Requirements Analysis

VR Workspace (‘Expanse’)

Group 13:
Allen Tung
Anthony Wong
Dmitriy Kozorezov
Joshua Chan
Marc Tabago
Yue Yang

Changelog:

System scope changed to Android only. Multiple OS compatability was determined to be too ambitious, and outside the scope of our capabilities under the time constraint

Use case name columns added to tables for increased readability

Figure number and captions added

1. OVERVIEW

1.1. Introduction

This project aims to improve upon the applications and accessibility of augmented reality software. Like how the Graphical User Interface revolutionized the personal computer, we hope to create an augmented workspace that will change the way users interact with computers.

The industry does have several virtual and augmented devices on the market. Devices like the Google Glass and a car's Heads-Up Display serve to display relevant information to the user. Though they are useful, the scope and power of their applications is limited or very inaccessible to the normal consumer.

1.2. Problem Diagnosis

Personal Computers have made many great leaps over the decades with increased availability and applications in the industry and consumer markets yet, the inherent design of the computer remains difficult to use for new and average users alike. To the average end user, the computer consists of hardware peripherals(mouse, keyboard ,monitor), the computer itself, a Graphical User Interface and applications that they use. We decided to focus on a major component that the user sees and uses often; the Graphical User Interface.

Nowadays, users depend on a plethora of applications for work and home use. This results in a very cluttered interface where several applications demand the attention of the user. Often users may have to split the screen to fit and view all their applications or maximize and minimize applications constantly in order to manage their workspace. Example as seen in Fig 1-A

Fig 1-A
A very cluttered desktop

Microsoft and Apple, both industry leaders in the personal computer market, have created their own solution to the problem. They utilize virtual desktop managers like Windows 10 Task Viewer or Mac OS Mission Control, where users can create multiple virtual desktops and organize applications based on the user's needs. They can, for example, place work-related applications such as spreadsheets and word-processing on one desktop and have another desktop designated for internet browsing. Example shown in Fig 1-B.

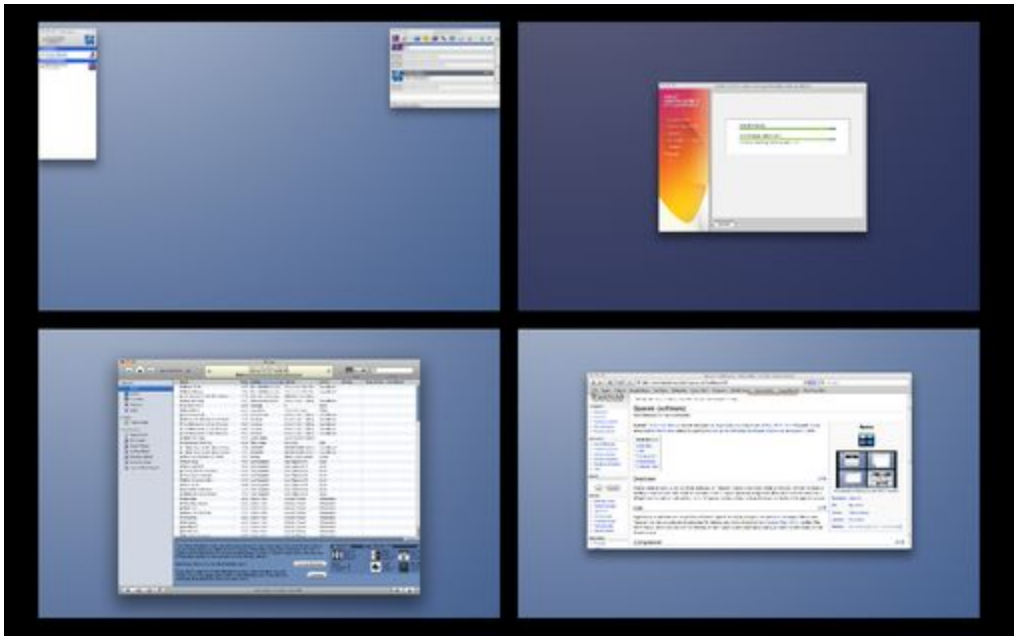


Fig 1-B

An Example of Mac OS X “Mission Control”

However, switching between two spaces still requires user-configured hotkeys or dragging for the user to view their desired applications.

Businesses and consumers looking for a physical solution for more desktop space also have the option of using several monitors in order to create a larger virtual workspace, but this method is not without problems of its own; in accommodating upwards of two or three monitors (to expand horizontally), to even five or six in some cases (to create a taller workspace), constraints are reached in terms of practical space available to accommodate such a setup, as well as in terms of cost of not only the monitors, but also the mounting solutions and even possible changes in desk required. Considering these problems, a need arises for a solution that offers the functionality of a vast virtual workshop that is comfortable in usage, in space, and in overall cost.

We discuss our novel solution to this problem below in section 1.3: Solution Proposal.

1.3. Solution Proposal

We propose Expanse, a tool for visualizing and manipulating your application windows using augmented reality. Expanse can provide every customer with the full functionality of a multi-monitor workspace and much more, at affordable cost.

What is Augmented Reality?

Augmented Reality can best be described as “taking digital or computer generated information … and overlaying them over in a real-time environment”.[1]

Why Augmented Reality?

Augmented reality can be used to supplement and provide more information than one would normally. A simple example would be a heads-up display found in jets; the pilot is able to see the world around him as well as the digital overlay of instrumentation information. This same principle can be applied to an office workspace where we overlay a virtual desktop and present more information to the user.

To better understand the project, consider the follow scenario -
An office worker is sitting at his desk wearing a head-mounted display consisting of Google Cardboard and his smartphone. In front of him is a computer with the standard mouse, monitor and keyboard, on the monitor display is a word processing application, Autocad, and email. He is able to either with a gesture or command, pull these applications into virtual space and overlap his actual view. He can move these applications around the virtual space, say he puts his email application towards his left and his Autocad application to the right. As he turns his head left, the AutoCad application will leave his field of view and he will only see his email application in front of him. Turning back right, the Autocad application comes into view and still remains open despite leaving his field of view earlier.

How can Augmented Reality be used to simplify the workspace?

Many developers have to have a workspace with many monitors and dedicated desktops to be able to work with having many programs running at the same time. These developers use the many monitors to separate the application screens to be able to view them all and work on them all at once. The solution we are working on brings all of these monitors into one space, so you can work with them all in one space rather than having to move between computer monitors.

To make a cost-effective augmented reality device, we sought to use Google Cardboard and smartphone devices. The program itself would take windows from a connected desktop and

display the information in virtual space allowing easy manipulation. By using this method it may save a lot of space and money, by creating a virtual workspace you can access from anywhere by just putting on virtual reality glasses.

On a large scale, this has the potential to save companies a lot of money. By distributing these headsets and uploading their workspace onto it, they will not need to buy as many expensive pieces of machinery and computers. Considering companies have thousands of employees, each with their own dedicated machinery, this will also save the room taken up by these computers, and the need for heavy maintenance.

2. Goal And Scope

2.1. Project Goals

Project Goal	Priority	Comment/Description/Reference
Functional Goals:	1	For details see the Project Requirements Specification [2]
Mirror Desktop	1	Get a working desktop replication on phone
Gesture Control	2	Get webcam to track hands
Augmented Reality	3	View your environment alongside the desktop
Business Goals:	1	
Minimum Usability	1	Meet all priority 1/2 goals for usable product
Consumer Accessibility	2	Ensure final product uses minimal self-instruction
Technological Goals:	2	
Create Chrome app	1	Render desktop in 3D
Full Mobile Integration	2	Flesh out applications across Android/iOS
Quality Goals:	2	
No Command Lag	1	Have the display run seamlessly, with no lag.
Ergonomic Design	2	The unit must be comfortable to wear daily.
Constraints:	1	
Eye Health	1	Safe to look through for an extended period

2.2. Project Scope

Augmented Reality Desktop using Heads-up Display with full gesture interactibility.

2.2.1. Included

1. Support for Android operating systems. In the future, expanding the scope to include support for other operating systems may be viable, but we will show proof of concept for Android operating systems first.
2. Full head and hand tracking

2.2.2. Excluded

1. Support for other virtual reality hardware devices (eg. Oculus Rift, Samsung Gear)

3. Project Organization

Name	Role
Joshua Chan	Virtualization/Rendering and Gesture Tracking
Dmitriy Kozorezov	Virtualization
Marc Tabago	Gesture Tracking
Allen Tung	Interface
Anthony Wong	Virtualization
Yue Yang	Interface
Andrew Chan	Gesture Tracking

3.1. Project Requirements

- A Head Mounted Display that is low cost and ergonomic to the user as this is intended for long periods of use
- Effective tracking (Position and Orientation) of the user for head movement and gesture based(hands and eyes) commands. Important metrics to consider here are:
    - Latency: the amount of time between the movement of the user's head, and the movement of the display, which may lead to motion sickness, or a decreased sense of satisfaction, and feeling of 'fakeness'
    - Accuracy: the range of which the reported position of the user is correct[2]
    - Resolution: The exactness with which a system can located a reported position[2]
- A browser-based rendering library to render and manipulate the desktop and application windows into a 3-D scene, accessible from any modern smart-phone
- A program to capture screens & window information
- A web server to serve the rendering app and window information

