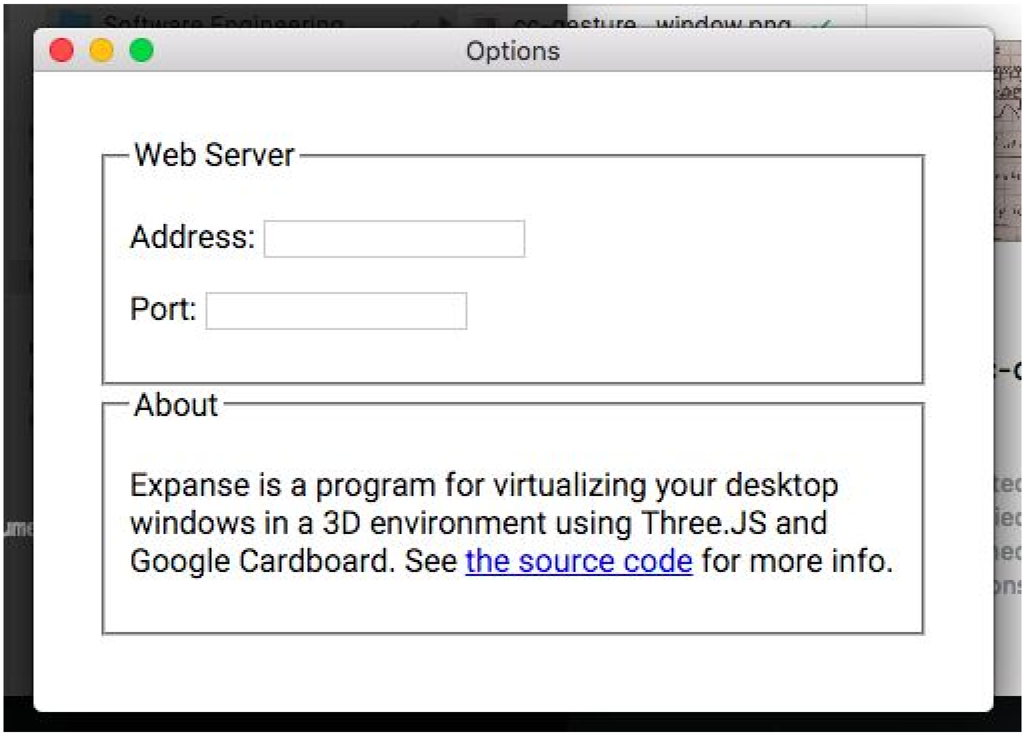
Window Capture Debug GUI



Initial GUI

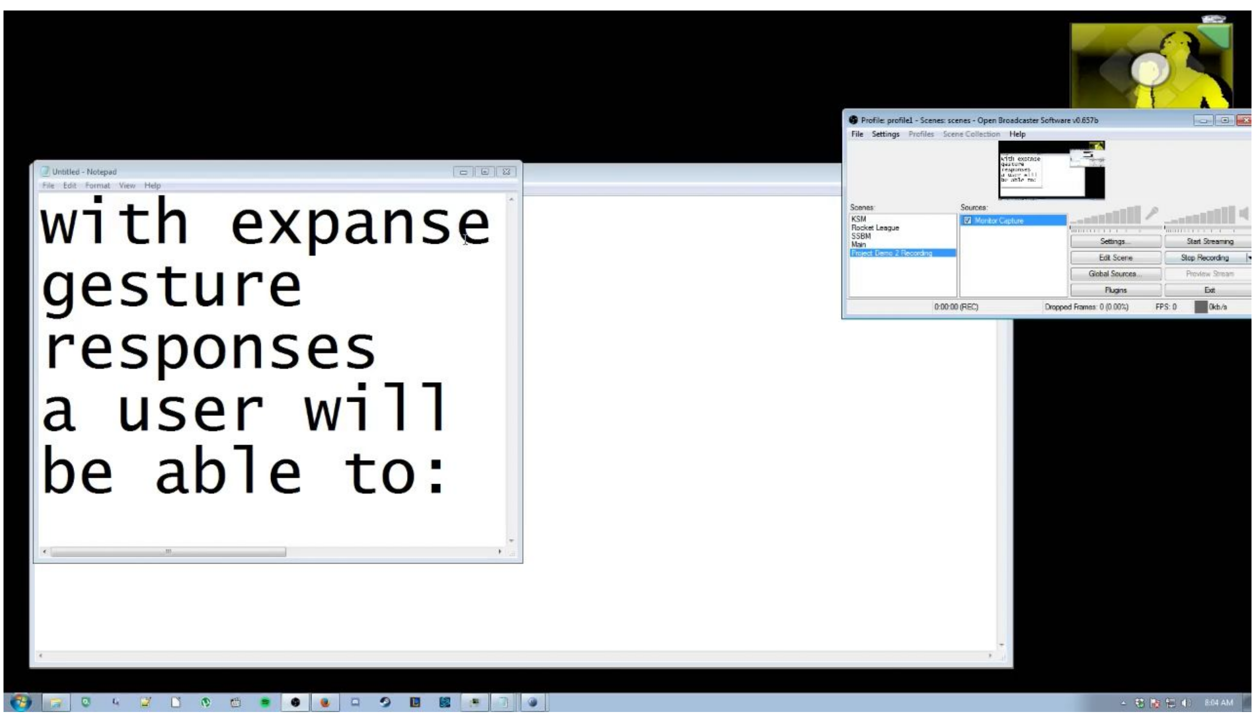

Window
Enable All
Disable All
Show Name
Chrome


Gestures
Enable or disable the gestures tracking functionality
Enable/Disable
Start the recalibration wizard for the gesture tracking system
Recalibrate



We originally planned for there to be three major windows in the application, centered around a system tray icon. These windows were implemented, but we ultimately did not have enough time to link the main server processes with the GUI.

Gesture Capture GUI



In going forward with the new ideas regarding the implementation of gesture tracking, it made sense to reflect these new ideas in an updated GUI, which in this case was able to forego much of the previous UI's bulkiness by focusing on showing only the parts key to operation: a view of the user's hand and the possible options available for the user to select from the singular screen.

B. Discussion

We created multiple debug GUIs in the processing of crafting our system, specifically for our window capture and gesture tracking subsystem. The window capture subsystem presented a small pop-up menu that allows the user to select a port and IP to stream the captured window images and data to. The gesture tracking subsystem offers a GUI that showed what the kinect is currently seeing and tracking, as well a directional overlay for the 'hot' areas that would trigger a gesture. In addition to the real-time video, it also offered textual information abou the hand tracking status and what actions are being triggered.

- Window screen UIs
    - The changes we made to the UI was to make it easier to format the streams and encode/decode the data coming through the connecting sockets.
- Gesture screen UI
    - The gesture screen ui changed quite a bit to accommodate the changing input type/method for processing gestures. Initially we wanted to recognize semi-complex, user-created patterns of movement. However, this proved difficult to implement within the scope of time allotted for the assignment, so we transitioned to a directional-based gesture tracking system. Instead of recognizing a pattern, it would recognize if a user's hand was in a certain area of the recognizable area. We also revised it for easier debugging,
- Options UI
- Window Capture UI
    - The UI was adjusted to allow a user to run the window capture program and close it without needing to open up a console and input their IP as a command line argument. As the window capture program is generally on the same machines as the server, the default option for a blank IP box would be the localhost machine.
- Gesture Capture UI
- Actual WindowObj UIs

Unfortunately, the major application GUI was not able to be linked an implemented with the actual server application, due to time constraints and other issues in the code base.

