

Multi-Touch Table with Object Recognition

Codename Plank

Group 14:

Pete Oppold

Enrique Roche

Hector E. Rodriguez

Christopher A. Sosa

Table of Contents

Table of Contents	0
1. Introduction.....	1
1.1 Executive Summary	1
1.2 Motivation.....	2
1.3 Objectives and goals	2
1.4 Requirements and Specifications	3
2. General Project Research.....	4
2.1 Past Projects	4
2.1.1 NUI Groups Project LOCUS	4
2.1.2 TACTUS	5
2.1.3 Multi-touch Poker Table	7
2.2 Prototype	7
3. Touch and Fiducial Recognition Software System.....	10
3.1 Fiducials	11
3.2 Vision Libraries	13
3.2.1 CCV	14
3.2.2 reactIVision	16
3.2.3 D-touch	17
3.2.4 Bespoke Multi-touch Framework	19
3.2.5 Conclusion	21
3.3 Communication.....	22
3.4 Touch and Fiducial Recognition Software System.....	23
3.3.1 Top Level Program Flow	26
3.4.2 Reading from TUIO.....	26
3.4.3 Implementation	29
4. Showcase Program.....	31
4.1 Multi-touch User Interface.....	32
4.2 Graphics API.....	32
4.2.1 OpenGL.....	32
4.2.2 Microsoft XNA Game Studio	34
4.3 Design	34
4.3.1 Scenarios	35
4.4.2 Gestures.....	38

4.4.3 Graphics Structure	41
4.4.4 Code Structure	43
5. Computational Container System	46
5.1 Enclosure.....	47
5.2 Image Display	48
5.2.1 Short Throw Project.....	48
5.2.2 Long Throw Projectors	50
5.2.3 LCD Display	51
5.2.4 Comparison	53
5.3 Computer.....	54
5.4 Enclosure.....	55
5.4.1 Component Mounting	55
5.4.2 LED Frame.....	56
5.4.3 Component Cooling.....	57
6. Control System.....	58
6.1 Power	58
6.2 Pulse Width Modulation	59
6.3 Temperature Sensing	61
7. Image Recognition System	62
7.1 Types of Image Recognition.....	63
7.1.1 Frustrated Total Internal Refraction (FTIR)	63
7.1.2 Front/Rear Diffused Illumination (DI).....	64
7.1.3 Diffused Surface Illumination (DSI)	65
7.1.4 Interpolated Force Sensitive Resistance	66
7.1.5 Optical Imaging	67
7.1.6 Kinect (3D Imagining).....	67
7.1.7 Conclusion	68
7.2 Active Material	68
7.3 Diffuser	68
7.4 Abrasion Resistance.....	71
7.5 Camera	71
7.6 Optical Low Pass Filter.....	73
7.7 LEDs	73
8. Design Summary.....	74

8.1 Touch and Fiducial Recognition Software System.....	75
8.2 Showcase Software (weDefend).....	78
8.3 Computational Container System	84
8.3.1 Image Display	85
8.3.2 Computer.....	87
8.3.3 Enclosure.....	88
8.3.4 Enclosure Features.....	90
8.4 Control System.....	93
9. Future Work	94
10. Administrative Content.....	96
10.1 Roles and Responsibilities	96
10.2 Division of Labor.....	97
10.3 Milestones and Timelines	98
10.4 Software Development Model	99
10.5 Software Version Control	100
10.6 Budget.....	101
11. Owner’s Manual.....	102
11.1 Parts List	102
11.2 Transportation of Planck.....	103
11.3 Hardware Setup of Planck	103
12.3.1 Cleaning of Acrylic.....	106
12.3.2 Mounting of the Acrylic.....	106
12.4 Powering-up Planck	107
12.5 CCV	108
12.6 weDefend Simulation.....	113
12.6.1 Running weDefend	113
12.6.1 Using weDefend.....	113
Prep Mode.....	114
Action Mode	118
Debug Mode.....	119
Works Cited	122

1. Introduction

1.1 Executive Summary

Planck is a low cost multi-touch surface that uses diffused surface illumination technology to detect human touch and fiducials. It represents the next step in intuitive touch screen technology that expands the user experience and usefulness of multi-user applications. As large companies such as Microsoft, Samsung, Motorola, RIM, and Apple gravitate towards creating intuitive touch technologies for personal devices the feasibility of large multi-user touch surfaces replacing traditional input is becoming less science-fiction and more reality.

Planck is capable of using both human touch and fiducials to replace traditional forms of input used for home computers. It can be used in a variety of applications including entertainment, simulation and modeling, participation learning, teaching, and technical presentations. With the support of other systems, Planck can be used to create powerful high throughput multi-user systems that enable users to work together to achieve a common goal. Planck's aim was to offer a more intuitive input scheme that allows users to learn while doing.

Planck is a 40 inch screen that is fitted into a table structure that can be used by the average size person. Planck used a special mirrored particle acrylic that is illuminated by infrared light emitting diodes. An infrared camera was used to pick up diffused infrared light which served as input to the touch recognition system. The application image was displayed using a short throw projector onto a display mounted below the acrylic. The computer used to process all the recognition and application software was fast enough to respond to a touch in less than 10 milliseconds and the power supply was able to provide sufficient wattage to the whole system.

Planck is made up of four systems, two of them software and two of them hardware. The two hardware systems are the Computational Container System and the Image Recognition and Control System. The Computational Container System provides a structure for all the components of project Planck to rest in, or on. This is much like all of the computer hardware parts being mounted inside a computer case for structure, security, damage-control, and heat dissipation. While there is no novel implementation of the design of the enclosure in project Planck, there is a specific goal in mind. The enclosure was designed to replicate the surface of a smart-phone. The most important reason for this is to allow users to seamlessly switch from their smart-phone to comfortably using Planck. The computer in project Planck was also constructed of superior commercial grade electronics. This is necessary so that there are no latency issues in the transfer of the touch and fiducial processes throughout the software levels. Current multi-touch technology is used in small applications such as cell phones and tablets. These devices are also limited to touch based events. Planck utilized Diffused Surface Illumination technology. This technology is designed for large touch surfaces and has the capability of recognizing objects. Planck used a sheet of edge diffused acrylic, which when illuminated by infrared LEDs, can be tracked by a cost efficient

webcam. Plank also included a microcontroller, which will allow for the sensitivity and cooling within the device to be controlled. These technologies encompass the hardware that was realized in Plank.

The software systems in Plank are comprised of the Touch and Fiducial Recognition Software System and the Showcase Application weDefend. The Touch and Fiducial Recognition Software System reads input from the Image Recognition System's camera. A vision software library, CCV1.5, was chosen to interpret the images and extract meaningful touch and fiducial data. The gesture recognition module interprets the meaningful data extracted and uses algorithms to define gestures that are significant to the showcase application. The gestures are delivered in a shared data structure and are updated as they are received from the Image Recognition System. The last system is the Showcase Application named weDefend. The application is a model of military defense and personnel escort missions realized through a real time strategy application. It requires that users work together in order to accomplish military defensive objectives. The application also shows off the novel input that Plank is capable of and how traditional input is not required for complex applications.

1.2 Motivation

In recent years large software companies such as Microsoft, Apple, and Google have been moving toward highly intuitive, easy - to - use, touch based interfaces. However complex applications still require the use of traditional user input like mice and keyboards. Our goal is to remove the need for traditional input in favor of multi-touch gestures and real life objects that are in some way relevant to the application.

The aim of Plank was to create a novel tool set that can be integrated into several software solutions. Plank is a consumer level optical recognition table capable of controlling complex systems where the end user may use a combination of their hands and supplementary objects. Plank will be of sufficient size to accommodate multiple users simultaneously.

Plank provides an intuitive multi user environment via a multi-touch table. It expands the diversity of applications a touch device is capable of due to its rich set of possible inputs. The multi-user experience can be used in applications related to entertainment, simulation, business presentation, and many more. The ability to use fiducials that are attached to real life objects also adds to the user experience allowing users to make both a tactile and logical connections to a command.

1.3 Objectives and goals

The primary goal of Plank was to remove the need for traditional input such as mouse and keyboard. We also wished for the table to accommodate multiple users simultaneously. To do this, a toolset would need to be created that is a combination of software and hardware components. The table would need to be of sufficient size to have

multiple people using the table simultaneously. Lastly, another piece of software would need to be created that would showcase the toolset.

1.4 Requirements and Specifications

The specifications and requirements for Plank were created to fulfill the objectives. A table would be created. The requirements were that it would be at least 40 inches diagonally to allow for multiple users and be sturdy enough to support the weight of several users that may be leaning against it. The table will act as both the input device as well as the display. It was decided that Diffused Surface Illumination would be used. This required the inclusion of infrared LEDs as well as certain types of acrylic. A full list of requirements and specifications are as follows:

Specifications

- System will be comprised of Diffused Surface Illumination technology (DSI).
- A special mirrored particle acrylic.
- IR LED's to illuminate acrylic.
- IR camera to detect 'blobs' on the acrylic surface.
- Image display to transmit desktop image to a display below the mirrored particle acrylic .
- Table to enclose all hardware.
- System should be able to read fiducials .
- The acrylic screen will have an LED spaced at least every inch encompassing the border.
- Usable system screen size of 40 diagonal inches.
- The border between the screen and acrylic edge shall be no greater than three inches.
- The enclosure will be tall enough for an average sized user to use standing up.
- The enclosure's top will resemble that of a touch screen smart-phone.
- The user will have to wait minimally for the system to cold boot.
- Multi-touch surface will have a design resembling that of a smart-phone. This includes edge-to-edge acrylic and all borders in the color, black.

Requirements

- Computer
 - The projector shall display an image onto the projection surface
 - The computer shall be fast enough to display the associated object from the finger touch in less than 10 ms
- Enclosure
 - The internal temperature of the enclosure shall not exceed the highest operating temperature of any of the enclosed devices
 - The enclosure must be able to fit through a standard doorway
 - The enclosure must be big enough to house all hardware items

- The enclosure and all hardware (the assembled product) must be lighter than 100 pounds so two people can carry it.
- The enclosure must have a doorway to access all internal parts
- The enclosure shall not be constructed with a height higher than 4 feet.
- The enclosure shall be able to support up to 400lbs of weight.
- Blob/Object Detection Software
 - The system shall detect fiducials.
 - The software shall detect a finger touch.
 - Fiducial recognition and output should have a latency of at most 0.5 seconds
 - System shall recognize fiducials 2x2 inches
- Image Recognition System
 - The surface touch system shall detect fingers and fiducials in an indoor, dimly lit environment
 - Power
 - The power supply must provide enough wattage to power the system (LED's, Computer, camera, projector)
- Showcase Application
 - Software application shall be able to drag multiple objects simultaneously
 - Application must run on a Windows platform
 - The application shall run at a HD resolution of 1280x720.

2. General Project Research

Before choosing our project, and as reference material for our project, we researched other previous projects and consulted others in the community who had previously built similar tables. Through this research, we discovered new considerations on how to implement our project, as well as the need for us to create a prototype.

2.1 Past Projects

Many types of multi-touch surfaces have been made by hobbyists and large companies alike. Below is a few of the projects more relative to project Plank and Group 14. These projects were created by hobbyists on NUI Group, a PHD candidate, and a prior UCF senior design team.

2.1.1 NUI Groups Project LOCUS

Toby's multi-touch gaming table, Locus, was created for designing multi-touch roll playing games for his personal use. He works at the University of Boston fulltime and is an active member on NUIgroups forums. Toby has created several tables and applications. Of special interest, is his newest 42" DSI Gaming Table, and his role-playing game.

Locus is a 19" tall 'coffee' table. There is an upper ridge around the perimeter of the display that allows for accessories to be attached. These accessories include cup holders, craft tables, and dice rollers. Locus uses a hybrid Diffused Surface Illumination(DSI) technology. 850nm IR RibbonFlex are used to illuminate a 43"x24", 10mm thick, piece of ACRYLITE Endlighten XXL. The LED ribbons are bought as a whole unit and require no work by the designer. They are mounted directly to metal L channels, which are wrapped in aluminum foil to heighten illumination using reflectivity. These channels are then secured around all edges of the acrylic waveguide. The edges are polished for effective illumination. They are wired up in array fashion and powered by a 12v power supply. Locus uses a Hercules Dualpix HD 720p Webcam to detect blobs off the illuminated acrylic. The IR filter on the camera was removed to allow the camera to detect IR light. The Dualpix has a resolution of 1280x720 and runs CCV at 50-60 fps. It is a CMOS camera that uses USB connection and is outfitted with a distortion free wide angle lens.