3.1.1. Planned Testing

Refer to Schedule section 4.1 for timing of deliverables
Virtualization:
1.1 Able to use Google CardBoard to view a virtual space(eg: a gray box)
1.2 Able to place virtual space onto real scenes(eg: simple 3d shapes into real environment
1.3 Able to place an application window into virtual space
1.4 Able to manipulate application window with a mouse input
1.5 Able to manipulate application window with hand input
1.6 Able to obtain application window from desktop computer

Gesture:
2.1 Able to get hands detected by hardware
2.2 Able for hands to replace mouse controls
2.3 Able for hands to accurately mimic mouse control
2.4 Able for hands to do gesture based controls
2.5 Able for hands to interact in virtual test space
2.6 Able for hands to accurately interact with virtual desktop

Interface:
3.1 Server acquired and setup
3.2 Able for computer and phone to communicate with a server
3.3 Able for computer to capture screen information
3.4 Able for computer to send screen information to phone
3.5 Able for phone to place application window into virtual space

4. Schedule and Budget

4.1. Work Breakdown Structure

A. Virtualization Team: Creating the Virtual Workspace (Shared Infrastructure)
B. Gesture Tracking Team: Creating gesture controls
C. Interfacing Team: Creating the interface between desktop and phone

4.2. Schedule

Letters represent team(s) responsible for milestone

Milestones	Description	Milestone Criteria	Planned Date
M0	**Start Project**		**<2016-01-27>**
M1	**Project goals and scope defined**		**<2016-01-28>**
M2	**Start Planning**	**Approval from Prof. Marsic**	**<2016-01-31>**
M3	**Research into Google CardBoard API(A,B,C)**	**Requirements agreed, project plan reviewed, resources committed**	**<2016-02-01>**
M4	**Research into virtualization software(A)**	**Requirements agreed, project plan reviewed, resources committed**	**<2016-02-06>**
M5	**Research into gesture control software(B)**	**Requirements agreed, project plan reviewed, resources committed**	**<2016-02-06>**
M6	**First report part 1 Statement of Work & Requirements (A,B,C)**	**M3,4,5 completed, documentation written**	**<2016-02-07>**
M7	**First report part 2 Functional Requirements Spec & UI (A,B,C)**	**M3,4,5,6 completed, documentation written**	**<2016-02-14>**
M8	**Full Report #1 Preliminary planning complete**	**Documentation complete as per M6, M7. Framework built.**	**<2016-02-21>**

M9	**Benchmark 1**	**Refer to 3.1 Planned Testing Section 1.1, 2.1, 3.1**	**<2016-02-28>**
M10	**Design Draft 2(A,B,C)**	**Benchmark 1 successfully passed, revisions made**	**<2016-03-06>**
M11	**Benchmark 2**	**Refer to 3.1 Planned Testing Section 1.2, 2.2, 3.2**	**<2016-03-13>**
M12	**Design Draft 3(A,B,C)**	**Benchmark 2 successfully passed, revisions made**	**<2016-03-25>**
M13	**Benchmark 3**	**Refer to 3.1 Planned Testing Section 1.3, 2.3, 3.3**	**<2016-04-01>**
M14	**Design Draft 4**	**Benchmark 3 successfully passed, revisions made**	**<2016-04-03>**
M15	**Benchmark 4**	**Refer to 3.1 Planned Testing Section 1.4, 2.4, 3.4**	**<2016-04-10>**
M16	**Design Draft 5**	**Benchmark 4 successfully passed, revisions made**	**<2016-04-15>**
M17	**Benchmark 5**	**Refer to 3.1 Planned Testing Section 1.5, 2.5, 3.5**	**<2016-04-26>**
M18	**Final Benchmark**	**Refer to 3.1 Planned Testing Section 1.6, 2.6**	**<2016-05-07>**

4.3. Budget

As we aim to make this project a low-cost endeavor, we want to set a budget of $120. The initial costs seem to come from the hardware purchases we'll need to make - a decent webcam for gesture recognition, a microcomputer to do the processing, and some peripheral cables.

Sources:

1. Greg Kipper; Joseph Rampolla (31 December 2012). *Augmented Reality: An Emerging Technologies Guide to AR*. Elsevier. ISBN 978-1-59749-734-3.
2. Tripathi, Anish. "Augmented Reality - An Application for Architecture." *Augmented Reality - An Application for Architecture*. University of Southern California, 2000. Web. 31 Jan. 2016.

Functional Requirements Specification

Stakeholders

1. Government associations
    a. They may use the product in research and development, improving productivity and reducing budget
2. Schools, particularly design schools
    a. Students could gain experience immersing themselves in biology labs, design classes, or virtual field trips with just the cost of this product.
3. Any company with desk workers
    a. This product may reduce the cost of maintaining the workspace, through reduced energy consumption from reducing the number of monitors, and reducing the capital required to purchase said monitors.
4. Designers of all types
    a. They may use this application to model their designs in virtual 3D space, without needing to fiddle with a finicky 3D design software, rotating objects on a 2D screen.
5. General consumers
    a. General consumers may also purchase this product to increase their productivity at home, as many are limited by their budget to purchase large monitor displays. The novel 3D features will also attract interested consumers.

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 changes from the HUD to be saved on the computer itself. Render open windows in 3D.
Tracking Device	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#4	**Visualizing 3D models**	Allows the user to view 3D models and objects
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#9	**Allow Access for Multiple Users**	Allow several different users to be able to use the device and have their own workstations.

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.
UC#12	**Save/Load**	User can save and load information from the computer on the workspace.

Actor	Type	Goal	Use case name
User	Initiator	View currently open windows on desktop through the 3D device	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 3D device	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	Gesture/

		to server to render.	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

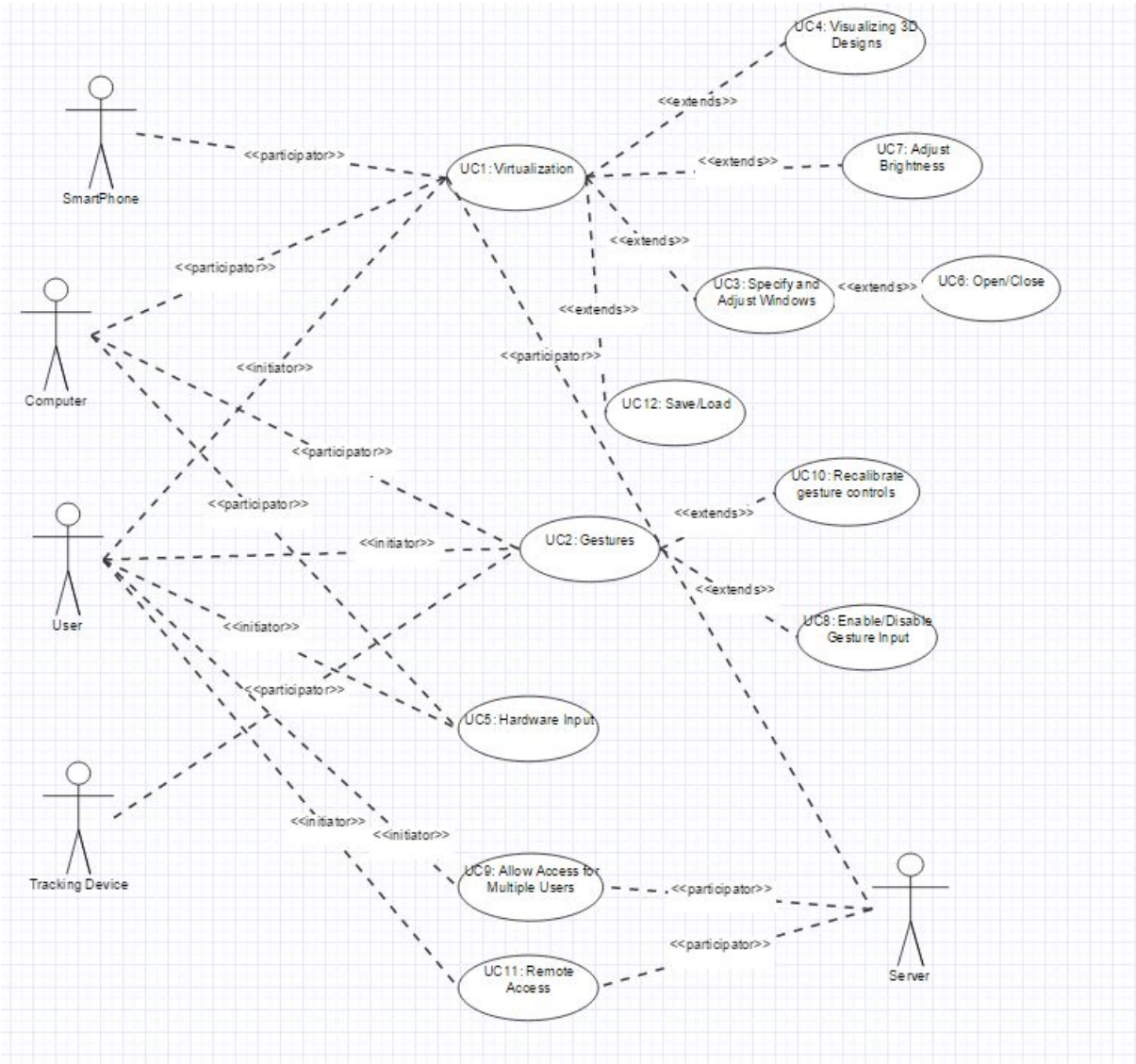


Fig II- Use Case Diagram

iii. Traceability Matrix

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

REQ-20	**x**		**x**									
REQ-21	**x**											**x**
REQ-22	**x**		**x**									
REQ-23	**x**		**x**									**x**
REQ-24	**x**			**x**								

iv. Fully Dressed Descriptions

ID:	**UC#1**
Title:	**Virtualization**
Description:	**Specified desktop application windows are captured and sent by a server, then displayed on the virtual reality program running on the phone.**
Primary Actor:	**User**
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 in virtual space**
Main Success Scenario:	**The User initiates a request to virtualize their workspace. The computer receives this request, then transmits the current window data to the server. The server sends this data to the phone, where it is rendered into the 3D scene. The phone displays the current workspace to the user.**
Extensions:	**If the User specifies certain windows not to be displayed, the computer will exclude these windows from view.** **If there are no open windows, there will be no windows displayed on the workspace.** **If there are more than the maximum number of windows open, only a set limit of windows will be sent to the server.**