12. Design of Tests

A. Test Cases

[Deprecated] Test Case Name: Window Render Selection **Function Tested:** refreshWindow() **Pass/Fail Criteria:** Pass if only the correct windows are sent to the render server. Fail if any incorrect window is sent to the render server.	
Test Procedure	**Expected Results**
Create windows completely to the left, right, above, and below the FOV.	No windows rendered
Create windows approximately around the border of the FOV- half in, half out.	All windows rendered
Create windows completely within the FOV	All windows rendered
Merge a few windows from each above case	Only windows from cases 2 and 3 rendered
Create window outside of the FOV. Then change FOV to view window.	No windows rendered, then 1 window rendered once FOV changed
Create window within FOV. Then adjust FOV to view nothing.	1 window rendered, then no windows rendered once FOV changed

Test Case Name: Window Render accuracy **Function Tested:** windcap() **Pass/Fail Criteria:** Pass if window images created visually match what the user sees. Fail if they do not		
Test Procedure	**Expected Results**	**Results**
Open a variety of windows	Every window is at least captured	Every window is captured. A few background windows that do not display are also captured.
Open Google Chrome	Window is captured and	Window image is the correct

	updates reflecting changes	size, but only displays black
Open Windows Explorer	Window is captured and updates reflecting changes	As expected
Minimize a window explorer window	Window is still captured, but displays no image	As expected
Open HexChat, an internet relay chat application	Window is captured and updates reflecting changes	Window is not captured, only a 1x1 pixel is captured.

Summary: The window capture program seems to run into some issues with programs that have many child windows, displaying their content by way of children rather than on the main window itself. Windows explorer works perfectly with the program, so we know that the capturing itself works, but fine tuning is required for better performance.

Test Case Name: User Gesture Recognition
Function Tested: Ability of the system to track user hand, relate input to stored gesture

Pass/Fail Criteria
Pass: user's intended input is a gesture and is recognized as one
Fail: user's non-gesture input is recognized as a gesture, or user's gesture input is not recognized as a gesture

Test Procedure	**Expected Results**
Present nothing to the sensor (control)	No gesture recognized
Present non-hand object to the sensor	No gesture recognized
Present non-forehand view of the hand to the sensor, non-gesture position	No gesture recognized
Present non-forehand view of the hand to the sensor, gesture position	No gesture recognized
Present forehand view of the hand to the sensor, non-gesture position	No gesture recognized
Present forehand view of the hand to the sensor, gesture position	Gesture recognized
Present forehand view of the hand to the sensor, gesture position, outside of sensor's	No gesture recognized

effective range	

Test Case Name: WindowObj Creation Function Tested: createWindow(feed,position) Pass/Fail Criteria: Test will pass if a WindowObj is instantiated	
Test Procedure	**Expected Results**
Call function (Pass)	WindowObj is instantiated
Call function (Fail)	Return error if invalid data is passed, feed already exists or position is invalid

Test Case Name: WindowObj Deletion Function Tested: deleteWindow(windowObj) Pass/Fail Criteria: Test will pass if a WindowObj is deleted	
Test Procedure	**Expected Results**
Call function (Pass)	WindowObj is deleted
Call function (Fail)	Return error if invalid pointer is passed

Test Case Name: WindowObj Reposition Function Tested: moveWindow(windowObj,position) Pass/Fail Criteria: Test will pass if a WindowObj is moved to the correct location	
Test Procedure	**Expected Results**
Call function (Pass)	WindowObj starts at a known position and if relocated to a correct position is successful
Call function (Fail)	Test fails Window does not move to the correct location or does not perform the desired effects.

Test Case Name: Virtualize Work Space/WindowObjController Functions
Function Tested: Testing if the workspace is virtualized.
Pass/Fail Criteria: Pass only if the workspace can be manipulated and virtualized. Fail if the workspace can not accept the user's input in workspace.

Test Procedure	Expected Results
User calls for a new window object to open.	New window opens on desktop and virtually in workspace.
User closes window in virtual space.	Virtual window terminated and closed on desktop.
User looks around the space.	The workspace moves with respect to the user's camera.
User drag virtual windows	Virtual windows move only in the workspace, not on the desktop.
User minimizes window in virtual space.	Windows are minimized in workspace only.
User tries to move camera behind window objects	Workspace fails to look behind the windows.
User tries to load workspace with no connection to server.	Virtual workspace fails to render.