The projector chosen is an InFocus XS1 short-throw projector that produces a 4:3 aspect ratio. The native resolution is 1024x768 and can produce a 60" display from 32" away with a vertical offset of 3.6". A sheet of Evonik 7D006 acrylic is used for the rear-projection material. Toby was originally using the Evonik 7D512 as the rear-projection material but had issues with detecting fiducials with that particular model. The projector is mounted horizontally. Its projected image bounces off one ACRYLITE Reflections 0A000 X1 acrylic mirror and onto the rear-projection material. Lastly, a Toshiba M200 laptop, running Windows 7, is chosen to run the application for Locus.

2.1.2 TACTUS

TACTUS is a multi-touch table designed fully by Dr. Paul Varcholik at UCF's FIA institution. TACTUS was designed with the goal to run a wide variety of applications, whereas Project Planck's table was designed to run one application in mind, the showcase simulation. TACTUS has its own custom blob detection software written by Dr. Paul Varcholik. Several applications were run successfully on TACTUS such as InkDemo, SurfaceSimon, and the TACTUS mouse emulator. InkDemo is an application that "allows for single-touch, pen-based/writing-style interaction through a multi-touch surface." SurfaceSimon is a multi-touch application that emulates the electronic game by Milton Bradley, Simon. The mouse emulator allows the user to play applications that regularly require a mouse with finger touches instead. TACTUS is also unique in that it supports the use of pen interaction with the display.

TACTUS is a 36" tall table with a 6" border around the display to act as an armrest. TACTUS uses Frustrated Total Internal Reflection (FTIR) technology. 32 Osram 485 LED's are used to illuminate a 32"x24", 1/2" thick piece of acrylic. The Osram LED's emit a wavelength of 880nm. They are mounted directly inside 5mm depressions that are drilled every 1" around the perimeter of the acrylic waveguide. This prevents the need for polishing the edges of the acrylic. They are wired up in array fashion and powered by a 12v power supply. TACTUS uses a Microsoft LifeCam VX-6000 to detect blobs off the illuminated acrylic. The IR filter on the camera was removed to allow the camera to

detect IR light. The LifeCam has a resolution of 1280x1024 at 30fps. It uses a USB connection.

The projector chosen is a Mitsubishi XD500U-ST short throw projector that produces a 4:3 aspect ratio. The native resolution is 1024x768 and can produce a 60" display from 33" away. A sheet of Rosco Gray, 7mm thick PVC, rear-projection material is chosen to diffuse the projector's image. The diffuser is mounted above the waveguide. The projector is mounted vertically and projects the image directly upon the diffuser without the use of any mirrors. A compliant surface was constructed from 1mm thick SORTA-Clear 40. This is a translucent silicone rubber that acts as a coupling material between the finger and acrylic. This improves the effectiveness of FTIR. Lastly, a MicroATX computer was used for operating the multi-touch surface software.

The TACTUS software library, Bespoke, is the core of the blob detection software. The stages of the software are shown below in Figure 1. The first stage is the input of the raw camera image from the LifeCam. This image is then rotated or flipped to match the orientation of the projector's image. Next, the image has the background filtered out. The background image was constructed while the system was dormant. The present image subtracting the background image leaves the blobs that should be tracked. The next stage converts the 24bpp color image to 8bpp grayscale. The threshold stage further isolates pixel values to black or white. Following the threshold stage is the blob detection stage. The Blob detection stage is very important in that it actually groups the white pixels (ON) and classifies them as blobs. There is a filter that ignores any blobs that don't meet a minimum size. Next, scaling crops the image to reflect that of the projector's resolution. This deletes the border that might be useless because of extreme illumination. The Calibration stage adjusts to create a perfect point that is reflected on the projector, camera, and display surface. Finally, Point Tracking transforms this heavily modified image into an FtirPoint object that has many attributes associated with it. This is the final blob that will be sent to end-user application. It contains all the information an application would require such as a unique identifier, timestamp, bounding box, speed, direction, and duration.

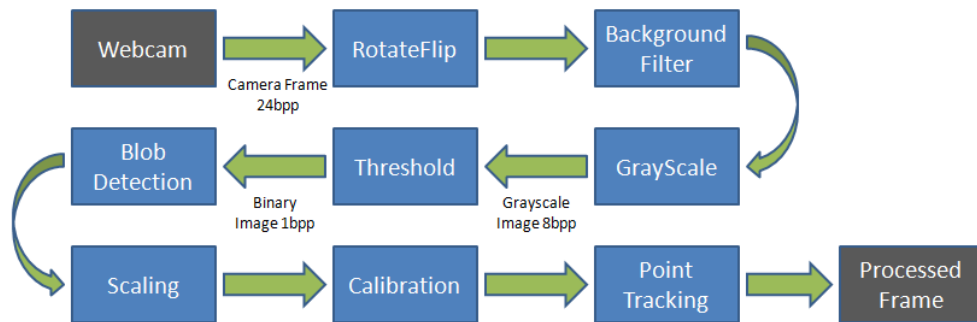


Figure 1 - TACTUS Image Processing Process

TACTUS incorporates the Bespoke 3DUI XNA Framework and the Open Sound Control Library (OSC) for use in the creation of multi-touch games and simulations. Games/simulations were created on this platform such as SurfaceCommand, InkDemo,

Waterfall, and a mouse emulator. The application most related to Planck's application is SurfaceCommand. SurfaceCommand presents a 3D map that can be explored with a group of spaceships. It is a proof-of-concept application that shows the effectiveness multi-touch can have with a multiple object aerial game. The spaceships can be controlled using multi-touch commands, such as a finger press. The 3D map view can be zoomed in or out with a 'pinch' command or moved using a finger swipe.

2.1.3 Multi-touch Poker Table

The Multi Touch Poker Table (MTPT) is a multi-touch table designed by previous UCF students. The table's main purpose is to run a game of Texas Hold 'em Poker for up to 4 people that incorporates the use of their iPhones into the game. The Poker table also showcases the possibility of creating a multi-touch table on a limited budget. The MTPT project uses much of the same design as TACTUS. As such only the details that are of interest will be outlined below.

Poker Table is a 39" tall table. MTPT uses FTIR technology. MTPT uses the Playstation 3 Eye camera, a very popular webcam on the NUIgroup community. The group of MTPT designed their own linear power supply to power the LEDS. The power supply also includes an intensity adjustment circuit using a 555 timer.

MTPTs showcase application, a game of Texas Hold 'em, was designed to interface with both the multi-touch table and each player's iPhone. The two cards the player's receive were sent to the iPhone instead of being shown on the table. This was done because of the complication that would have arisen if the player had to look at his cards from the table and risk other user's seeing his hand. Another application that was also designed to run in parallel with the game of Texas Hold-em is the Restaurant Menu. Restaurant Menu allows each player to input their food order and pass the menu on the screen from player to player with a swipe of a finger.

MTPT uses TouchLib as their blob detection software. TouchLib is used to calibrate their screen and pass the blob objects detected to their Poker Table application. These blob objects will be used as the source of user input for the application, just like a mouse. TouchLib refers to the blob object as TouchData. The attributes of TouchData can be found in **Error! Reference source not found.** below. MTPT used the QT C++ framework to create their application. Their application was written in C++ using the Microsoft Visual Studio IDE. PokerSource, a library that includes a set of rules for the game itself, is written in the programming language C. The MTPT TCP server is written in Python.

2.2 Prototype

The decision to create a prototype was to gain the knowledge needed to design our own multi-touch table and software application. Multi-Touch computing, and its software, is new to the entire group. It is important to fully understand the object detection software

before designing an application that requires its use. It would also serve a second purpose to be used as a test bed for future pieces of hardware and software. Many of the components that would be designed and purchased for Plank cannot be independently tested without additional components or a full functional board. This has the added benefit of giving us a reference point to compare our design against.

The prototype was designed in a way that many of the components used in it may also be used in the final design. An example of this would be the very similar acrylic used, as well as the use of the same method of detecting touches. With a working prototype we were able to install, load, and use our object detection software to interact with many of the open-source applications available today. With a more thorough understanding of the object detection software, we can effectively brainstorm and design our show-case application.

One of the main concerns with the prototype was to keep the costs down while not limiting the functionality of the prototype. First, rather than buy a computer to install into the device, we used one of the group's laptops to power the device. Second, we opted not to build a display screen into the design to keep costs down. A simple alternative is to use a standard external monitor from the multi-touch device or the monitor of the laptop. The user was not able to visually see where they are going to press on the screen until they actually conduct the press and it displays on the external monitor. This is a minor concern in the overall goals of the prototype. Lastly, the construction of the enclosure was built using medium-density fiberboard (MDF). While the wood doesn't have any visual appeal, it is strong enough and durable enough for the enclosure as well as being some of the cheapest wood you can buy. For example, a 4'x8', 3/4" thick, sheet of MDF is about \$24 compared to a 4'x8', 3/4" thick, sheet of Hickory at \$100.

The enclosure was designed using MDF. Dado cuts were implemented into the design of the enclosure to allow the sides to lock into each other. This allows for more stability of the enclosure, eliminates the hassle of gluing 90 degree angles, and ensures the box is built with the exact measurements intended. The overall dimensions of the box were 27" x 18" x 21". The goal was to allow the LED's and acrylic to be easily removable for maintenance and storage. There is a removable frame at the top to house the acrylic and LED's. A channel was cut in the frame to house the LED's. The LED's will be drilled into very thin pieces of wood that will rest in the channels. These LED frames will sit snugly up against the acrylic to effectively shine the IR light directly into the acrylic.

A sheet of EndLighten XL material was used as the active layer for the prototype. An additional piece of abrasion resistant material was purchased to protect this layer. The camera that was used was donated by a member of the design team. It was a Logitech Orbit webcam. This camera already utilizes a manufacturer supplied wide angle lens and contained an infrared block filter that was removed with minimal difficulty. Additionally a 3.5" floppy disk was used as a low pass filter. Figure 2 shows the completed prototype.



Figure 2 - Fully Built Prototype Enclosure

OSRAM 485P LEDs were used for the prototype. A number of problems were found when mounting the LEDs. The method of soldering LED leads directly was found to be very cumbersome and it was found difficult to keep the LEDs evenly spaced and un-rotated. 70 LEDs were soldered into 10 chains of 7 LEDs, which were then mounted at 1 inch intervals into the previously mentioned frames. It was found that these frames too drastically reduced the amount of light. Previous projects had utilized different methods of mounting which may increase the performance. The LOCUS coffee table surrounded the LED frame in aluminum foil to reflect stray infrared back into the acrylic. The TACTUS table drilled directly into the acrylic to mount the LEDs.

Even with the LED frame removed, it was found that the amount of illumination from the acrylic was unsatisfactory. The prototype did function at this point, but it only functioned in very low ambient light conditions. The image received from the camera was also unsatisfactory due to it requiring a significant amount of gain and exposure. This may not be directly a result of the LEDs. Because we have no way of testing parts individually, it is possible that this may be due to using an unsatisfactory camera or low pass filter. It may even be possible that the acrylic is preventing the infrared light from penetrating deep enough. The LED spacing was reduced to .5 inch, and at this level it was found that the brightness of the LEDs was sufficient enough to use the board in normal lighting conditions.

A pulse width modulation circuit was also created to drive the LEDs. This circuit was made from a 555 timer controlled by a potentiometer, and used a bipolar junction transistor to amplify the modulated signal across the LEDs. This functioned exactly as expected, but may not be used in Plank's final design. Because the brightness of the LEDs was deemed insufficient, and the spacing between the LEDs was reduced, the design for Plank changed to reflect that. This required that Plank use many more LEDs

and a much higher current. This current may be too high to use a similar transistor, and other transistors may not produce a high enough gain to drive the LEDs.