	If the computer can not contact the server, an error will be returned to the phone, which requested the virtualization. **If the server can not contact the phone, it will attempt to send an error to the computer. In the meanwhile, the phone will not display anything, nor respond to gesture or peripheral hardware commands, in a frozen state.** **If the phone does not receive a response within a specified time, it will return an error to the user.**

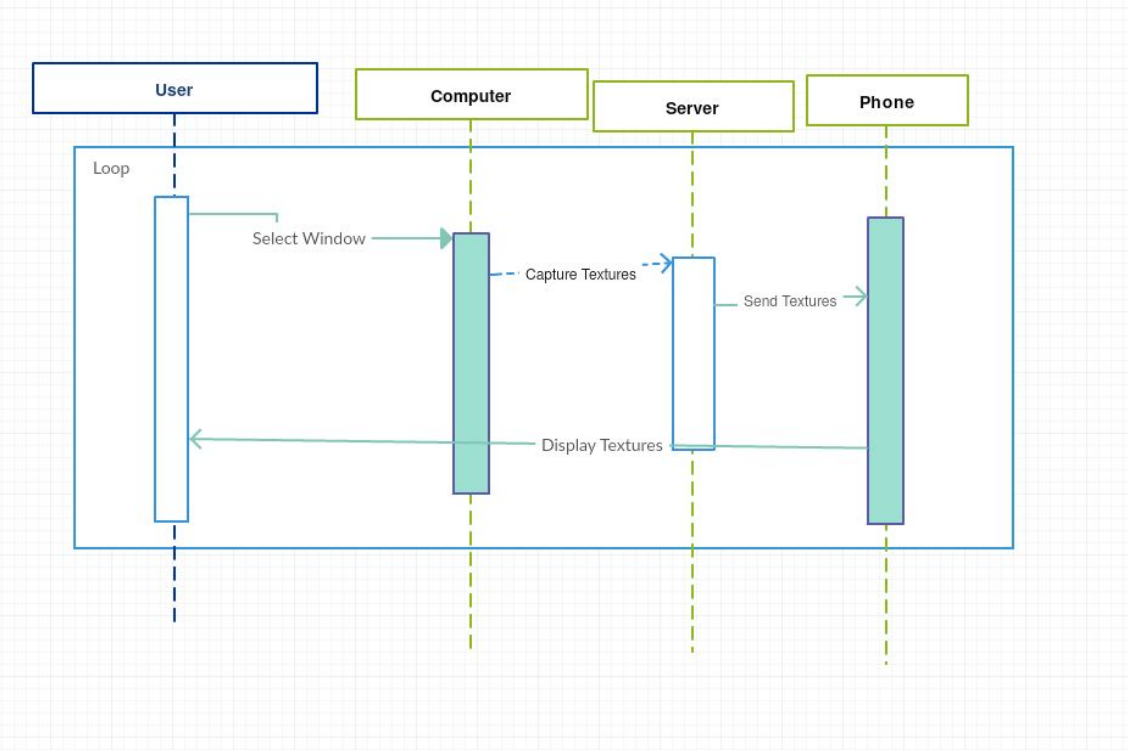
ID:	**UC#2**
Title:	**Gesture Tracking**
Description:	**The system responds to navigational and organizational actions input to it by visually relaying the equivalent system output.**
Primary Actor:	**User**
Preconditions:	**1. User has compatible gesture tracking equipment connected to the system** **2. User performs movement/a gesture recognizable by the system**
Postconditions:	**1. The VR state changes to reflect the changes made by the user.** **2. The system becomes immediately ready to accept another movement/gesture**
Main Success Scenario:	**The user initiates a defined movement with their hands. The tracking device captures this movement, and sends the registered command to the computer. The computer processes the command, and reflects it on the desktop, and/or the current saved state of the VR display. The desktop state updates, and new window information is captured, then sent to the server. The server renders the data, and sends it to the phone.**
Extensions:	**Unrecognized movements and gestures are ignored by the system so as not to distract the user.** **If the gesture tracking hardware loses connection with the system, the system saves the current workspace and**

	attempts to make no changes to it until the hardware is reconnected. **If multiple pieces of gesture tracking hardware are connected to the system, this will recognized as an error, and the current state of the workspace will be saved and freeze. This freeze will hold until the user chooses a primary hardware, or all other hardware disconnects except for one.** **If the user performs multiple recognizable gestures at once, the system queues them via the system's internal gesture priority, and will execute them one after another in rapid succession.**

ID:	**UC#3**
Title:	**Specify certain windows to display.**
Description:	**User will specify certain windows open on the desktop to exclude from the 3D display. Extends UC#1.**
Primary Actor:	**User**
Preconditions:	**1. Preconditions for Virtualization are fulfilled** **2. Hands are tracked and calibrated**
Postconditions:	**1. The user-specified windows are displayed, and no other.**
Main Success Scenario:	**The User chooses from the currently open windows, and sends a command to the computer that they wish to stop rendering and displaying the windows on the phone, but not close them on the computer. The computer acknowledges, and excludes the specified windows from the data it sends to the server. Virtualization continues as normal. The windows disappear from the user's view with a preset animation.**
Extensions:	**If the window is closed on the desktop before the process completes, the computer will disregard the missing windows in the specification request, and send an acknowledgement.** **If no acknowledgement from the computer is received, the phone will continue sending acknowledgement requests**

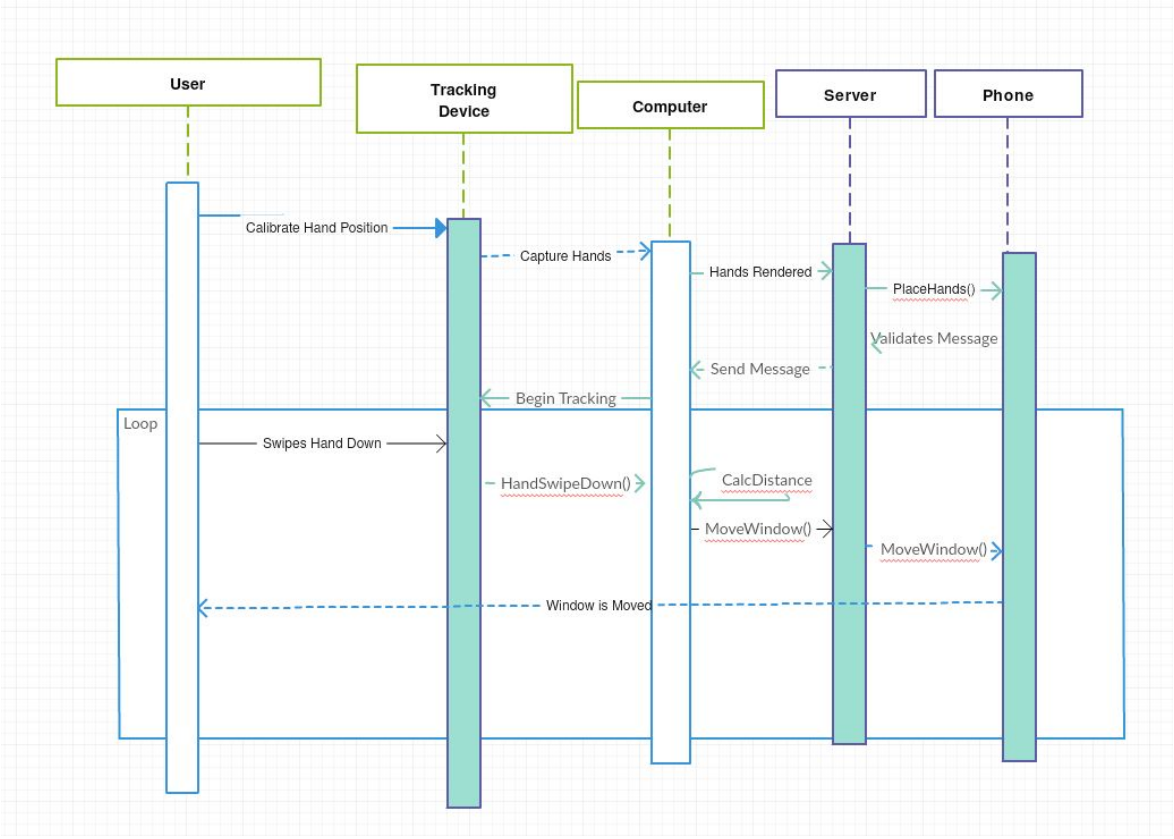
	and communicate with the computer until consensus is reached, or a preset maximum time limit is reached.