B. Test Coverage

Expanse has critical distributed systems infrastructures that are essential to the functionality of the system. Our tests focus on measuring the performance of the virtualization engine on the client, testing about ⅓ of our whole system. We have also written a few tests for examining the performance of the IPC between window capture/gesture capture programs and the server, as well as tests measuring the efficiency and accuracy of the gesture capture program.

C. Plan for Integration Testing

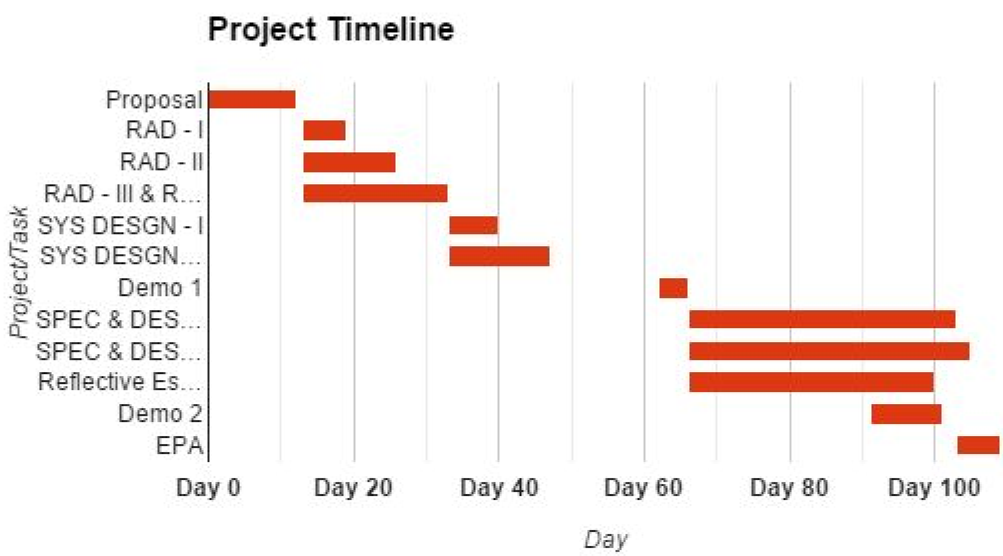
Since we are using a distributed system, we will utilize bottom-up testing, locally testing each component on the same system to ensure we get proper input and output of each subsystem. The purpose is to isolate each component so that we can confirm that errors do not arise from each system. Next, the system will be tested across a distributed system, where the focus of our testing becomes reliable data transfer and latency. We test the responsiveness of the server and it's ability to properly transfer data from programs in C++, to another, in JavaScript.