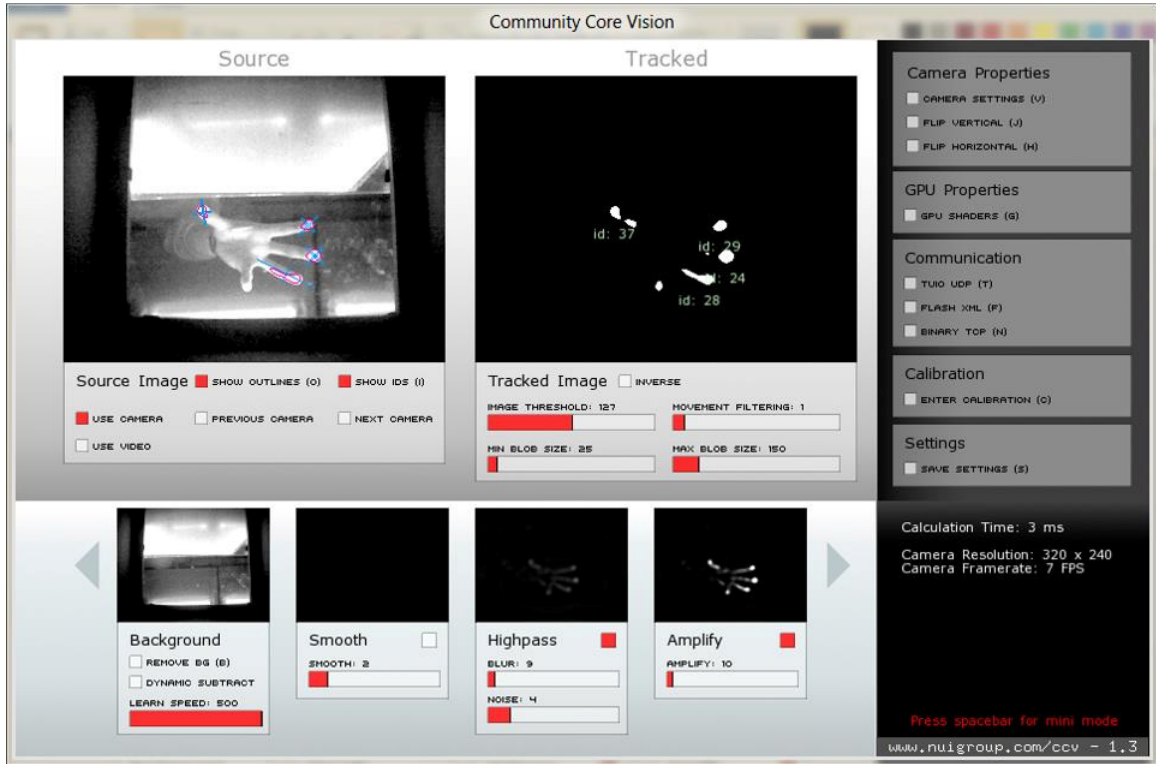


Figure 3 - CCV Output from Prototype



Figure 4 - reacTIVision Output from Prototype

3. Touch and Fiducial Recognition Software System

The Touch and Fiducial Recognition Software System takes the input from the IR camera and processes it to output position, orientation, speed, and identification information about finger touches (or blobs), as well as markers (or fiducials), on the surface. Figure 5 illustrates this idea.

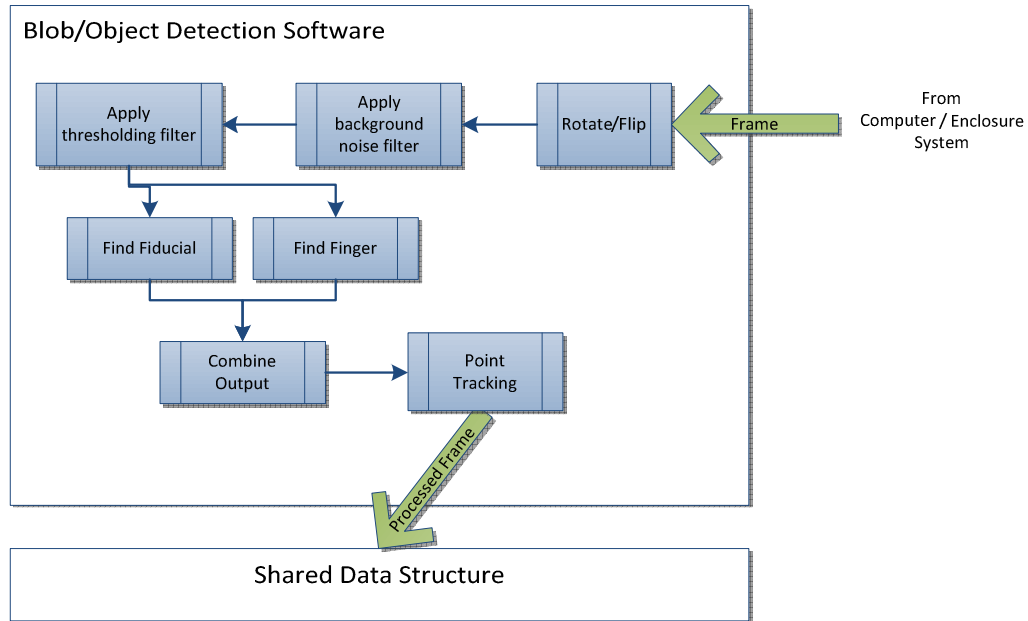


Figure 5 - Block Diagram of Touch and Fiducial Recognition Software System

At a high level, the system reads input from the camera as a video stream and converts each frame to a binary video image. It then runs vision algorithms on the image in order to find blobs. It finds the blob's position on the image and lists all blobs in a list. The software also scans the image frame for fiducials which match the patterns we're interested in and will be fully described later.

Planck used a vision library in order to process the incoming video and output it to the showcase program for display on the surface. Five different libraries were considered and will be discussed further on in this section. They are: CCV 1.5, OpenCV, D-touch, reacTIVision, and the Bespoke Multi-Touch Framework. Some of these libraries have native support for all the features we need and others don't. Some of them support touch/blob recognition and can determine position of a finger touch. Some of the vision libraries considered have support for fiducials, so a combination of libraries was considered in order to achieve the functionality set forth in the requirements and specifications. Another possible issue that will be addressed is whether we use one camera source to track objects on the surface or two. Given the size of the screen being used and the small size of the fiducials specified, the camera being used in Planck must have enough pixel density to see the fiducials. In the event the screen size is too large for the camera, stitching two cameras together may be a possibility we explore.

3.1 Fiducials

In their most basic definition, fiducials are markers located on an image that can be recognized by vision systems to be used in an application. Images containing fiducials can be used for various activities. Uses range from aligning objects in a photograph, using several fiducial markers to visually align a robot so that it may properly

manufacture a PCB board on an assembly line, marking objects in a video so that they may be tracked visually by software, serving as reference points in a scene so that other objects in the image may be measured against that reference, as well as many medical imaging uses (Erickson & Jack Jr).

Planck used fiducials in the form of patterns printed on paper that can be recognized by the vision library we choose. When an object with a fiducial applied underneath is placed on the surface, the vision library detected it. Each pattern has its own identifying ID number. Each unique ID number has a specific meaning to the showcase software. An ID number meant that a certain fiducial has been placed on screen. The showcase software then was able to output objects corresponding to that fiducial on the surface that interacts with the user.

The fiducials position, ID, and orientation were able to be acquired from the input video stream by the vision library chosen. This information was sent to the showcase software so that it was able to be used to interact with the user. The ability to capture orientation information from fiducials extends the capabilities of these markers. In current mainstream multi-touch technology, such as the iPad, the user only has the ability to move objects on-screen using gestures and pressing buttons with their fingers. Fiducials provide the opportunity to link specific objects on the screen with certain actions on the interactive surface. A fiducial may be placed on the surface prompting a specific output on the screen. The fiducial can then be rotated about itself in order to generate another meaning specific to that object or to the program. Specific fiducials prompted different output from Planck commensurate with the shape they are adhered to, creating a user experience that reaches beyond that of a simple touch interface.

With the proliferation of tablets and touch-screen phones, multi-touch surfaces are becoming more and more popular. Planck is essentially a very large touch-screen surface featuring true multi-touch, where true multi-touch is the act of at least two simultaneous touches. We're adding to the user experience by using objects, in the form of fiducials. The use of fiducials in Planck will further extend these trends in usability and make the user experience more natural and intuitive to use. The markers will be meaningful to both the operator of Planck as well as the software so that a seamless experience is created for the user.

Figure 6 has several instances of fiducials. All of these examples retain the core meaning of a fiducial, which is to transfer information in a coded format digitally. A common kind of fiducial seen is QR-codes, which are b in Figure 6. The kinds of fiducials used in Planck are Amoeba fiducials, which are d. CCV and Reactivision software are capable of recognizing Amoeba fiducials. These were chosen due to their speed of lookup.

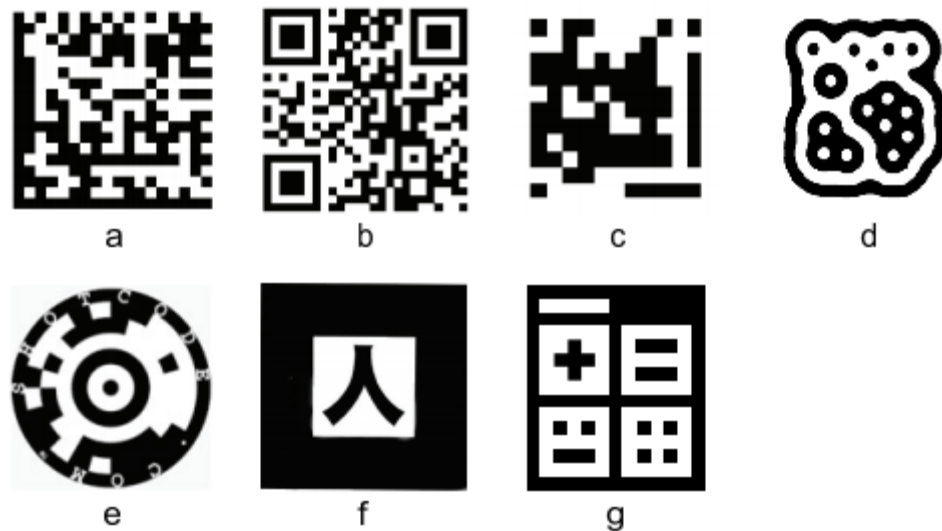


Figure 6 - Example of Different Types of Fiducials
(Reprint pending approval)

3.2 Vision Libraries

Five vision libraries were considered to be implemented in Planck. The decision of which one was picked hinged on several factors. There are many requirements that must be met when choosing the vision library. First, the library needed to offer at least the minimum latency in image processing as set in the requirements of project Planck. Second, it needed to run on the computer that was chosen, and support the camera chosen in the Image Recognition System. Third, the library needed to be able to work with Diffused Surface Illumination (DSI) technology. Fourth, the library needed to be reliably stable. Fifth, the library needed to support fiducials. Sixth, it needed a strong community would be beneficial if questions needed to be answered. Finally, the vision library needed to be versatile enough to communicate with the showcase application that was written.

After research, it was noted that all of the libraries under consideration would work with DSI. With that requirement checked off, the next two requirements that were focused on were that the library be able to detect both fiducials and finger touches. Some of the libraries can do both and some of them can do only one exclusively. Picking a library that does support both Fiducials and touches should have the added simplicity of our showcase application only having to interface with one less piece of software. While it isn't required that the input video library be able to recognize and transmit information about both fiducials and finger touches, it was weighted more highly if it did offer both.

Community support of the vision library is also part of the equation in picking our vision library. Two of the group members had exposure to general vision algorithms, but a knowledgebase regarding the software would simplify the usage of it in the system. Since

getting the touch and fiducial data isn't the sole aim of Planck, considerable amounts of time could not be spent on getting just that one subsystem working.

3.2.1 CCV

CCV is a community developed "open source/cross-platform solution for computer vision and multi-touch sensing" platform (NUI Group Community). CCV was made by an entity called Natural User Interface (NUI) Group Community. The group's interests lie in making open source user interfaces so that new "interaction techniques and standards can be developed that would benefit developers and designers throughout the world" (NUI Group Community).