System Sequence Diagrams
UC-1, Scenario: Basic



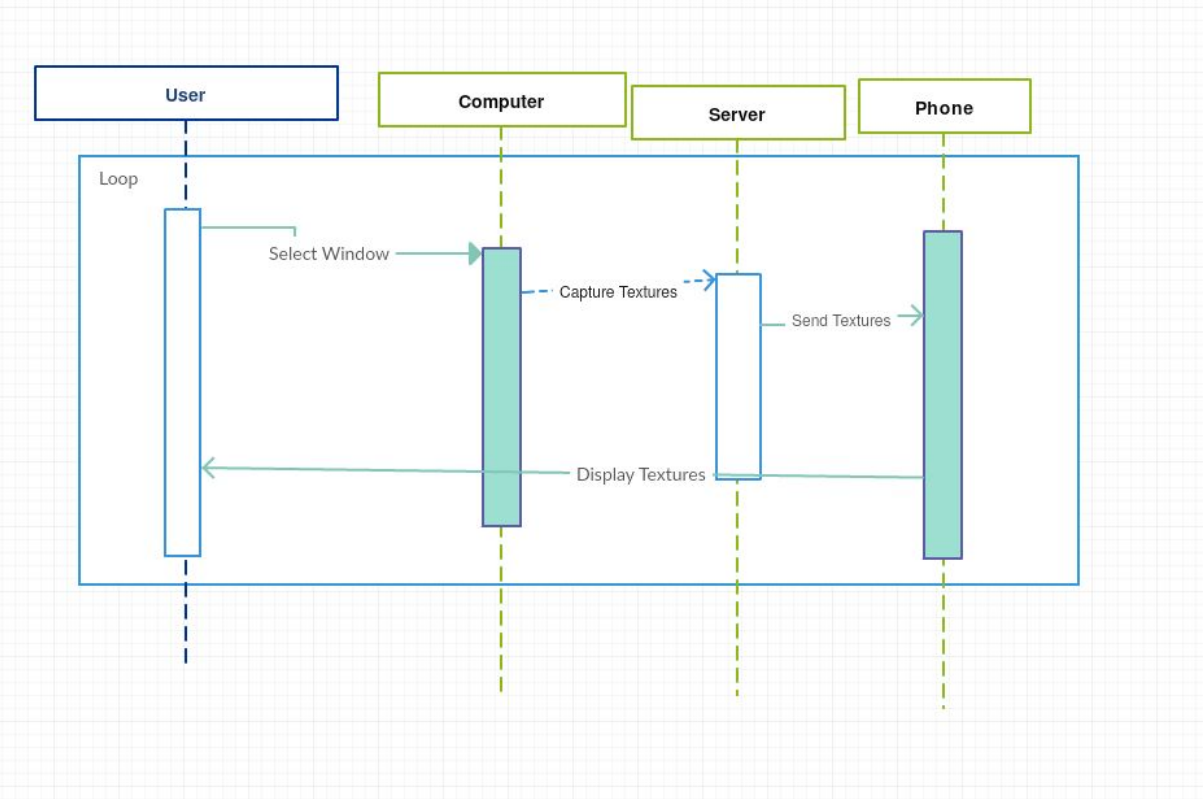
In this use case, we simply go through the window selection process, detailing the sequence that we go through to display windows. We loop between choosing, capturing, and displaying. All the communcation is detailed in the diagram.

UC-2, Scenario: Basic



In this use case, the system responds to navigational and organizational actions input to it by visually relaying the equivalent system output. After initial calibration, we track the hands and loop the second portion whenever a hand swipe is detected.

UC-3, Scenario: Basic



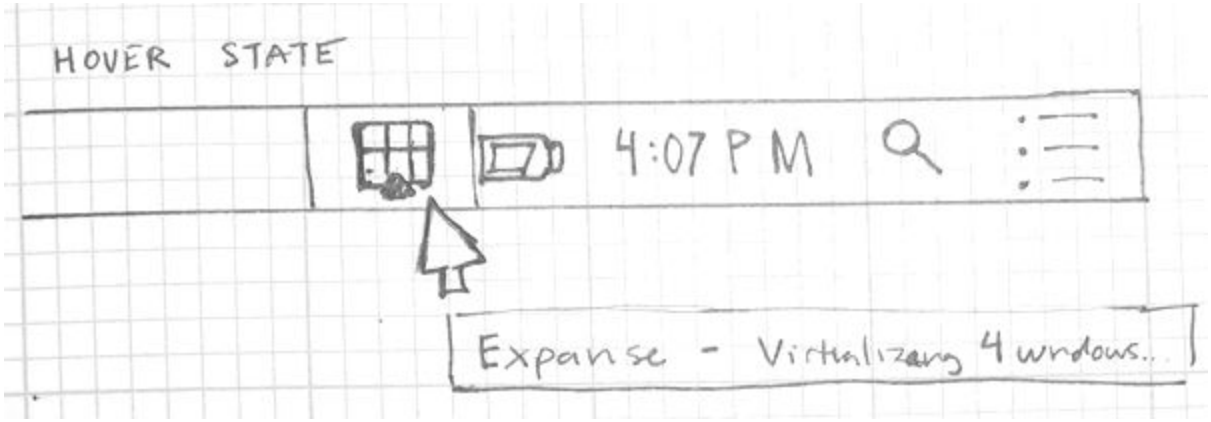
In use case 3, the user will specify certain windows open on the desktop to exclude from the 3D display. It extends UC#1. In use case 1, the selection of windows isn't implicit, although it is still displayed in the diagram, the selection is specifically the main portion of use case 3. The sequence is the same, however.

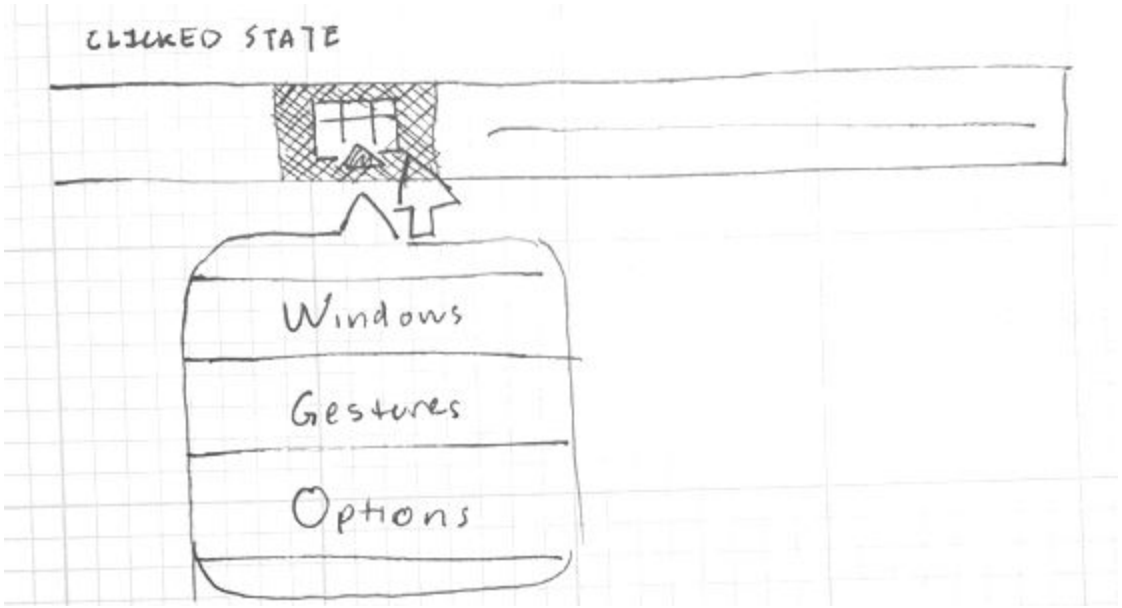
User Interface Specification

A. Preliminary Design

Expanse will have *two* major user interfaces - a **control interface** and **interaction interface**. The control interface will be seen in the host computer running the Expanse application, from which Expanse will be sending screen feeds from. The interaction interface will be seen via the HMD, which is comprised of a Google Cardboard and smart-phone. The interaction interface consists of a virtual, 3d environment with floating windows and other objects, which will be rendered onto the phone screen as two images with a positional offset which the user will see as one three-dimensional image using Google Cardboard's lenses.

In the **control interface**, Expanse will have a small graphical footprint. After starting the application (by clicking the application icon, starting it from the command-line, or what-have-you), our icon will appear in the user's menu-bar or system tray (depending on the OS). That icon will be Expanse's primary interface and, when clicked, will reveal a menu, and, when hovered, will reveal Expanse's current status. Below are example drawings of the hover and clicked states (*note:* the particular icon is not finalized and subject to change).



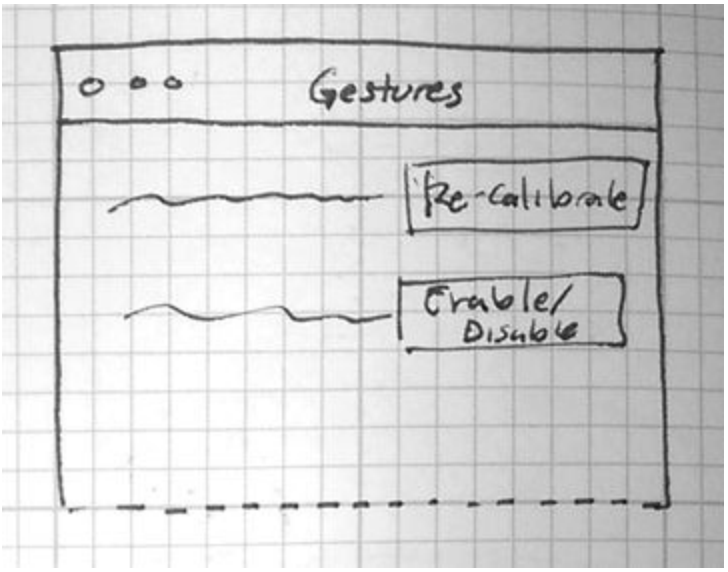


There are three menu items - **Windows**, **Gestures**, and **Options**. Clicking anyone one of the menu items will close the menu and open a pop-up window with information about the given topic, as well as several forms and fields relevant to the selected topic.