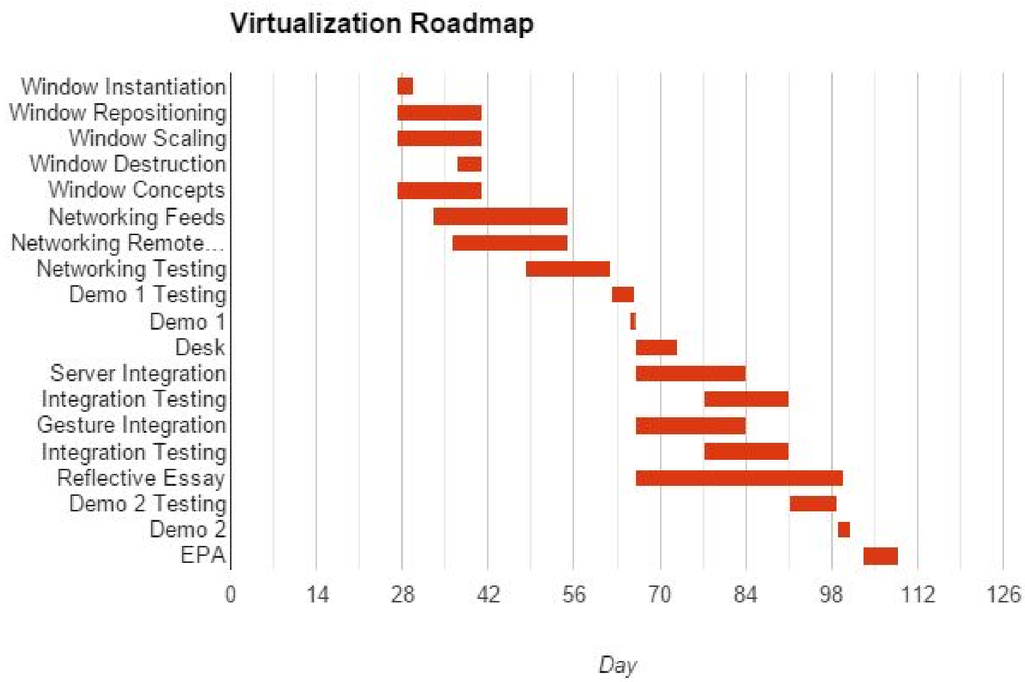
D. Non-functional requirement Testing

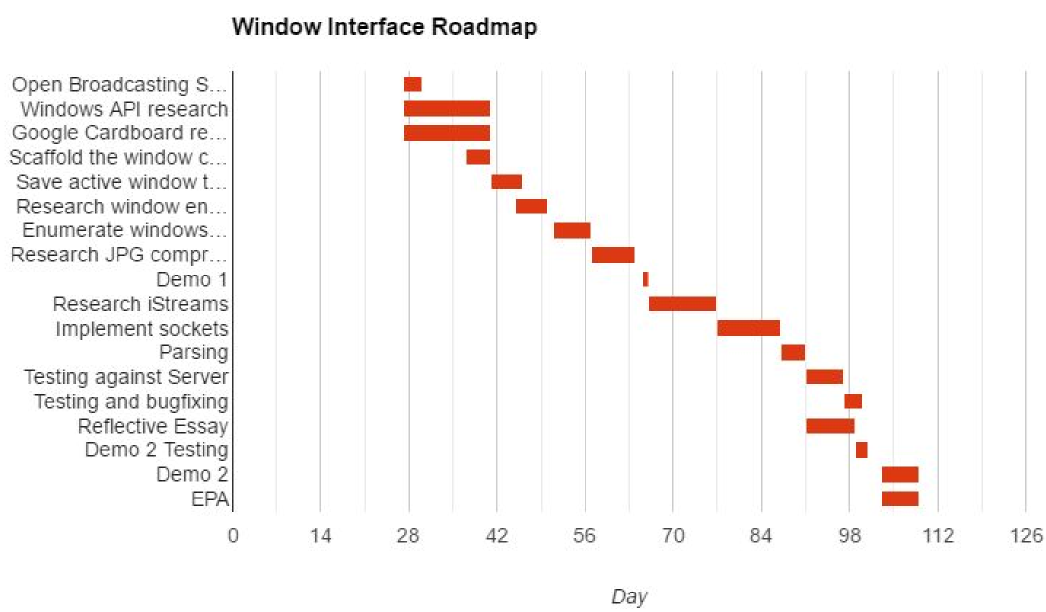
One non functional requirement would be that we maintain a comfortable 30 frames per second refresh rate. We plan to track the refresh rate, and display it in an unobtrusive corner of the user interface.