CCV's website states that CCV can take a "video input stream and output tracking data (e.g. coordinates and blob size) and events (e.g. finger down, moved and released) that are used in building NUI aware applications (NUI Group)". This software library can interface with many different cameras and can connect with networking communications protocols such as TUIO/OSC/XML. The library also supports various lighting technologies, including DSI, among many others.

The system requirements for CCV as listed on their site are a Pentium 4+ (Recommended Core 2 Duo), 512MB+ RAM (Recommended 1024+), and a web camera for tracking. The Windows computer running the software must also have QuickTime and Visual Studios 2008 Redistributable x86 (Natural User Interface Group ~ X1).

Visually, CCV is dependent on having an infrared-lit surface for it to see touches and fiducials. These include diffused surface illumination, frustrated total internal refraction, diffused illumination, or Laser Light Plane.

In order to use this system, the software must be downloaded from NUI Group's repository. It is then unzipped and placed in a folder. There is an executable file that can be run. Once this file is run the program starts and the user is prompted to allow the program to be able to send/receive information through Windows' firewall. Since the program can use TUIO/OSC to communicate movement of objects, blobs, and fiducials, it must have access to send information through the network, even if it's using the network in the computer to communicate with another program on the same machine. Once the software is allowed access to the network or even without, the user is treated to CCV's user interface from which many different parameters can be tweaked.

CCV can communicate touch events in 3 different ways. It can use the TUIO protocol via port number 3333 to communicate these events, Flash XML to transfer the messages, or a Binary TCP protocol. TUIO is the same protocol used by both ReactIVision and D-touch to perform communication. TUIO was created by the team that made ReactIVision and was built upon the OSC protocol. OSC was originally created to transfer MIDI data from musical keyboards and other musical devices that can use the MIDI standard. It's a network protocol that stores the data in a packaging scheme and transmits it via

UDP/TCP. There are several data checks in place in the protocol to ensure that the data delivered does not become corrupted in transport.

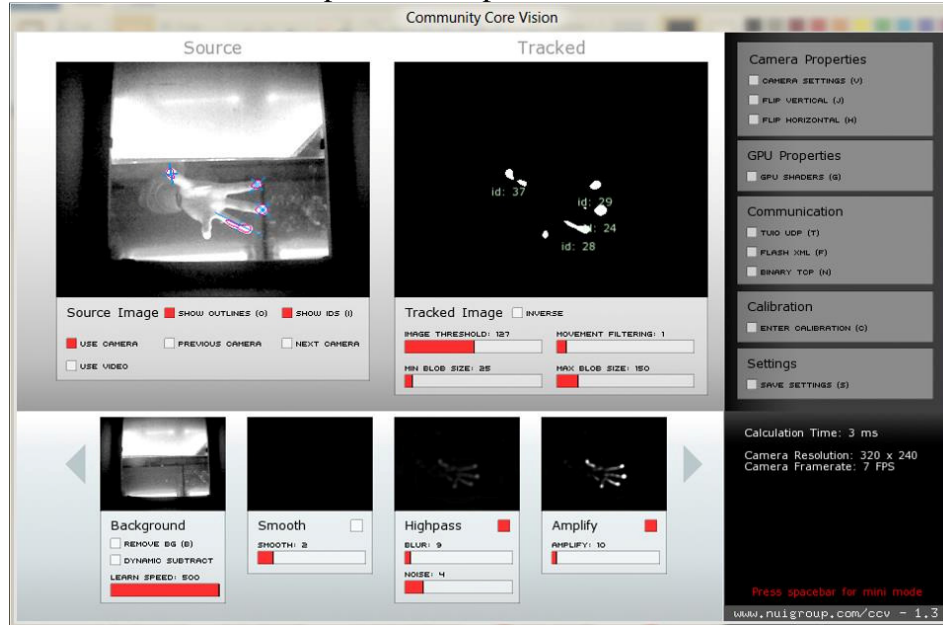


Figure 7 - Example of CCV Configuration Screen

This interface allows for the configuration of several different options. First, different input devices can be selected. Second, various filters can be applied to the incoming image. Also, the size of blobs/finger touches can be configured. Third, fiducial and object tracking can be turned on or off. Fourth, robust output can be generated for debugging purposes. Fifth, the vision software can be calibrated and settings loaded or saved. Lastly, image flipping horizontally and vertically can be configured, and various other software settings can be tweaked. The online manual states that we should use the following settings with this software and a DSI lighting technique:

1. Turn off the smooth and amplify filters.
2. Turn on the high-pass filter.
3. Adjust the high-pass blur and noise sliders until fingers are clear and distinct.
4. If blobs are weak, turn on the amplify filter to brighten them

Next, calibration must be run. Calibration will allow CCV to align touches detected by the camera in relation to where they are on on-screen. This allows the system to be more accurate in terms of aligning a touch to the screen from the user and the location in software that the touch is recognized. The next step of the setup process is to alter the .xml file that contains CCV's configuration data about camera "input resolution, frame rate, communication, video, and blob settings." The ports used for communication can be changed in this file. The maximum number of fingers to be tracked can also be set in this file. Once these configurations are completed, the manual suggests user run through a series of flash demos to test the communication (Natural User Interface Group ~ X1).

In the sparse documentation found for CCV regarding fiducials, it states that the tracking of Fiducials has been a supported capability since version 1.4. As of the writing of this paper, version 1.5 has been released with “optimized fiducial tracking (NUIGroup).” There are a number of forums that state that their tracking isn’t as optimized as the reactIVision library. Very terse testing was performed with a simple camera and no DSI surface and it was found that fiducial tracking of reactIVision fiducials (printed on a sheet of paper) were detected by the system. It was noted in one of the forums regarding CCV, that fiducials should be printed with a laser printer as the ink doesn’t reflect IR light. The terse testing on the camera showed that fiducials were being tracked on screen and their orientation was being detected.

3.2.2 reactIVision

reactIVision is an open source, multi-platform computer vision framework that was created for the fast and accurate capturing of fiducial markers and multi-touch finger tracking. It was developed at the University of Pompeu Fabra in Barcelona, Spain. Its primary purpose is to assist in the rapid prototyping of the music creation table, ReactTable. It serves as the primary sensor in this table.

No specific information regarding the computer requirements could be found on reactIVision’s website. ReactTable’s website contains requirements for the table. Since the performance needed out of Planck is similar to that of the ReactTable, these system specs should suffice for the purposes of project Planck. They are: Intel processor 2GHz or higher; 1GB free HD space; and 1 free USB 2.0 port.

Similar to CCV, the system requires an infrared-lit surface for the software to be able to get input from. Such include diffused surface illumination, frustrated total internal refraction, diffused illumination, or laser light plane. The software can be configured using a file entitled “reactIVision.xml” under the windows environment. A different name is given to the file in a Macintosh environment.

The fiducials used in reactIVision were their own field of study. The creation of these was based on the work by Costanza and Robison. They created the D-touch library which included a set of fiducials that could be optically recognized using common vision algorithms. The creators of reactIVision wanted to create a faster fiducial recognition scheme in order to incorporate it in their music table, reactTable. This requirement led them to consider new ways of tracking fiducials on infrared touch tables.

The team set about creating a new set of fiducials using genetic algorithms by altering the angles between concentric circles. They used graph techniques just as the D-touch library had incorporated, except their look-up dictionary could outperform the current D-touch library since they had created their own shapes using a genetic algorithm. Faster fiducial performance led to the required recognition speed. As stated in the setup section, reactIVision has the ability to read both the library of fiducials created for D-touch as well as the fiducials optimized for speed created for reactIVision.

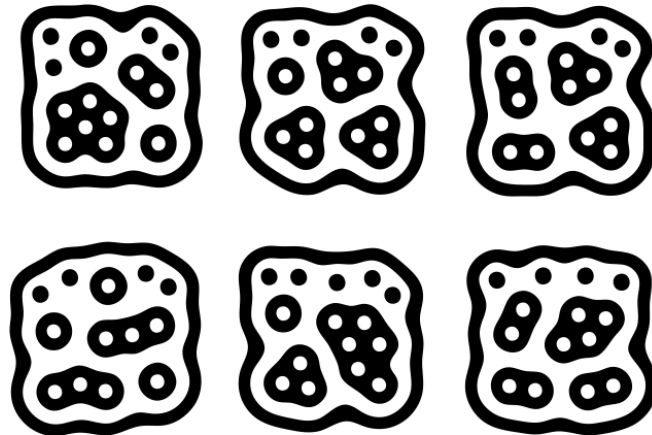


Figure 8 - Examples of reacTIVision Fiducials (Reprint pending approval)

Reactivision sends touch events through port 3333 and uses the TUIO protocol, created specifically for Reactivision, to send the touch events. TUIO was created for use in Reactivision and the ReactTable. Reactivision completely uses all of the parameters TUIO can contain. Other libraries may not capitalize completely on all of the robust features TUIO can handle since they were not designed with the use of TUIO specifically in mind.

3.2.3 D-touch

The D-touch visual marker recognition library, or libdtouch, is an open source marker recognition library that enables the construction of applications that can detect fiducials from a video stream. Any shape can be made into a fiducial. It just has to follow certain parameters necessary to the program in order to be recognized by the software. Orientation and position information is calculated relative to a grid that is placed in the background of the fiducial markers placed on-screen. (Costanza, Home Page, 2011)

D-touch is the name of an umbrella project that encompasses libdtouch; the library that can read fiducials. D-touch server, DTServer, is a standalone application that uses the libdtouch library to read fiducials and delivers the results through an output socket. Audio d-touch and a mobile version of d-touch are also part of the project. D-touch was created to be used in any type of camera system. It is not specific to infrared lit-type surfaces, such as project Planck. It can be used with a regular camera on a table top with ample lighting so long as the software can see the fiducials placed on-screen.

There are two ways of using the fiducial recognition abilities of d-touch. DTServer can be used, or the source code can be imported directly into a C++ program written by the user. The source code is available for download upon contacting the authors.

The system requirements for the use of DTServer require that the computer on which DTServer is run have a web camera and that it have either Macintosh or Windows operating system. On the receiving end, the software written to listen to the library must be able to listen to socket ports in order to receive data from DTServer.

DTSerfer refers to a file of marker ID's called "seq.txt". This file contains the objects that can be tracked by the software. The order of their appearance in the file determines the ID number that is returned by the software defining the object. The first marker listed in the seq.txt file is used as a calibration marker. Four of these markers are expected to be found on the four corners of the usable area. The position of all other markers will be calculated relative to these four first markers.

D-touch was created for the specific purpose of recognizing shapes and interacting with those. As such, its fiducial support is very comprehensive. The most distinctive feature of d-touch is that the markers can be shapes and figures that are meaningful to humans as well as the computer. The other fiducial recognition systems featured in this paper all contain symbols that aren't immediately meaningful to humans as can be seen in the figures below. D-touch allows for people to generate markers freehand whereas the other methods generate their markers algorithmically. Some examples of D-touch fiducials are included in Figure 9.

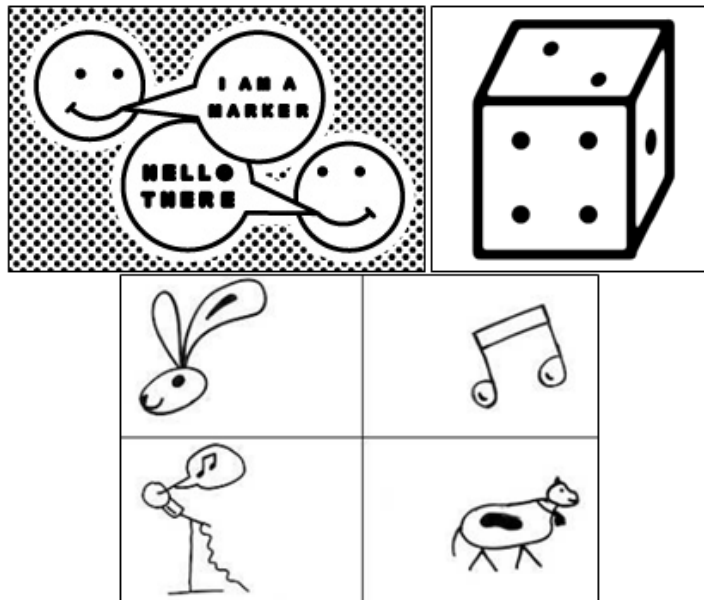


Figure 9 - Examples of D-Touch Fiducials (Reprint pending approval)

In order to create fiducials in d-touch, some simple rules must be observed:

1. A valid marker can be composed of a black region containing 3 or more white regions
2. At least half of these white regions must contain one or more black regions (Costanza, Home Page, 2011)

Since d-touch allows the creation of markers that can be meaningful to humans, different types of markers can be readily understood by users for different functions on their first use of the system. This can be quite handy when there are several inputs being perceived on one surface. Markers that belong to an individual or markers that carry a certain type of function can quickly be deciphered by novice users when the markers themselves resemble their function.