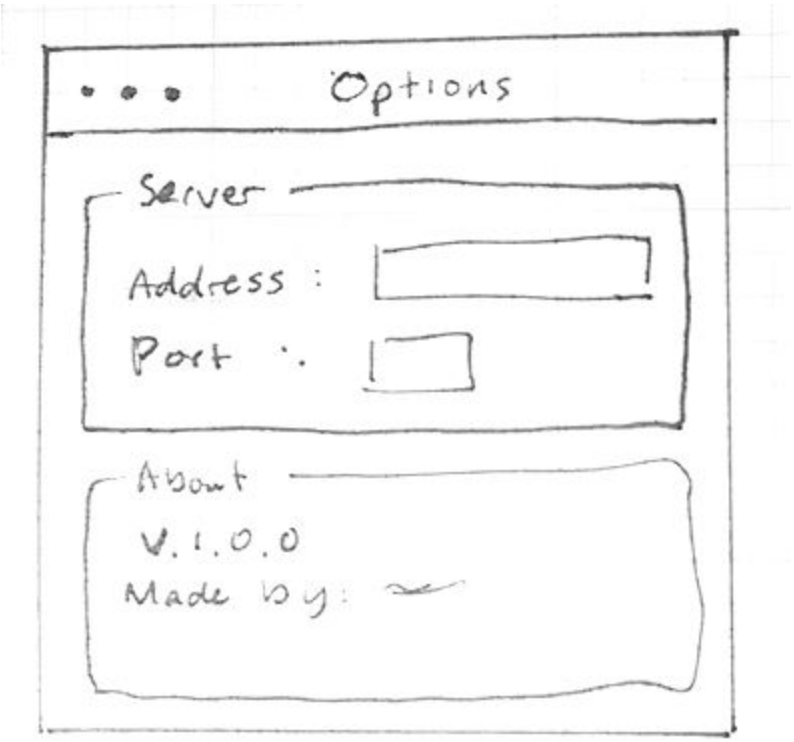
The **Windows** item will open up a window that lists the currently running applications with windows. This list shows and controls what window feeds are sent to the client for

virtualization. You are presented with buttons to either enable all windows or disable them all, and you can also click the option buttons on each item to enable/disable specific applications.

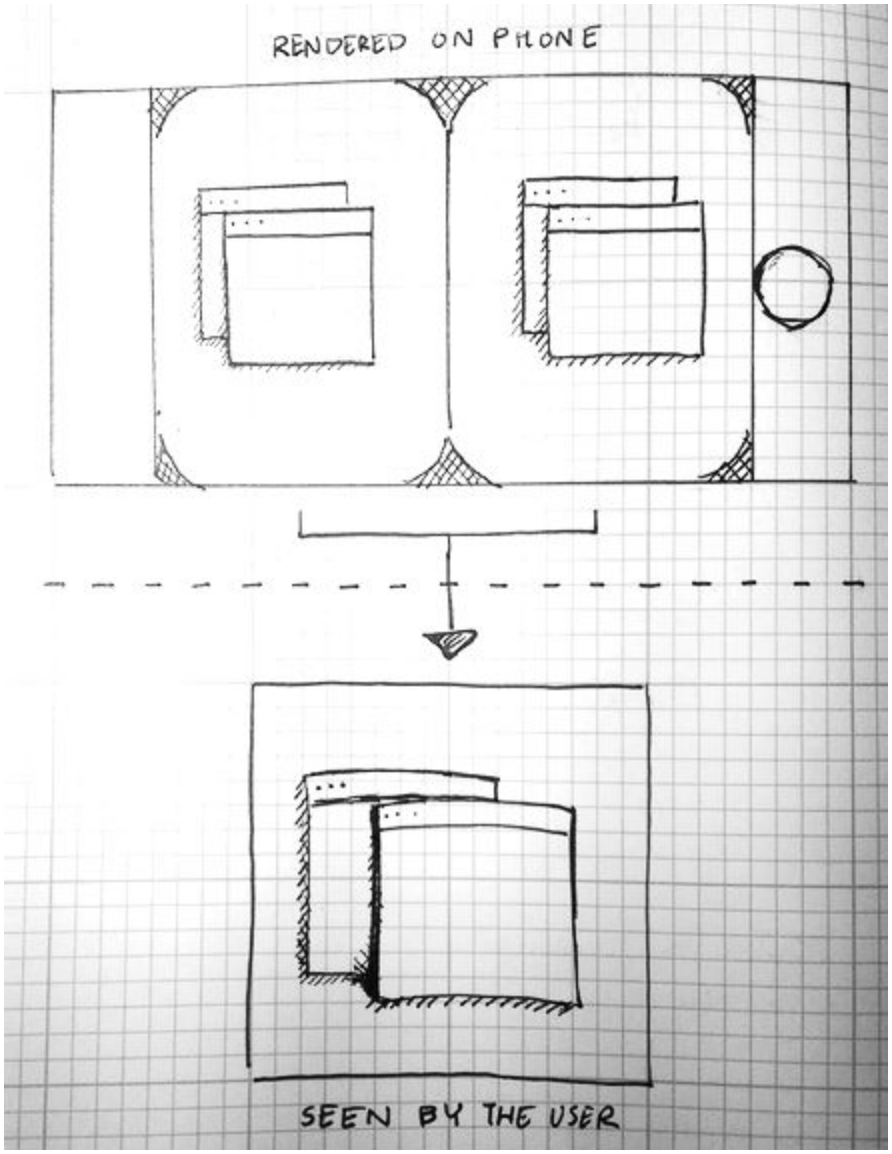
In the **Gestures** window, the user will see information corresponding to hand-tracking. He or she will be able to enable/disable hand-tracking, start a calibration/re-calibration process, and more. As this is a preliminary specification and we have not started designing the implementation, the contents of this window will most likely be added to.



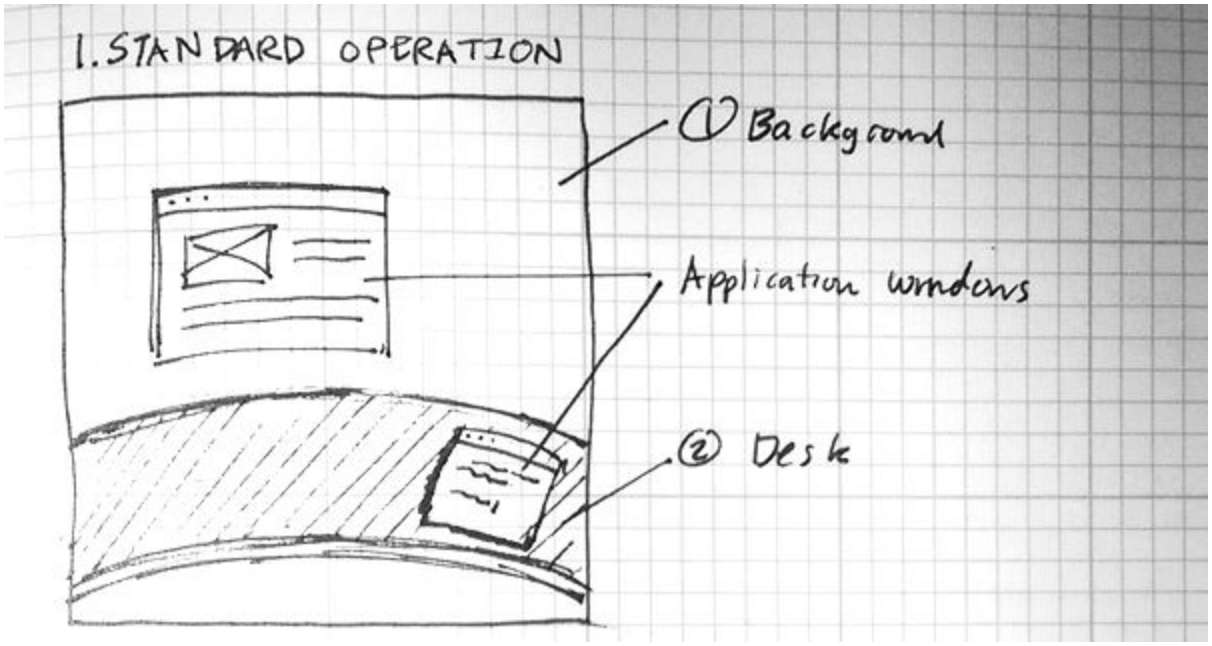
The **Options** window will serve as a general information and control resource, where you'll be able to view 'about' information on Expanse, as well as server details/debug information.