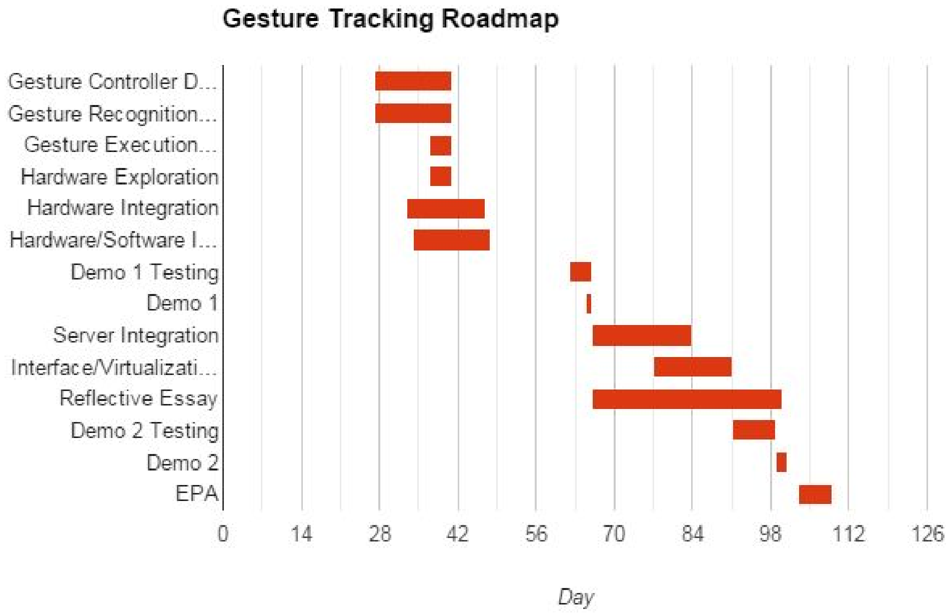
13. History of Work, Current Status, and Future Work

History of work




Virtualization Roadmap
Window Instantiation
Window Repositioning
Window Scaling
Window Destruction
Window Concepts
Networking Feeds
Networking Remote…
Networking Testing
Demo 1 Testing
Demo 1
Desk
Server Integration
Integration Testing
Gesture Integration
Integration Testing
Reflective Essay
Demo 2 Testing
Demo 2
EPA
0
14
28
42
56
70
84
98
112
126
Day


Window Interface Roadmap
Open Broadcasting S…
Windows API research
Google Cardboard re…
Scaffold the window c…
Save active window t…
Research window en…
Enumerate windows…
Research JPG compr…
Demo 1
Research iStreams
Implement sockets
Parsing
Testing against Server
Testing and bugfixing
Reflective Essay
Demo 2 Testing
Demo 2
EPA
0
14
28
42
56
70
84
98
112
126
Day



Current status

At this time all the three subgroups have finished their own part respectively. The gesture team have successfully detected and tracked the gesture with the help of Kinect and translate that to command; The window team can extract all the open windows and the desktop then send the information to the server using TCP; And the virtualization team successfully created virtual windows with 3d views.

Future work

As of right now, the encoding between the window capture program and the expected encoding on the server side is in slight disagreement. The window capture team hopes to resolve the encoding issues, and have proper parsing of each image. In the current scheme, the windows team can only thread for each additional window open, in order to continue handling messages as they come in. If there is no threading, the system will freeze up, blocking on networking rather than handling mouse and keyboard input. As such, we hope to change the server software so that it can support multithreading, rather than node.js, which cannot handle multi threaded networking.

The encoding may not be the issue, but rather the multithreading could potentially be the issue as well. One of the future tasks would be to implement a simplified version of the software to send only the active window, necessitating no enumeration of windows every 50 milliseconds. In this new scheme, the active window would consume the entirety of the user view in virtual space, although VR schemes would be utilized in the view of switching windows. The active window may be changed freely, through the use of alt-tabbing, gesture controls, or the accelerometer in the phone tracking head movement. Although the features would be diminished, the program would be more robust, and easier to use, given that the resolution on phones is low, compared to an oculus rift or similar virtual headgear.

14. References

"About Timers." (Windows). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms644900(v=vs.85).aspx>.

"BITMAP Structure." (Windows). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/windows/desktop/dd183371(v=vs.85).aspx>.

"EnumWindows Function." (Windows). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms633497(v=vs.85).aspx>.