D-touch fiducials are recognized using topological features, not the geometry of the object: “Marker recognition is not based on shape, but on the relationship of dark and light regions (Costanza & Huang, Designing Visual Markers, 2009).” This approach allows the time spent recognizing markers to remain constant so performance doesn’t suffer the more markers are placed on the input video stream. The markers are decoded and stored into region adjacency trees. First the image is converted to binary black and white. From there, all adjacent pixels in an image are considered one region. The adjoining regions within these joined regions also get the same treatment and are considered their own region. The process continues on into the further embedded regions, not to exceed 3, until done. This process allows for any marker to be stored as a tree of adjacent regions.

The process of looking for fiducials at this point in the algorithm has been reduced to searching for trees that represent the marker. This type of problem is called “subtree isomorphism” and can be solved in $O(n \times m^{1.5} / \log(m))$ -time, “where m is the number of nodes in the target tree and n the number of nodes in the scene tree (Costanza & Huang, Designing Visual Markers, 2009).” The image below contains the adjacency tree along with the image it is comprised of. Region A contains regions e, f, and b. Region b contains regions c and d.

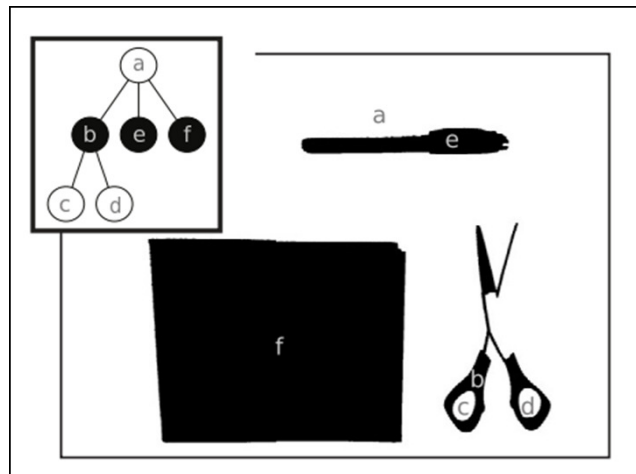


Figure 10 - D-touch Process for Determining Fiducials (Reprint pending approval)

Communication in libdtouch happens through a socket server. Any program that can handle sockets can implement a way of reading information from Libdtouch. On the d-touch.org website there are several examples in several languages that implement a way of reading information from the library. Any one of these can be used as a basis for how the showcase application can receive information from Libdtouch.

3.2.4 Bespoke Multi-touch Framework

The Bespoke multi-touch framework is another multi-touch software framework written in C# that is available for use. Bespoke was created by Dr. Paul Varcholik, a faculty member at Florida Interactive Entertainment Academy (FIEA). He created it for work on

his doctoral thesis entitled, 'Multi-Touch for General-Purpose Computing: An Examination of Text Entry'. Dr. Varcholik has built several multi-touch surfaces and has written many applications for them. Bespoke is BSD licensed, open-source and compatible with a wide range of multi-touch hardware including FTIR, DSI, and DI. A number of applications written by Dr. Varcholik come equipped with the Bespoke framework, such as a Windows mouse emulator and a 2D Ink/symbol recognition. Finally, an independent presentation layer and OSC network support for communication using unicast, multi-cast, and broadcast UDP/IP come packaged with Bespoke.

Bespoke has no formal requirements for minimum hardware to be used in conjunction with the framework. Dr. Varcholik used a small-footprint MicroATX computer for operating his multi-touch surface, TACTUS. Bespoke requires the installation of .NET 2.0 Framework prior to use. Visual Studio 2005 or 2008 is also required to compile the software. Finally, XNA Game Studio 2.0 is required to utilize the XNA presentation layer.

The Bespoke Multi-Touch Framework can be downloaded from Dr. Varcholik's Software Development repository. This is an executable file that will prompt the user to input the location for the framework to be installed. Upon completion of the installation, the user is recommended to run the 4-point calibration. The calibration will allow Bespoke to align the surface, camera, and projector so blobs detected are in correct relation to where they are on screen. After calibration, the multi-touch running Bespoke is ready for use. The user can choose to view the source code or run the applications using Visual Studio. The user can also run the applications mentioned earlier from an executable file that is located in the install folder of the Bespoke framework.

ReactIVision wrote the protocol, TUIO, that sits on-top of Open Sound Control (OSC). Dr. Varcholik wrote his own TUIO-like protocol for transmitting the Touch objects. The multi-touch Network Gateway uses a C# Open Sound Control implementation to efficiently transmit multi-touch points. This is the same method that the multi-touch framework, libdtouch, uses. Dr. Varcholik includes a XML configuration file that allows for customization of several options. These options are the port, IP address, and a choice between unicast or multicast.

The gateway's application-level protocol transmits two types of messages, Point messages and Alive messages. Point messages relay the values associated with a single multi-touch interaction point. Alive messages contain the ID's of the set of active points. 'Point' messages are sent packed in the OSC bundle but 'Alive' messages are sent over a period of time and require no interaction from the surface. The details of the messages are shown in Figure 11 below. In the left table, the size (bytes) of both messages is shown. In the right table, we can see the information that the 'Point' message contains for each touch object.

The underlying protocol that the Network Gateway uses is UDP. Because of this, packets may arrive out of order and others may get lost. To deal with this issue, the protocol checks the time-stamp within the Point message.

Bundle			Point Serialization		
Osc Protocol	Value	Size (bytes)	Field	Data Type	Size (bytes)
Bundle Prefix	#bundle	7	ID	GUID	16
Alive Message		32 per point	Rectangle.Location.X	Int32	4
Point Messages		85 per point	Rectangle.Location.Y	Int32	4
Point Method			Rectangle.Width	Int32	4
Osc Protocol	Value	Size (bytes)	Rectangle.Height	Int32	4
Address Method	/gateway/point	14	TimeStamp	Int64	8
TypeTag	,biiiihfffh	11	Speed	Float	4
Payload	FtirPoint	60	Direction.X	Float	4
		85	Direction.Y	Float	4
			Duration	Int64	8
Alive Method	Size per point				60
Osc Protocol	Value	Size (bytes)			
Address Method	/gateway/alive	14			
TypeTag	,b	2			
Payload	FtirPoint.ID	16			
		32			

Figure 11 - Message Format of Bespoke Communication Layer (Reprint with permission from Paul Varcholik)

3.2.5 Conclusion

All of the libraries discussed prior can be used in Planck. Some of them such as reactIVision and CCV 1.5 come with fiducial support built in. D-touch only serves as a fiducial tracking library so it was not one of the optimal choices considered in this project. The Bespoke Multi-Touch Framework also does not support fiducials. In order to use it, a fiducial framework would have needed to be incorporated into the source code or a vision library that focuses solely on fiducials would need to be run simultaneously. There are software programs that allow for a camera resource to be used by these types of programs simultaneously in windows. This option would have added unnecessary overhead to the system and would have been inefficient given the substantial amount of processing Planck will be doing in order to capture touch events and gestures. The NUI user group stated that the fiducial tracking algorithm of reactIVision had very recently been ported directly into version 1.5 of CCV. This user forum also serves as a place where the authors of the open source CCV can announce their latest releases and news.

CCV contains a myriad of settings and configurations that can be changed. The filters included in the software, and the multitudes of software adjustments that can be made, make it a strong candidate to be the vision library chosen. ReactIVision tracks fiducials very well. In fact, it was built specifically to track fiducials. Finger tracking was implemented in hindsight. Since our system incorporates fiducials and is mostly controlled by fingers touches we need a vision library that can do both of these things.

This research concluded that CCV was the most applicable vision library to use in project Planck. It offered very versatile filters that will work with Diffused Surface Illumination technology. It offered input from any webcam that can be used with windows. It also

offered the ability to track fiducials using reacTIVision's fiducial tracking code and their physical fiducial objects. It offered quick and easy configuration and calibration on the surface of Planck. Finally, it offered a large support community with vast experience making multi-touch surfaces that can be consulted in the event of a major issue.

3.3 Communication

Once the vision library detects touches and fiducials, it must communicate this information to the showcase software. Research had shown that most vision libraries use the TUIO specification to transfer the information detected to external pieces of software. "TUIO is an open framework that defines a common protocol and API for tangible multi-touch surfaces." (Kaltenbrunner) The standard defines how touch events and objects placed on the surface are encoded so they can be sent as control data to the client application. The client application would then be responsible for decoding this information and using it. The TUIO.org website has a number of enabled tracker applications as well as client libraries that can be used in many programming environments. Example client programs can also be found in the Reactivision library documentation website.

TUIO came about as part of the creation of the ReacTable synthesizer, an optical fiducial tracking table that could be used to create music. This table is also responsible for the creation of the Reactivision library which will be discussed more in depth later. TUIO is based on Open Sound Control, a network communications protocol originally designed for communications between computers and midi instruments.

TUIO tracks three types of objects. The first, fiducials, can be uniquely identified and their position and orientation can be determined on the surface. The second type of object is finger touches. These do not get specific orientation data. The third type of object is any untagged object on the surface whose shape is tracked by an ellipse with a bounding box so that orientation and area information may be calculated. As stated earlier, TUIO is encoded using OSC so that the information may be transmitted more efficiently. TUIO can be used with any UDP implementation and sends packets on the default port, 3333. As of TUIO 1.1, TUIO can also be transmitted through TCP and FLC (Flash Local Connection) in order to support communication with Adobe Flash applications. UDP implementation is the default to keep the latency of communications down.

TUIO contains two different types of messages: SET and ALIVE. SET messages transmit information about an object's position and orientation. ALIVE messages state which objects currently reside on the surface. In order to distinguish one segment in time on the surface from another, a time stamp is also delivered with each set of SET and ALIVE messages. This is called the FSEQ message. In the event multiple sources are being used with one client, a SOURCE message can also be sent to identify one TUIO source from another. In order to mitigate packet loss in this communication scheme, redundancy optimizations have been built. This ensures the system is still reliable. Since TUIO doesn't send a message when objects are added or removed, the client side application is responsible for tracking this.

A typical TUIO bundle will consist of the optional source message, an initial ALIVE message, a number of SET messages indicative of all objects on the surface, and lastly an FSEQ message that contains the ID number of the frame we're on. Every object on the surface has a unique identifying number called a Session ID that it keeps until it is removed from the surface. This allows for the ability to track many fiducials with the same symbol ID. Any new blob or cursor that appears on-screen also gets its own unique ID.

Different attributes are tracked for different objects placed on the surface. These include X and Y coordinates as well as vectors for velocity. Fiducial objects contain additional information such as their object id, angle, and rotation values.

The attributes are normalized for each axis and are represented by floating point numbers. The TUIO tracker implementation takes the sensor data and divides by the sensor dimension. Position values are expressed as follows:

$$\begin{array}{l} x = \text{sensor_x} / \text{sensor_width} \\ y = \text{sensor_y} / \text{sensor_height} \end{array} \quad \text{(Equation 1)}$$

Velocity is measured by calculating the movement over a distance in one axis in one second of time. Normalization also occurs in in these values. The box below contains a sample calculation:

$$\begin{array}{l} X = (\text{sensor_dx} / \text{sensor_width}) / dt \\ Y = (\text{sensor_dy} / \text{sensor_height}) / dt \\ m = (\text{speed} - \text{last_speed}) / dt \end{array} \quad \text{(Equation 2)}$$

Rotation velocity is measured by calculating how many rotations have occurred in a time-span of one second. A rotation velocity of (1.0) would mean that one rotation happened in one second. Rotation values are also normalized and the change in angle is calculated by subtracting the current angle of the object from the angle held previously by the object divided by the time difference between the two samples. The rotation acceleration, r, is calculated by taking this change in angle and dividing it by the change in time between the two frames.