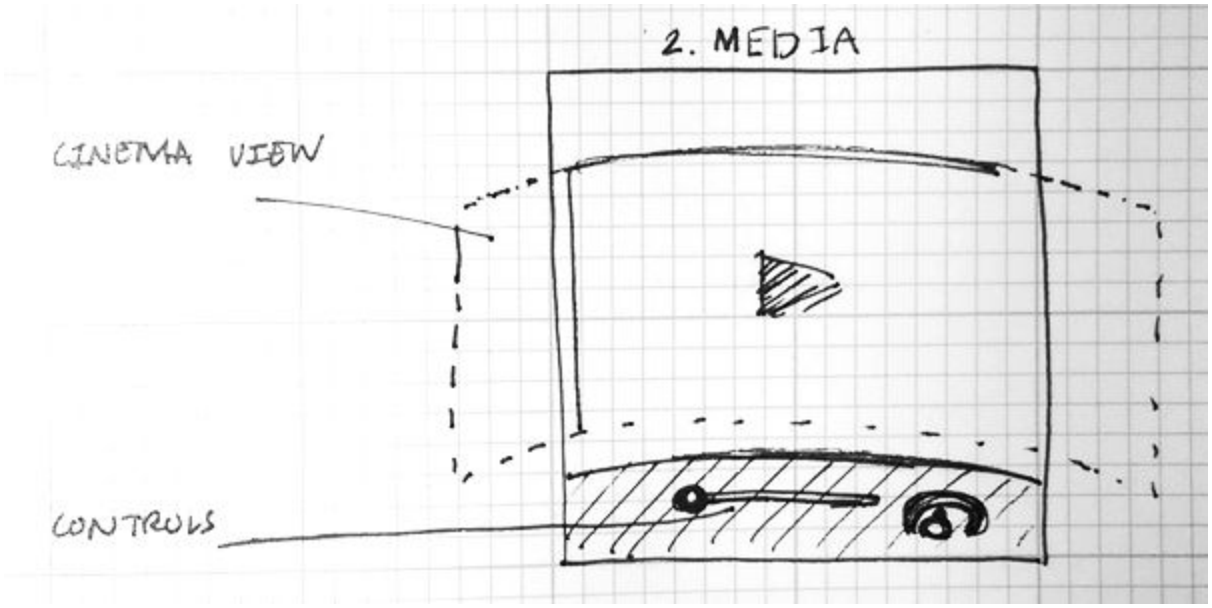
The **interaction interface** is where the user will be spending the majority of his or her time in. This is what will be displayed in the HMD. On the phone, it will be rendered as *two* separate, ovular rectangles, each containing a slightly different view into a 3D scene. These two rectangles will both essentially be looking at the same thing, but slightly laterally 'off' from each other - sort of like if you have two adjacent cameras trained on a single object. The images from each camera would be close - but a little offset, which is necessary to create a realistic, stereoscopic effect that fools the user's eyes. The Google Cardboard has lenses that focus on the rectangular images and refract them into our eyes, and from there the brain interprets them as a single image in a fairly realistic 3D realm.



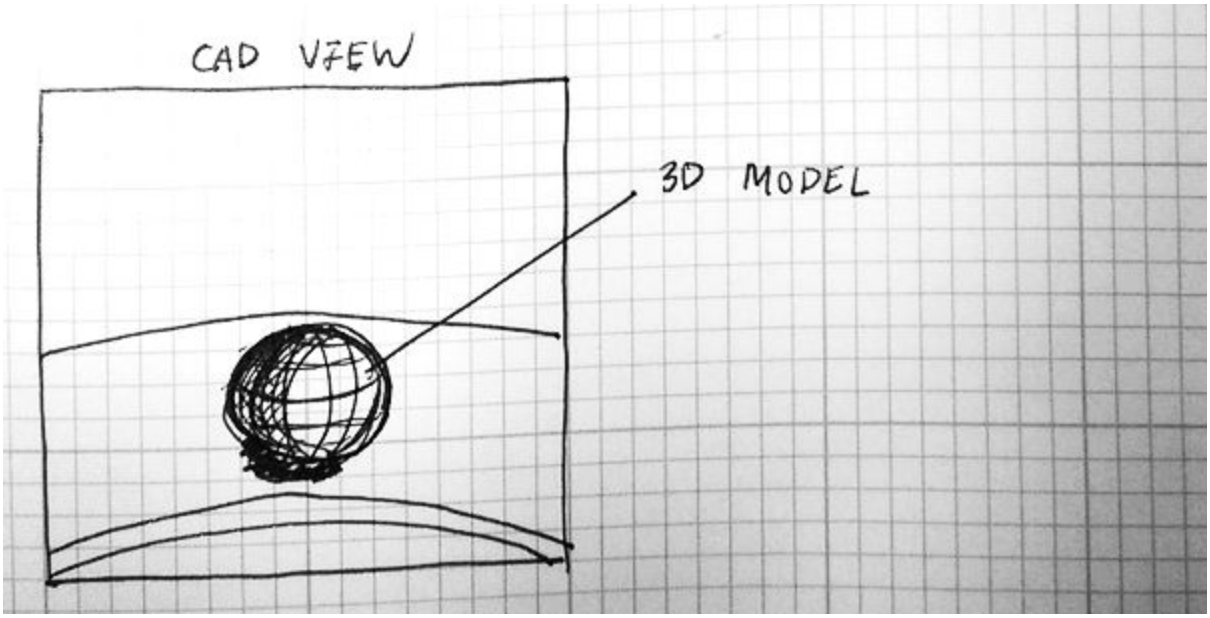
This interface will have three major display features: **standard**, **media** and **CAD**, and these display features are closely linked to our different use cases. They are explained later below.



In general, the interaction interface will be subdivided into three spatial areas - a **desk**, a **background** and **free-space**. The background area is mostly just negative space that fills the user's vision that does not contain a particular object, or anything in general. The desk is a table-like virtual surface that users can move applications to and on which a user can manipulate objects. Free-space is the area between the background, desk, and user, which will host the virtual windows and other virtual objects. Virtual windows will appear as exceedingly thin rectangles. Users will be able to drag and drop, close, and resize these windows according to the User Stories established in part I.



The **media** feature will be a special view for full-screen video player applications. A giant, slightly-curved window will be shown to the user, just like a cinema or a large TV. Controls will appear on the desk for volume and place, and the user will be able to interact with these using the hand-tracking system.



Finally, the **CAD** view feature will let you render arbitrary low-poly objects in the virtual workspace. Using the hand-tracking, a user will be able to rotate, scale, and reposition these objects.

B. Effort Estimation using Use Case Points

UCP = (UAW+UUCW) * TCF * ECF

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-4	**Visualizing 3D models**	10
UC-5	**Hardware Input**	10
UC-6	**Open/Close**	5
UC-7	**Adjust Brightness**	5

UC-8	**Enable/Disable Gesture Input**	5
UC-9	**Allow Access for Multiple Users**	15
UC-10	**Recalibrate gesture controls**	10
UC-11	**Allow Remote Access**	10
UC-12	**Save/Load**	10

UUCW = 120

Unadjusted Actor Weight (UAW)

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

UAW = 9

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

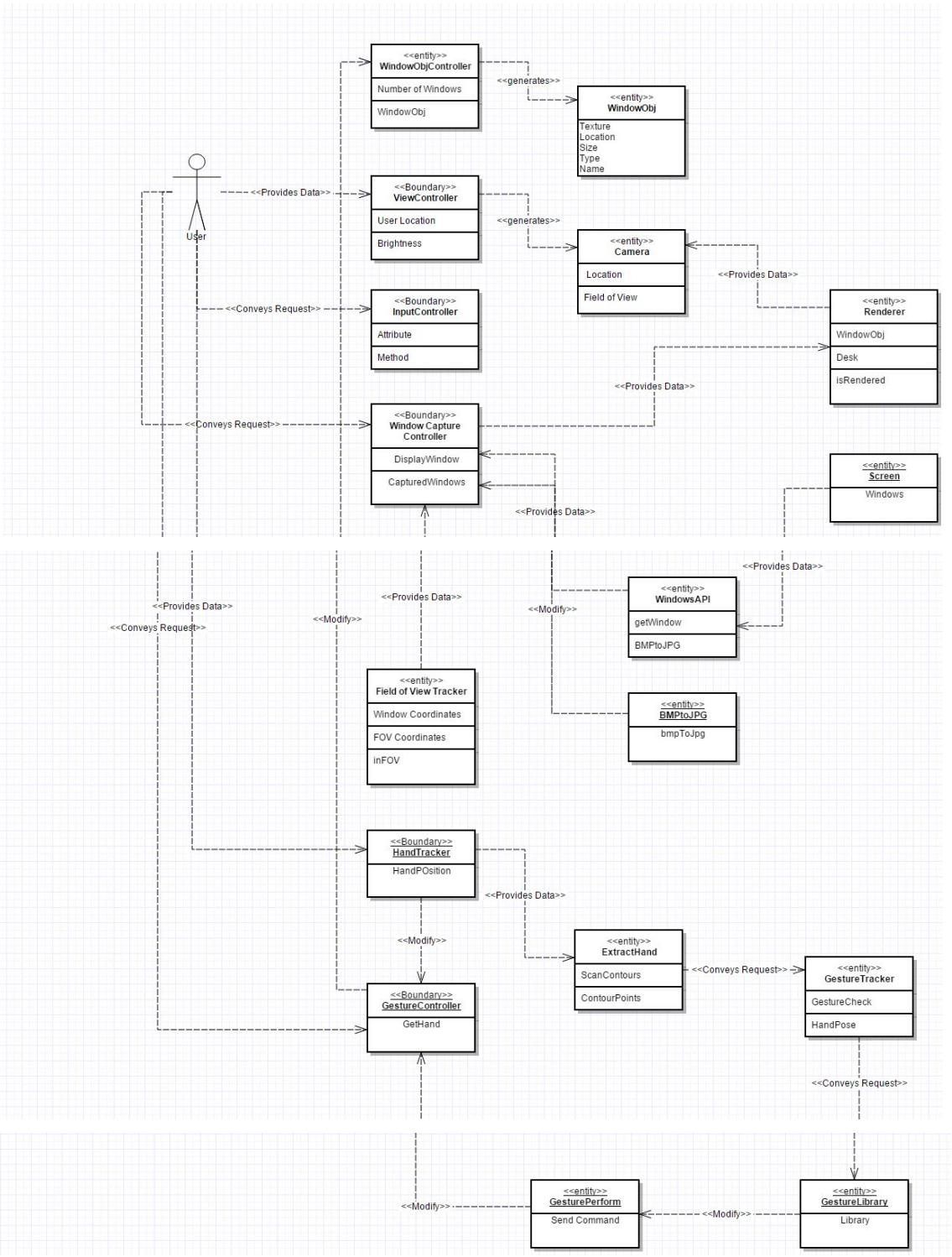
ECF	**Environmental Complexity Factor Weight**	**ECF perceived complexity**	**EF**
Familiarity with development process used	**1.5**	**3**	**4.5**
Application experience	**0.5**	**3**	**1.5**
Object-oriented experience of team	**1.0**	**3**	**3**
Lead analyst capability	**0.5**	**0**	**0**
Motivation 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