"GDI+ (Windows)". Msdn.microsoft.com. N.p., 2016. Web. 5 May 2016. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms533798(v=vs.85).aspx>.

"GetWindowDC Function." (Windows). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/dd144947(VS.85).aspx>.

"GQPBJ/Reticulum." GitHub. N.p., n.d. Web. 04 May 2016. <https://github.com/GQPBJ/Reticulum>.

"Handles and Objects." N.p., n.d. Web. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms724457(v=vs.85).aspx>.

"Kinect for Windows SDK." 1.8. N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/hh855347.aspx>.

"Kinect Hardware Requirements and Sensor Setup." Kinect Hardware Setup. N.p., n.d. Web. 04 May 2016. <https://dev.windows.com/en-us/kinect/hardware-setup>.

"MemoryStream Class." (System.IO). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/system.io.memorystream(v=vs.110).aspx>.

"Mixing HTML Pages inside Your WebGL - Learning Three.js." Mixing HTML Pages inside Your WebGL - Learning Three.js. N.p., n.d. Web. 04 May 2016. <http://learningthreejs.com/blog/2013/04/30/closing-the-gap-between-html-and-webgl/>.

"NetworkStream Class." (System.Net.Sockets). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/system.net.sockets.networkstream(v=vs.110).aspx>.

"Processes and Threads." N.p., n.d. Web. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms684841(v=vs.85).aspx>.

"Retrieving the Class Identifier for an Encoder." (Windows). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms533843(v=vs.85).aspx>.

"System Requirements." System Requirements. N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/hh855359.aspx>.

"System Requirements." Wikipedia. Wikimedia Foundation, n.d. Web. 04 May 2016. <http://en.wikipedia.org/wiki/System_requirements>.

"Thread Handles and Identifiers." N.p., n.d. Web. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms686746(v=vs.85).aspx>.

"Three.js - Examples." Three.js - Examples. N.p., n.d. Web. 04 May 2016. <https://stemkoski.github.io/Three.js/>.

"Use Case." Wikipedia. Wikimedia Foundation, n.d. Web. 04 May 2016. <http://en.wikipedia.org/wiki/Use_case>.

"User Interface Specification." Wikipedia. Wikimedia Foundation, n.d. Web. 04 May 2016. <http://en.wikipedia.org/wiki/User_interface_specification>.

"Window Features." (Windows). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms632599(v=vs.85).aspx#zorder>.

"Windows." (Windows). N.p., n.d. Web. 04 May 2016. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms632595(v=vs.85).aspx>.

"Windows GDI (Windows)". Msdn.microsoft.com. N.p., 2016. Web. 5 May 2016. <https://msdn.microsoft.com/en-us/library/windows/desktop/dd145203(v=vs.85).aspx>.

Kim, K., Kim, J., Choi, J., Kim, J., & Lee, S. (2015). Depth Camera-Based 3D Hand Gesture Controls with Immersive Tactile Feedback for Natural Mid-Air Gesture Interactions. Sensors, 15(1), 1022-1046

Marsic, Ivan. "Copyright (c) 2012 by The Institute of Electrical and Electronics Engineers, Inc. All Rights Reserved." Software Engineering (2012): n. pag. Web.

Ryan, Daniel James. "Finger and Gesture Recognition with the Microsoft Kinect." (n.d.): n. pag. Web.

“Capturing an Image.” N.p., n.d. Web. 5 May 2016. <https://msdn.microsoft.com/en-us/library/dd183402(v=vs.85).aspx>

“Electron” <http://electron.atom.io/>

“iStream Interface” <https://msdn.microsoft.com/en-us/library/windows/desktop/aa380034(v=vs.85).aspx>.

“Node-http-server” Web. <https://www.npmjs.com/package/node-http-server>.

“Node.js” <https://nodejs.org/en/>

“Server API<http://socket.io/docs/server-api/>

“The Transmission Control Protocol”<https://condor.depaul.edu/jkristof/technotes/tcp.html>

“Three.js <http://threejs.org>