3.4 Touch and Fiducial Recognition Software System

The recognition software was responsible for receiving input from the vision library, interpreting the input as one of the specific gestures we're looking for, packaging them into the objects that we need for our showcase program, and then packaging these objects into a shared data structure that the Showcase Application System will peek into to find the current list of touches and fiducials.

There are two ways of implementing the recognition software. The first way involves using a network port as a communications tunnel to feed the information to the showcase software. The other approach was to use a shared data structure and the recognition module embedded inside the showcase software as a thread. It was chosen to use a shared data structure, with the recognition module embedded.

The first way of implementing the recognition software would be to have the recognition software running separately and then sharing its touch and fiducial data through a network socket. Some issues arise from the network socket configuration. A protocol would have needed to be used in order for the showcase application and the recognition software to communicate via a network. In this type of system the recognition software would have a socket listener that would be receiving input information from the vision library via TUIO. Once the touches and fiducials are recognized and packaged, the recognition software would need to use a socket on a port different than the one used to receive the input data to package the touch and fiducial information using a protocol, potentially TUIO as well, and use the new socket as a server to send this information to the showcase application. Our own protocol could also be developed if we found that TUIO didn't work for our intended application. To summarize, the recognition software would need to act as a client to the vision library, run its algorithms to make the gestures with this data, package this data into a protocol built for network communication, and then use a socket server from within to send this data to the showcase application. The showcase software would need to have a socket listener that would receive these communications.

There are some advantages and disadvantages to this approach. The advantages are that the recognition software wouldn't need to be built in the same language as the showcase software. Since the recognition software uses a standardized protocol to send the information, both applications speak the same language. The main disadvantage with this approach is that the recognition software needs to have a server built into it. Not only will it act as a client to the vision library, it will also serve content to the showcase software via a network. This would have generated an unnecessary middle man in our infrastructure that could be avoided by creating the program within the showcase software and simply sharing a data structure. The added complexity of having to write a server service within the gesture library also adds an additional level of complexity that can be mitigated by not creating the recognition software in this way. The recognition software would be in charge of three things: 1-receive TUIO information via a network socket, 2-recognize and package the touch and fiducial information into a protocol ready to send to the showcase software, 3-establish a network connection with the showcase software and send the data over the shared socket. As more touch inputs are received from the input vision library, they are parsed and sent through the network in real time.

The entire process of receiving input from the surface, parsing it into gestures, and then displaying the output to the screen needed to happen within a latency of at most 0.5 seconds. The smaller the latency the more enjoyable the user experience would have been. In this scenario the gesture recognizer would be communicating with the showcase

software via a network. The additional overhead of creating a server and having to transmit this data via a network may induce latency. This should be avoided at all costs.

The alternative approach to creating the software independently and using a network socket to communicate was to create the recognition software within the showcase application as a threaded class. This simplifies the design process since the only networking aspects involved in this approach would be to instantiate a client socket to read the information received via TUIO from the vision library. The process of parsing the input to find and create gestures still needs to happen. Instead of packaging the newly found gestures into a network protocol so that it can be sent through the network socket, the gestures are placed into a shared data structure such as a linked list. The showcase software can then 'peek' into the shared data structure and begin reading the gestures directly. As input information is received from the vision library, the gestures are recognized algorithmically and are added to the shared data structure for reading. Every gesture will have a timestamp associated with it as part of the identifier used by the showcase software. The gesture recognizer needs to be able to receive input from the vision library with very little latency. In fact, this piece of software needs to be able to receive input from the vision library all the time. The showcase software also needs to be able to draw to the screen on the time. The only way to achieve both programs running concurrently, particularly with the gesture recognizer being nested within the showcase software, is to write the gesture recognizer as a threaded class of the showcase software. Such a design ensures concurrency of the two applications. At the application layer, the operating system will handle the multithreaded aspects of both programs and will handle any collisions that may arise. One of the major disadvantages associated with including the gesture recognizer within the showcase program is that it must be written in the same language as the showcase software. This language needs to support the ability to communicate over a network and the ability to use threads. C# is planned to be the language we will be writing the showcase software in. No members of the group have worked specifically with C#, but since it's an object oriented language and all of the team members are well versed in Java it shouldn't be difficult to pick it up quickly. C# also has the ability to run threads and the ability to communicate over networks. Also, there are some example implementations of the TUIO protocol in C# that can be used as a good starting point when writing the gesture recognizer/vision library client.

In summary, there are benefits and drawbacks to designing the gesture recognizer within the showcase application versus designing the gesture recognizer outside of the showcase application. While they are both viable options, designing the gesture recognizer within the showcase application will be the choice of implementation. This is because of the reduced latency on system communication and the simplification of communication between the two pieces of software, as there is no need to write a network protocol to communicate with the showcase application.

3.3.1 Top Level Program Flow

The gesture recognition software received TUIO messages. Each object on the screen was represented with its own unique ID number. Each fiducial also had a unique ID number. This list of on-screen objects was received through a socket and read first by the gesture recognition system. The list of received objects was stored in the program temporarily. When an object first came into the gesture recognition system, it was checked against all the current elements in the shared structure to see if they already exist. If it does exist, the object was updated with whatever the newly received information is. If the object was not in the list, a new container for it was created and the object entered a function that will determine if it is a touch or a fiducial. If the function determines that the object was a fiducial, the fiducial ID's are determined, and the object and its corresponding information were packaged to be placed in the shared data structure. If the object was not a fiducial, then it must be a touch. Once all of the new touches have been saved into the gesture data structure, the socket was checked once more for new data from the vision library and the process begins again. When an object was removed from the surface it was searched for in the list of gestures and was removed.

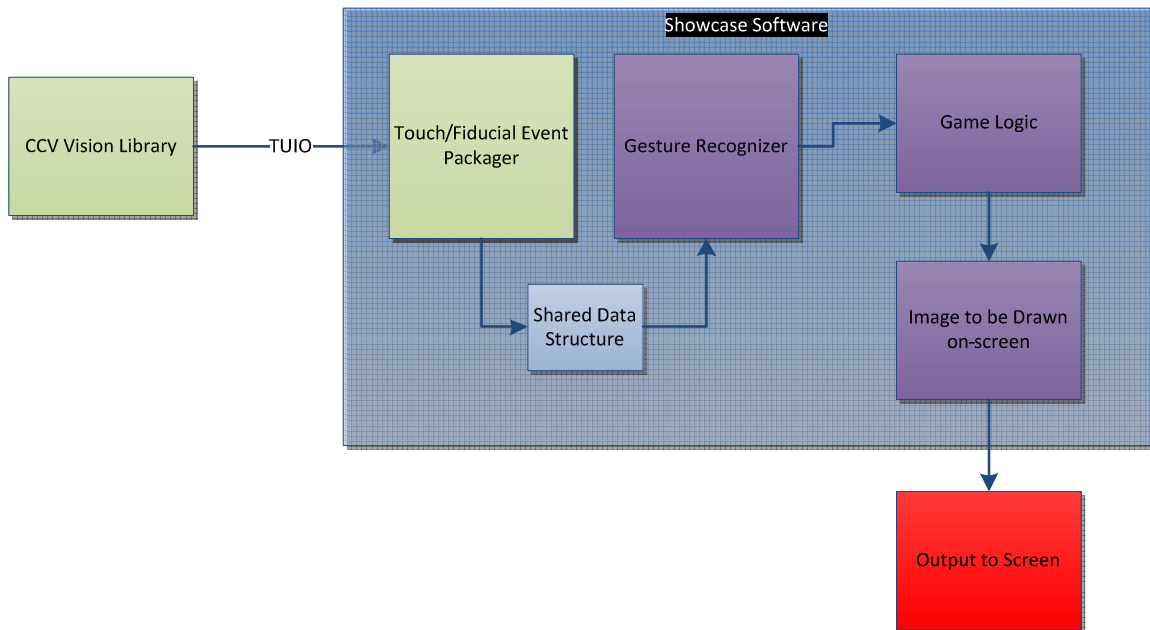


Figure 12 - Top Level Program Flow

3.4.2 Reading from TUIO

Messages were received from CCV 1.5 vision library over TUIO. These were read in by the gesture recognizer program using a TUIOListener class. Messages were received by the gesture recognition program whenever objects on the surface were added, removed, or moved. Messages were sent to any program that is registered with the TuiOListener

interface. This interface was provided as part of the many examples in the TUIO and Reactivision website. TUIO can convey information about fiducials (TUIO refers to these as objects) and touches (or cursors, as the protocol calls them). Each of these types of objects has the events discussed attached. The events are listed in Table 1.

Table 1

Message Name	Purpose
addTuioObject(TuioObject tobj)	this is called when an object becomes visible
removeTuioObject(TuioObject tobj)	an object was removed from the table
updateTuioObject(TuioObject tobj)	an object was moved on the table surface
addTuioCursor(TuioCursor tcur)	this is called when a new cursor is detected
removeTuioCursor(TuioCursor tcur)	a cursor was removed from the table
updateTuioCursor(TuioCursor tcur)	a cursor was moving on the table surface

In addition to these messages TUIO also sent a message entitled “refresh”. The purpose of this message was to indicate the end of a packet of information from TUIO. In our application it served as the end of one frame received from the screen. Once this message is received, the gesture recognizer can surmise that the next input received corresponds to a new session of touches on-screen. Every time an object was seen on the screen, messages were received into the gesture recognition application and these objects could be queried based on their type for the information we need about the gesture. The two types of objects that TUIO sent inside the touch events are objects and cursors. Table 2 shows the fields associated with a fiducial object (Kaltenbrunner).

The TuioPoint class is extended by the TuioCursor and TuioObject classes. Both of these serve as a container for the TUIO positions. TuioPoint is also comprised of a timestamp to indicate the last time this object was updated since the session start, the time this object first appeared on the screen since the session start and its x and y coordinates.

The TuioTime class that is referenced in currentTime and startTime is a structure made to keep track of how much time has elapsed since the beginning of a session. The time is initially set by the instantiation of the TuioClient class and can be retrieved anytime thereafter.

Table 2

Class TuioCursor		
Variable Name	Variable Type	Variable Description
Cursor id	Protected int	The individual cursor ID number that is assigned to each TuioCursor.
Motion accel	Protected float	The motion acceleration value
Motion speed	Protected float	The motion speed value
Path	Protected vector <TuioPoint>	A Vector of TuioPoints containing all the previous positions of the TUIO component.
Session id	Protected long	The unique session ID number that is assigned to each TUIO object or cursor.
State	Protected int	Reflects the current state of the TuioComponent
TUIO ACCELERATING	Static int	Defines the ACCELERATING state
TUIO ADDED	Static int	Defines the ADDED state
TUIO DECELERATING	Static int	Defines the DECELERATING state
TUIO REMOVED	Static int	Defines the REMOVED state
TUIO ROTATING	Static int	Defines the ROTATING state
TUIO STOPPED	Static int	Defines the STOPPED state
X speed	Protected float	The X-axis velocity value
Y speed	Protected float	The Y-axis velocity value

As each touch was received from TUIO it was added to a linked list array that contains all of the touch objects. In the event a fiducial was received on the surface, the object added to the linked list was the object type defined earlier. Since each fiducial object had a unique symbol ID, each fiducial was able to be differentiated from another. Additionally, each cursor and fiducial object had a unique session ID. In the event more than one instance of the same fiducial was found on-screen, the session ID could be used to distinguish the two. The fiducial, once added to the linked list, could be updated as needed based on the information received from the vision library through TUIO.

A few other methods that came with the TUIO standard involve the ability to write the gesture recognizer so that it could poll the vision library for new touches or fiducials. There were four methods provided by the creators of TUIO for these purposes. They were `getTuioObjects()`, `getTuioCursors`, `getTuioObject(long s_id)`, and

getTuioCursor(long s_id). A table summarizing their attributes can be found below in Table 3. The first two methods return a list of all active objects and touches on the screen. The latter two return specific attributes of each object referenced using the session ID (s_id). In this paradigm the program isn't event driven and is instead driven by how fast the software polls the vision library to update the information on-screen.