UCP = 104.9
PF = 28
Estimated Duration = 104.9 * 28 = 2938 Hours

1. Domain Modeling

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		ViewController
RS-4: Virtual camera that mimics user view in virtual space; sends visual data to user		Camera
RS-5: Handles user input obtained		InputController
RS-6: Renders windows given to it		ViewController
RS-7 Initiates virtualization of windows		User
Window Capture (Interface group)		
RS-8: Initiates window capture		User
RS-9: Decides which windows to capture		User
RS-10: Captures window textures and sends them to the render server		WindowCaptureController
RS-11: A physical object that displays the GUI to the user. Windows		Screen
RS-12: Facilitates the operation of each controller.		Windows API
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.		WindowCaptureController
RS-14: Keep track of window center coordinates and window size, center of the user's FOV is, and FOV size. Send this to		FOV tracker

window capture controller.		
RS-15: Convert the BMP from the windows API to a smaller jpg file.		BMP to JPG
Gestures		
RS-16: Initiates gesture control and gesture input		User
RS-17: Manages the gesture control objects and initiates hand acquisition algorithm		GestureController
RS-18: Finds the contours of the hand and detects contour hull and defect points		ExtractHand
RS-19: Observes current hand positioning and signals when hand performs relevant gesture		GestureTracker
RS-20: Relays the relevant user/computer action associated with a gesture and maintains execution of a single gesture at a time		GesturePerform
RS-21: Stores all performable gestures and acts as a reference to user		GestureLibrary
RS-22: Relays current position of hand as a single object in space		HandTracker
Server		
RS-23: Serves the app page with the		AppViewer
RS-24: Relays the window images taken from the Window Feeds subsytem		FeedStreamer
RS-25: Receives remote event clicks from the app running on the client and sends them to the application windows		WindowController
RS-26: Relays hand and gesture data from the Gesture recognition system to the app client running on the HMD		GestureCommunicator

ii. Association Definitions

Concept Pair	Association Description	Association Name
WindowObjController<->WindowObj	Controller creates WindowObj and passes textures to windowObj	generates
ViewController <-> Camera	ViewController places camera into scene, camera then captures the scene	generates
User<->ViewController	User is located; viewcontroller location and creates Camera	Provides Data
User<->InputController	User requests to begin keyboard and mouse or gesture control; InputController generates the appropriate objects such as GestureController	Conveys Request
Renderer<->Camera	Renderer renders objects in the scene that the camera captures; camera's field of view determines which objects to render	Provides Data
InputController<->GestureController	InputController indicates to GestureController that user wishes to start gesture tracking	Conveys Request
Screen<-> CursorCaptureControroller	CursorCaptureControler indicates if the user clicks or just moves the cursor; If there is a click, send the request to server	Conveys Request
User <-> WindowCaptureControroller	Initiate window capture request	Conveys Request
WindowCaptureControroller <-> FOV tracker	Sends coordinates of the current FOV and open windows in the FOV to the Window Capture Controller	Provides Data
User<->WindowCaptureController	Further filters the windows to be rendered by a manual user selection	Conveys Request/Modify

WindowCaptureController<->Renderer	Sends window textures to the renderer	Provides Data
WindowsAPI<->WindowCaptureController	The API provides the tools to capture the window texture data.	Provides Data
Screen<->Windows API	The API generates data from window textures on the screen	Provides Data
BMPtoJPG<->WindowCaptureController	Compress the textures to a JPG file format before sending to the renderer	Modify
User<-> GestureController	User presents hand which is recognized by gesture controller	Conveys Request
User<-> HandTracker	Track possible movements as user continues to move hand	Provides Data
HandTracker<-> GestureController	Sends updates regarding hand position to maintain accuracy	Modify
HandTracker<-> ExtractHand	Communicate intricacies of hand position after finding overall position	Provides Data
ExtractHand<-> GestureTracker	Identifies possible user gesture inputs from tracked hand movement	Conveys Request
GestureTracker<-> GestureLibrary	Matches signal from GestureTracker to stored gesture to begin execution	Conveys Request
GestureLibrary<-> GesturePerform	Relevant actions linked to gesture are executed	Modify
GesturePerform<-> GestureController	Executed actions update gesture control objects before hand acquisition resumes	Modify
GestureController<-> WindowsObjController	Updates to gesture control objects update the windows within the user's workspace	Modify

iii. Attribute Definitions

Concept	Attributes	Description
WindowObjController	NumberofWindows	Total number of windows currently rendered
	WindowObj	Tracks window objects for creation and deletion
ViewController	Camera	Keeps track of camera
	User Location	Where the user is located
	Brightness	Adjust brightness of scene
Camera	Location	Location of the camera in virtual space
	Field of View	How much of the scene is being shown
WindowObj	Texture	The screenshot texture that are being placed into the window
	Location	The coordinates of the window in virtual space
	Size	The size of the window object
	Type	Type of application, video, web,3D model.
	Name	Name of application
InputController	Start Gesture	Begins Gesture Controls
	Start Standard	Begins standard keyboard and mouse controls
	End Gesture	End Gesture Controls
Renderer	WindowObj	Renders window object properties
	Desk	Renders virtual desk
	isRendered	Determines if object is rendered
WindowCapture Controller	displayWindow	Allows user to select windows to capture or not capture
	capturedWindows	Data of what windows are to be captured.
Screen	Windows	Raw data on what is on the computer monitor display

Field of View Tracker	Window Coordinates and size	Data pertaining to where windows are located in the display
	FOV Coordinates and size	Data pertaining to where the user is looking
	inFOV	Returns the windows in the current FOV
Windows API	getWindow	Obtain the window texture
BMP to JPG	BMPtoJPG	Compress image file to prepare for sending
GestureController	GetHand	Initiate Hand Acquisition algorithm
ExtractHand	ScanContours	Scan hand for contours
	ContourPoints	Tracks countour hull and defects
GestureTracker	GestureCheck	Checks hand for known gestures; sends signal when gesture recognized
	HandPose	Tracks current hand pose
GesturePerform	SendCommand	Sends appropriate command based on recognized gesture
GestureLibrary	Library	Stores variety of hand gestures
HandTracker	HandPosition	Tracks and updates hand position

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
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.

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

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**.

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

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

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.

$$cos(\theta_S) \ = \frac{\overline{A} \bullet \overline{B}}{|A||B|}$$
$$\theta_S = cos^{-1}(\frac{\overline{A} \bullet \overline{B}}{|A||B|})$$

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:

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

where D is

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

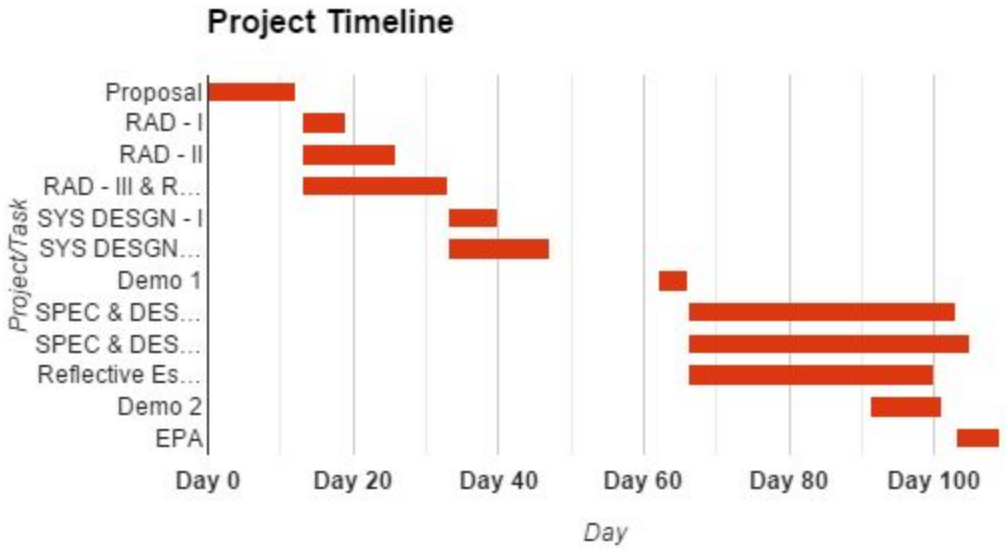
the first term is the clamped depth difference between the observed O and the hypothesis h, and the second term accounts for discrepancies between the color of the hand and the color of the visual tracking model. The formula kc