Table 3

Method Name	Input Type	Return Type	Method Purpose
getTuioObjects()	None	Vector	returns a Vector of all currently present TuioObjects
getTuioCursors()	None	Vector	returns a Vector of all currently present TuioCursors
getTuioObject(long s_id)	Long number specifying object we want info about	TuioObject or NULL	Returns a TuioObject or NULL depending on its presence
getTuioCursor(long s_id)	Long number specifying cursor we want info about	TuioCursor or NULL	Returns a TuioCursor or NULL depending on its presence

3.4.3 Implementation

The TFRSS software was implemented as a thread of the main showcase software. This class was called "GestureTracker". In order for it to listen to the vision library and TUIO, it implemented an instance of TuioClient. It also implemented the TuioListener class. A code snippet found below from the TUIO documentation is included to illustrate how the GestureTracker class instantiates the TuioListener, and how an object of the TuioClient class is handed the GestureTracker class so that it can respond to the events handed to it by TUIO. The code snippet below creates an instance of the GestureTracker called application. A new TuioClient is created and is set to begin listening on the default port (3333). In the third line, a new client, the GestureTracker object "application", is added to the list of listeners associated with the "client". In the final line, the client's connect method is executed which starts the TuioTime and allows the client application to connect and start listening to the vision library.

Plank used two C# 'List' objects that respectively hold cursors and fiducials from CCV. They were cursorList and fiducialList. The lists were private objects that were peeked into by the showcase application. They were accessed by two public getter methods that create duplicate objects of the lists and return those to the calling function. It returned a

duplicate of the list so that atomicity of the shared data structure was not violated. During the creation of the duplicate lists, the original lists were locked for writing, updating, and deletion of a touch or fiducial object by using a lock object. Any of the touch or fiducial update methods could override this lock to perform its desired function. The lock took care of ensuring no data was lost or overwritten so that the showcase software would not crash by implementing multiple read and exclusive writing access. The type of lock implemented in our software was a C# ReaderWriterLockSlim object.

Gestures in the showcase application were handled by linking individual touches to individual graphics objects using the individual ID number of each touch. The field of view (to be more clearly defined later on in this paper) on the soldier could only be altered if the soldier had been selected by a touch previously. This inherently means that two touches were necessary in order to activate changes in the field of view. In order to implement this gesture, one touch was linked to the soldier object and another was linked to the field of view object. The field of view object could only be selected by a touch once another touch had first selected the individual soldier in question. Using this linking paradigm allowed multiple touches to be linked to multiple objects on screen and ownership of objects on the screen based on touches was preserved. This in turn preserved our multi-user directive. An image two soldiers with blue field of views can be seen directly below.

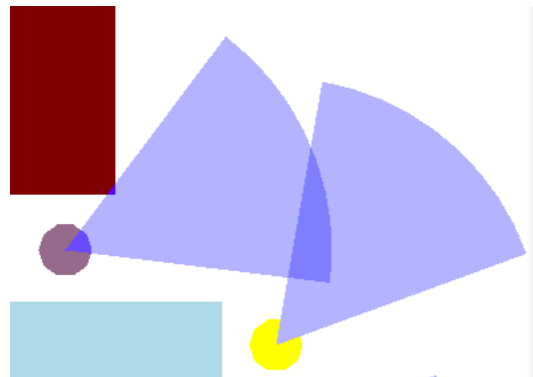
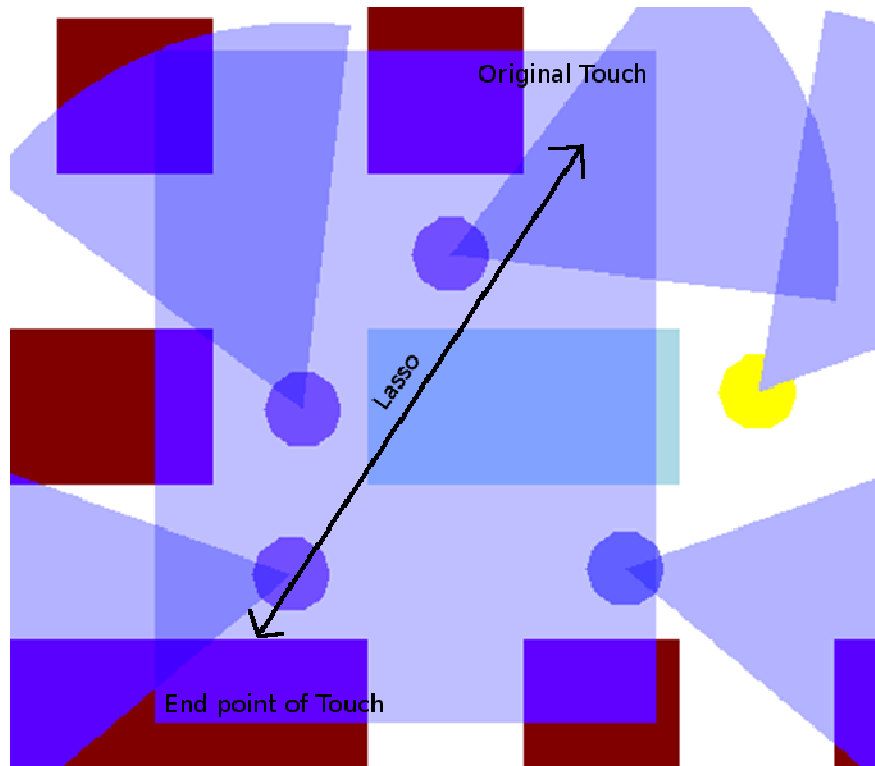


Figure 13 – Field of view example

Lassos were created on-screen whenever a touch is detected on the surface that wasn't located above any graphical object. Lassos were drawn as rectangles that extend from the touch's original position to the current location of the touch. Figure 14 shows a lasso object selecting many four circle soldier objects.



4. Showcase Program

The showcase program was needed to demonstrate the user interface that the multi-touch table was capable of. To show off this innovative interface a human controlled battle simulator could offer an interesting environment to implement multi-touch inputs. This battlefield simulator was similar to many existing pieces of entertainment software which fall under the real time strategy genre. This type of application is highly dependent on user controlled actions using traditional mouse and keyboard inputs which make it a desirable test bed for a multi-touch system. This type of software also requires a high volume of actions to be performed quickly in order for the user to accomplish their objectives; this demonstrated that multi-touch surfaces could be used in time sensitive applications. In addition to simply using touch instead of a traditional mouse input, the keyboard commands and shortcuts were replaced by real life objects with fiducials imprinted on the underside. In this way real life objects could be selected based on their relevance to what was happening in the application, making it easier to memorize what object performs what function rather than memorizing a key whose alpha numeric character loosely defines the function.

4.1 Multi-touch User Interface

The types of applications described above have a plethora of commands needed to successfully accomplish the objectives in the scenario that is presented to the user. A good example would be Starcraft, which, in addition to mouse clicks that select and move units and buildings, incorporates two menus for constructing units and structures. These menus are bound to two different keys that when pressed expand a larger menu. Within that menu six more choices are presented and must be selected using one of the six keys that is bound to that choice. On top of that, selecting different units and buildings may change the function of each key since certain buildings and units have their own abilities which can be activated using the keyboard.

Using multi-touch technology, Planck created a new paradigm for RTS input. The inputs still accomplished the same tasks as they always have in a RTS scenario; however the physical input from the user was dramatically different and arguably easier to learn and master. All traditional inputs were accomplished using two different methods; physical touching of the screen and using fiducial imprinted objects. All the inputs that previously required a mouse were handled by single or multiple finger touches. Any kind of selection of a unit or building as well as giving units commands could be handled through single finger touches. Unit abilities and special functions could be handled by using multiple fingers, double tapping the screen, or creating finger gestures that were linked to the command being given. Having the option of using multiple fingers to select units and give commands facilitates users who need to give commands very quickly, because they are no longer limited by the mouse cursor which can only exist in one window coordinate at a time.

4.2 Graphics API

For the user to be able to take advantage of the multi touch features that Planck will offer, some kind of graphic display was needed. The RTS battle simulator needed to have a graphics application to show the user the results of his or her input. Since very few RTS entries support any kind of touch interface, Planck would require a novel piece of software that exhibits the unique interface. For this to be accomplished, a graphics API was needed to render the elements of the simulator onto the screen. The following section will present some of the more popular APIs while paying close attention to the different ways that they handle input events to the program.

4.2.1 OpenGL

OpenGL is a library of two-hundred and fifty functions and methods as of version 4.2. These methods allow the programmer to manipulate graphics hardware to draw complex two-dimensional and three-dimensional computer graphics. It is platform independent as well as language independent. It is available on UNIX, Mac OS X, Windows, and embedded systems and can be written in C, C++, C#, Delphi, Java, PERL, FORTRAN, and many other languages. Access is provided to a particular system through drivers written by the manufacturers of the graphics processing units.

It takes primitives such as points, lines, and polygons and converts them into pixels by sending them down a pipeline called the OpenGL state machine. Before OpenGL 2.0 this pipeline was fixed. However, the newer versions support a fully programmable pipeline via GLSL. GLSL is the OpenGL shading language, it provides the programmer with the ability to modify the pipeline at the vertex and fragment level. This flexibility allows the programmer to implement physical models that approximate reality such as lighting models, shadow projections, Environment mapping, bump mapping, and per-pixel lighting to name a few.

Since OpenGL is platform independent it does not have any capability of creating a window in which the graphics can be rendered. All graphics applications must have a window in which they run on modern operating systems. Since OpenGL does not support this within its own libraries a separate API is needed to interface OpenGL function calls to the window and event handling systems that exist on modern operating systems. The tool created by Silicon Graphics in order to achieve this on windows is called GLUT or OpenGL Utility Toolkit. GLUT is organized into several sub-APIs which include window system initialization, entering the GLUT event processing loop, window management, overlay management, menu management, and callback registration. Events in a graphics application include things like a mouse moving, a keyboard stroke, or most importantly a call to the rendering function. The main function of a simple OpenGL program consists of a series of GLUT function calls which establishes the event callback loop, initializes the window, and calls the OpenGL drawing function.

On most operating systems the only version of OpenGL that is available is 1.0. In order to access the OpenGL functions the GLUT context initialized extensions are required. OpenGL extensions are useful additions to the API. These have been approved by the Architecture Review Board or ARB (Segal & Akeley, August 8, 2011). These extensions can include anything from the new functions that take advantage of the latest graphics hardware to the OpenGL shading language specification, which is now integral to high level OpenGL applications. In order to take advantage of these extensions the ARB created GLEW or OpenGL Extension Wrangler. A simple GLEW initialization call, as well as having the proper drivers installed, will allow the programmer to access these extra functions and accelerated graphics processors.

Two other libraries allow OpenGL to render models and create textures from images. In order to create textures from images OpenGL is compatible with a library called devIL or developer's image loader. It is a simple open source library with many supported digital picture formats working across all major operating systems and languages that are supported by OpenGL. In order to load large models that have been created by artists using drawing software another library is required due to OpenGL only being a drawing API. ASSIMP or Open Asset Import Library is one such library that OpenGL programmers can use to load large models and assets. Using ASSIMP, OpenGL can quickly and easily load Collada, blender3D, AutoCAD, LightWave, 3dsMax, and many other file types that artists use to create models. This library not only imports the vertices for drawing, it also imports all the normals and material properties of the model. This

allows programmers to create elaborate approximations of real life that include lighting, realistic shadows, texture mapping, bump mapping, and animation to name only few features.

The OpenGL API is a powerful, well documented system for creating high end computer graphics. It is cross platform and cross language and has a lot of support in the open source community. Because of this support many libraries have been and continue to be written to support the OpenGL API. It has been used for well over a decade in many industries. It boasts being the core of hit titles in the gaming industry such as Half-Life and Quake and continues to be the API of choice when it comes to writing professional CAD and simulation software. OpenGL makes available to the programmer a rich set of tools to manipulate powerful graphics processors to create realistic and useful applications that require graphics.