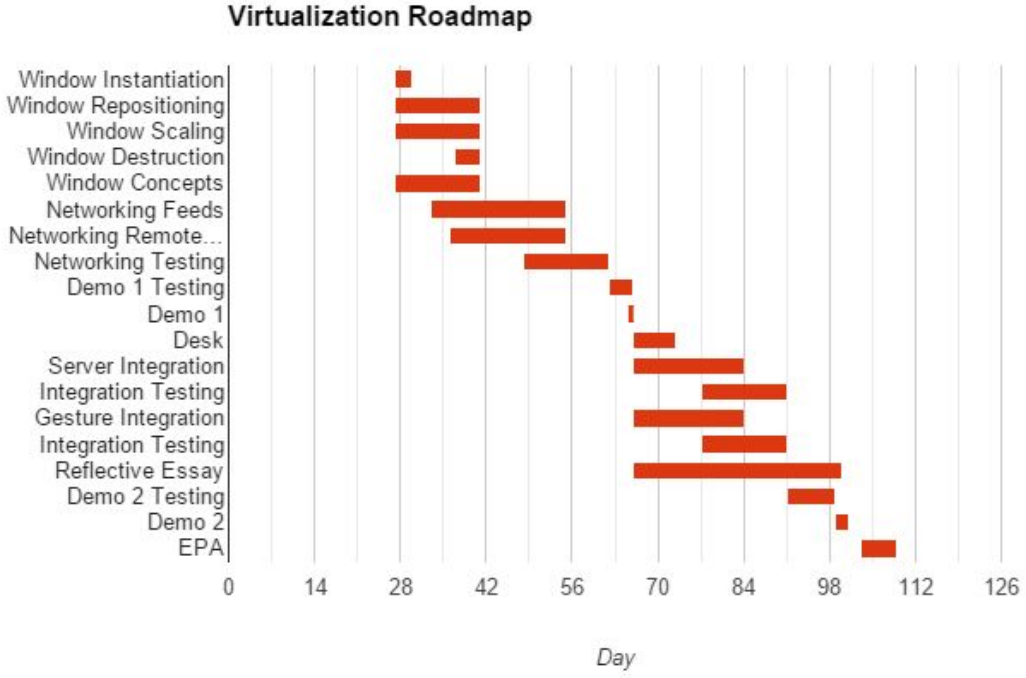
$$\mathrm{kc(h) = \sum_{p \in Q} -min(\varphi(p,h),0)}$$

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).

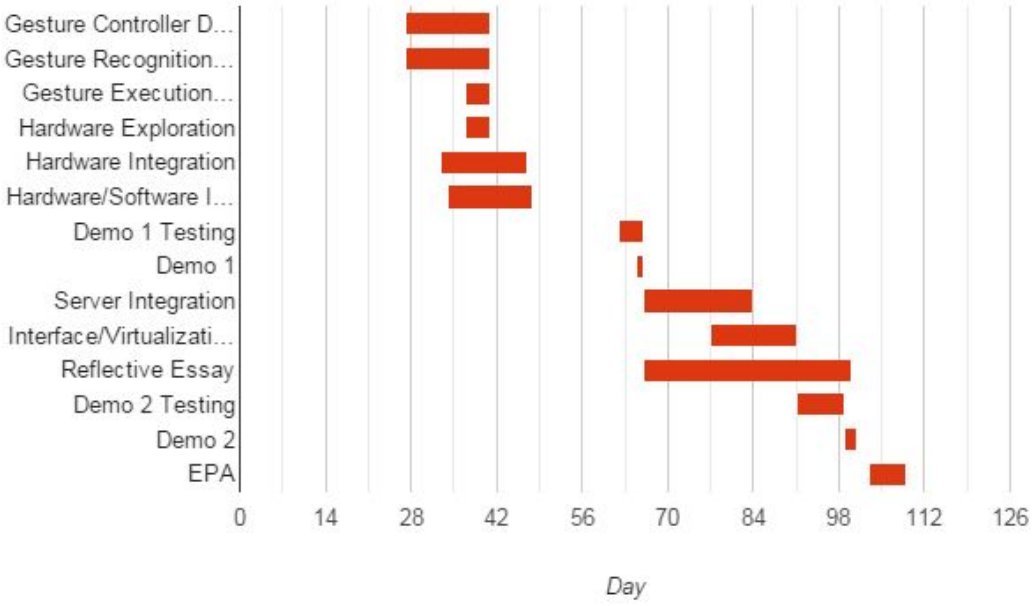
2. Plan of Work

a. Roadmap

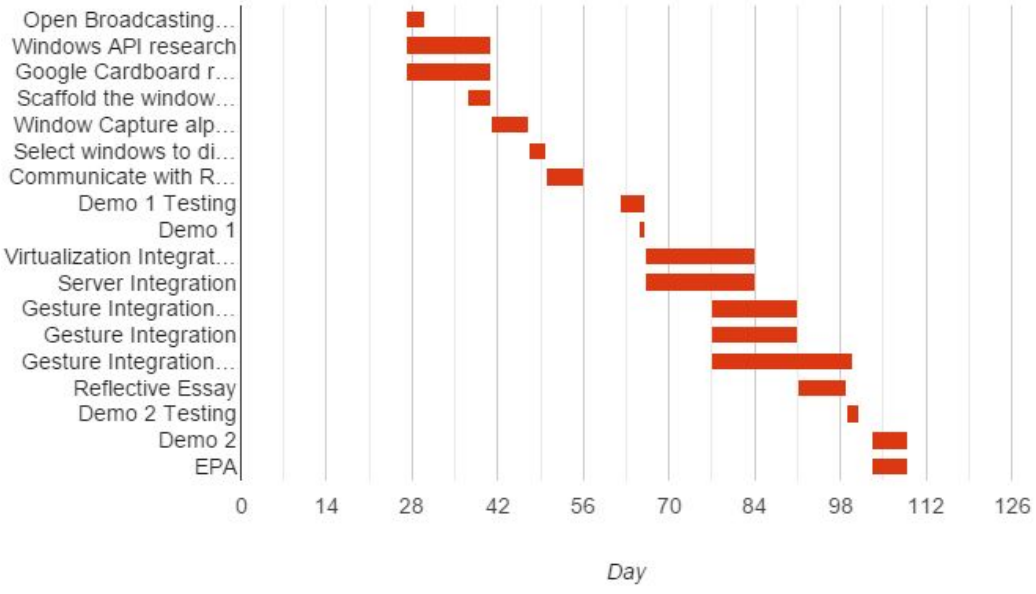




Gesture Tracking Roadmap



Window Interface Roadmap



Name	Team	Role & Responsibilities
Joshua Chan	Virtualization/ Rendering and Gesture Tracking	**Completed** - Scaffolded WindowObj(ect) and WindowObjController - Establish client app lifecycle and design - Research and selection of 3-D rendering library - Contributed to documentation - Create original perpendicular-window algorithm **In-Progress** - Implement Desk Object - Implement Window Manipulations **To Do** - Bare Server
Dmitriy Kozorezov	Virtualization	**Completed** - Coordinated with teammates to create best design of the object. - Researched Three.js and how it can help the project. - Contributed to documentation. **In-Progress** - Implement Window Manipulations - Implement Desk Object - Implement Window controls in virtual client. **To Do** - Continue research of Three.js and its applications in the virtual client.
Marc Tabago	Gesture Tracking	**Completed** - Contributed to documentation - Researched libraries for extended Microsoft Kinect functionality **In-Progress** - Implementation of GestureController **To Do** - Continue research of extended Kinect functionality - Begin hardware exploration/integration

Allen Tung	Interface	**Completed** - Contributed to documentation - Researched Windows API - Examined OpenBroadcaster Software **In-Progress** - Scaffold window capture automation - Researching Google Cardboard APIs **To Do** - Delegate work to Yue - Finalize the header files and function declarations to communicate with the render server - Agree on file format and exact outputs
Anthony Wong	Virtualization	**Completed** - Contributed to documentation - Researched Three.js API - Coordinated meetings between team members **In-Progress** - Implement Desk Object - Implement Window Manipulations - Design of various helper functions **To Do** - Coordinate with other teams to build a distributed network
Yue Yang	Interface	**Completed** - Contributed to documentation **In-Progress** - Researching various methods to accomplish tasks **To Do** - Coordinate with Allen to figure out work
Andrew Chan	Gesture Tracking	**Completed** - Contributed to documentation **In-Progress**

		- Continuing research of Kinect functionality **To Do** - Coordinate with Marc to continue delegation of work

3. References

Windows API:
https://msdn.microsoft.com/en-us/library/windows/desktop/ff486375(v=vs.85).aspx

Open Broadcaster Software:
https://jp9000.github.io/OBS/

Exploration of hand articulation tracking:
http://users.ics.forth.gr/~argyros/mypapers/2011_09_bmvc_kinect_hand_tracking.pdf