4.2.2 Microsoft XNA Game Studio

Microsoft XNA is a set of tools and libraries that attempts to free game developers from getting bogged down in complicated graphics processing code. It removes the need to code the setting up of pipeline states and complex texture loading code. It was intended to allow students, hobbyists, and independent game developers to make simple games that could be distributed through the Xbox Live network or published as open source game on the internet (Microsoft, 2011). It is based on the .NET framework and while it is a .NET compliant language, it is currently only supported in C# and C++ on Visual Studio 2008 or higher. It has similar capabilities to the DirectX SDK but it abstracts much of the setup code that is needed for creating win32 graphics applications. Using this abstraction, developers are able to focus on game development rather than creating windows handles and canvases to draw on.

XNA supports both two-dimensional as well as three-dimensional graphics rendering. It also comes with tools that allow the developer to port the application to various platforms including the Xbox 360. It has classes built in to aid the user with texture loading, input, 2D and 3D drawing, sound, and even model loading using the XNA Build tool. XNA also provides many starter kits for multiple genres of games. These include real-time strategy, first-person shooter, platform games, and many others. XNA is an easy to use platform with many object-oriented elements which makes it a good candidate for small teams that want to create aesthetically pleasing and interactive applications. However, XNA is not as powerful as dedicated GPU access SDKs which limits to some extent the possibilities graphically. It also has no support for touch surfaces, similar to the other graphics SDKs. XNA will still have to be integrated with the gesture recognition library in order to get input from the user and update the application logic.

4.3 Design

WeDefend would be a showcase application that demonstrates the versatility of touch and fiducial input on a multi-touch surface. Based on the many types of military model

applications that have been made in the past, a real time strategy application was the best choice. However weDefend was a sub-genre of real time strategy which has become popular in the video game industry called tower defense. A traditional tower defense game has the user defend an area or asset located behind the military units. The user must strategically place military units to bar the enemy from either reaching a certain area or destroying the asset being protected. If the user fails to protect an asset or allows too many enemies into the restricted area then the scenario is failed. We defend was not a full-fledged RTS due to the lack of manpower and experienced game producers needed for a RTS application. An RTS application also traditionally requires the selection of units which has many input collision and verification problems for a multi user touch environment. Therefore similar concepts from the research will be extended to work in a tower defense scenario.

Tower defense games have many benefits in both development and training for the user. It has become exceedingly more common for the military to use games as virtual training simulations for officers and enlisted soldiers (Macedonia, 2002). A tower defense scenario is inherently military in its nature and can be used to train military personnel in how to effectively set up blockades, whether they are naval, urban, or rural in nature. This type of model also has the added benefit of flexibility in the types of scenarios personnel can be trained on. Not every scenario must involve setting up a restricted area and keeping the enemy from reaching that area. Tower defense models can also be used to create virtual models of VIP (very important person) defense situations. These scenarios are more dynamic because they involve a moving asset rather than a fixed point that must be defended.

The tower defense model can be expanded to train personnel on many types of defense scenarios relevant to any modern military or police force. This model has added benefits of being more slowly paced than traditional real time strategy models allowing multiple users to work together in real time to both prepare and direct a defense scenario. Traditional real time strategy models rely heavily on what is called actions per minute. This requires the user to perform various tasks very quickly and make split second decisions. This type of model is more suited to a single trainee. In weDefend the trainees are required to work together and come up with a strategy in order to accomplish goals of the scenario presented before them.

This is a perfect showcase. Not only will it show off the intuitive user interface of a multi-touch surface, it also encourages team work and demonstrates that applications that can be used by multiple people are powerful tools that should be considered for training purposes.

4.3.1 Scenarios

WeDefend presents a common defense scenario that arises in both war-time and peace. The situation presented is the most basic defense scenario that usually arises at war time. Friendly military units must defend an important structure or area that is imperative for the enemy not to reach. In weDefend, the application boots up into title screen similar to

vintage arcade games where the user must place down the preparation mode fiducial in order to proceed. Once the fiducial has been put down, the user will be in the preparation phase. In the preparation phase an area or structure will be highlighted in green to indicate that this is the asset that must be defended. The user will then be instructed to use the fiducials at his or her disposal to create soldiers to defend the area. In this phase the user was able to place soldiers in any configuration they see fit to defend the highlighted area. The user could able drag a unit that he or she placed by mistake to a different location by dragging the unit to the new location. Setting up units and creating new units could only be done in the preparation phase.

Once the units have been set up the user can then adjust the field of view of each unit. Each unit's field of view will be a light blue cone that extends outward from the center of each unit. Anything that enters the field of view of a unit was fired at and destroyed in two shots. By using this feature the user sets up zones which his or her units will defend. The insurgents had a certain number of hit points; these points represented the unit's life. Additionally soldier units had an attack power attribute representing the amount of damage they are capable of inflicting on an enemy; this quantity was never shown to the user and no units attack power will be greater than or equal to any other units maximum hit points. In order for a unit to be destroyed, that unit's hit points must be reduced to zero. Once this is accomplished that unit was no longer be visible on the screen and was no longer be able to inflict any damage. In order to ensure that enemies do not reach the area being protected, it was allowed to have two units fields' of view overlap so that they will both fire upon a single enemy, granted that enemy was in the overlapping field of view area. Figure 15 demonstrates what a field of view looks like and what overlapping fields of view look like.

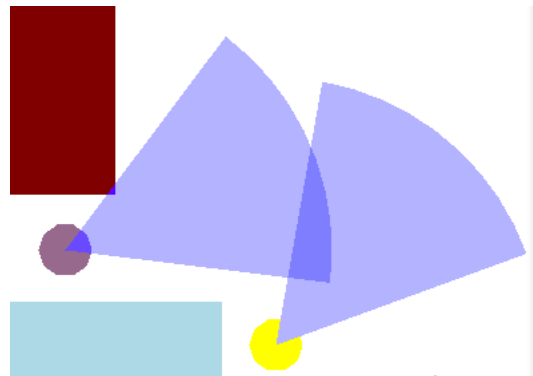


Figure 15 - Overlapping Field of Views

The user was allowed to augment the field of view of any soldier both in the preparation and action phase. The users were able to set-up patrol routes and create squads in the preparation phase only. In order to make the scenario more realistic and versatile the trainee may send a unit on a patrol route. A patrol route is a set path that is outlined by the user using waypoints or delineating a path for the unit to follow with a drag gesture. Patrol routes can be used to make units more effective by being able to cover areas that may contain enemies dynamically. Once a unit is set on a patrol route, his field of view was always along the line they are walking along. The field of view could also be

adjusted as the unit is patrolling in the action mode. Units followed their designated patrol routes until they reach the end of their route. At this time the unit performed an about face and continue patrolling back along the route. Units could have intersecting patrol routes allowing them to cover areas that another unit may not be covering. Should two units run into each other, one will be arbitrarily given a higher depth value and they will pass right over each other. This was a crude way of showing that they avoid collisions in real life.

The other option available was to put units into player groups. Since weDefend was meant to be played by multiple people, player groups allow for a player to colorize his or her soldiers to a random color. Although it did not stop anyone from grabbing another user's soldier it acted as a visual marker to let other players know that you were controlling a set of units. To create a player group the user highlighted the area that contains all the soldiers that were to be put into a particular group. Any units outside the selection area were not put in the group. The color they were assigned was random.

Once the user was finished adjusting his or her units, they will move into the action phase by placing the action fiducial anywhere on the screen. Once a user entered the action phase they could not return to the preparation phase without restarting the entire application. In the action phase enemies began to appear from many places and attempt to get to the highlighted zone. If an enemy reached the zone, an enemy entered counter will be incremented. In the scenario of defending a certain location the objective was to allow less than one-hundred enemies to reach the defense zone and survive until forever or until the users became bored. Should the enemy entered counter reach one-hundred, the user would have failed the scenario and a game over screen was presented, at which time the reset fiducial had to be placed in order to reset the application. In order to not fail the scenario, the user was able to issue only a few commands to his or her defense team. The user was still able to adjust the field of view of all his or her units. The only other thing the user could do in this phase was stop a patrolling unit. If a patrolling unit was stopped in the action phase, they will remain in the location where they were stopped.

All of these actions and automated systems were implemented by using a C# program with supporting classes. The automated logic for action phase state resides in the game1.cs update function, it continually checks for new user input, instantiates soldier objects in the preparation phase, instantiates enemy objects during the action phase, destroys unit objects during the action phase, and updates the fields of all objects. The visual rendering was accomplished by using XNA Game studio 4.0 by: creating a window to draw on, initializing the display data, and finally calling a display function that uses the user input and object states to render what should be on the screen. The game changed to the game over state when 100 insurgents reached the protected zone.

4.4.2 Gestures

In order to carry out the actions described in the above scenarios a set of touch and fiducial gestures was required to provide user input to the application. There are only three types of basic inputs that make up the many gestures that controlled the application. The three inputs are a fiducial being placed on the table, a single finger touch and drag, and a two finger pin and drag. Fiducials are pieces of paper with special patterns printed on them. These patterns can be read by the touch and fiducial recognition system and be given unique identification numbers which can represent different actions in the application. Fiducials can be used in place of menus to create a seamless, immersive, and intuitive feel to an application that would otherwise require menus. In weDefend, fiducials will be used to initiate the action phase, initiate the prep phase, place soldiers, activate debug modes, and reset the application. Fiducials do not need to necessarily exist as a piece of paper. They can be customized to be imprinted on some kind of physical token which can represent whatever action that particular fiducial stands for. By doing this a developer can create an even more immersive and believable model or simulation.

Once all users had finished in the preparation phase it was necessary to activate the action phase to initiate the defense scenario. To do this, the action fiducial needed to be placed on the surface. Once the touch and fiducial recognition system had recognized the action fiducial, it switched the game state to the action state via a finite state machine, which allowed the main action mode game loop to begin running. Once the action phase was in progress, the action fiducial, prep fiducial, and soldier stamp fiducial were totally inert and were ignored by weDefend. The stamp soldier fiducials only worked during the preparation phase and was used to “stamp” units onto the battlefield. When any of the fiducials for unit creation were placed on Plank the touch and fiducial recognition system reported a screen coordinate relating where that fiducial was placed. With that information, weDefend instantiated the object type and fill in the coordinate location field with an X and Y coordinate where the unit should be drawn by the graphics application. In this way, users were able to easily place units onto the battlefield without having to access any kind of menu. Multiple copies of the soldier fiducial existed so that each trainee will be able to place units where they see fit. Having multiple instances of these fiducials being placed at the same time was not be a problem because each fiducial had a unique identification number to differentiate it from the others. This allows weDefend to only be concerned with the data the touch and fiducial recognition system reports to it about a fiducial. These fiducials also became inert once the action phase had begun because placement of units was not allowed in the action phase.

The waypoint puck was used to create a patrol route in the preparation phase for a soldier. Creating waypoints via pucks was one of the only gestures that required both a finger touch and a finger drag. In order to create a patrol using pucks a user must place a finger on the unit that he or she wants to send on a patrol. Then, while still holding one finger on the unit, the user must then use the waypoint puck to drag out waypoints similar to how he or she would drag around units. In order to make clear to the user that waypoints are being created, fuchscia dots were drawn to give a visual cue of the path the soldier

will patrol. Creating patrol routes using pucks was a quick and easy way for a user to create a non-loop patrol. Similar to the other actions discussed thus far, the waypoint puck became useless once the action phase has started.

The remaining gestures used touch events to both setup and command units in both phases. The two gestures were a drag, which will be used for moving units in the preparation phase and creating user group lassos, and a pin-drag used to adjust the field of view and create patrol routes. A drag was defined as putting one finger on Planck and physically gliding the finger over the acrylic surface then finally lifting it off of Planck. The first action a user can accomplish with this type of gesture was moving an existing unit to a new location. In order to do this, the user must place a finger on a unit. The unit was then highlighted in green as a cue that the unit is active. Once a unit is active, the unit could be dragged across the battlefield and then dropped into place by removing the finger. This command was only valid in the preparation phase and was carried out by matching a touch event's screen coordinates to the screen coordinates of a movable object, then carrying out the necessary graphics translations to show the movement. The drag could also be used to group soldiers into groups. To group soldiers into a user group, a finger must be placed outside the coordinates of any movable or selectable object. After the finger is down it should be dragged around the units that are to be grouped creating a box or around them. Once the finger is lifted off the surface, all the units that were within the "lasso" were grouped and they were highlighted with a squad color to show that they were all now part of the same squad. The figure below illustrates how units can be grouped into squads.

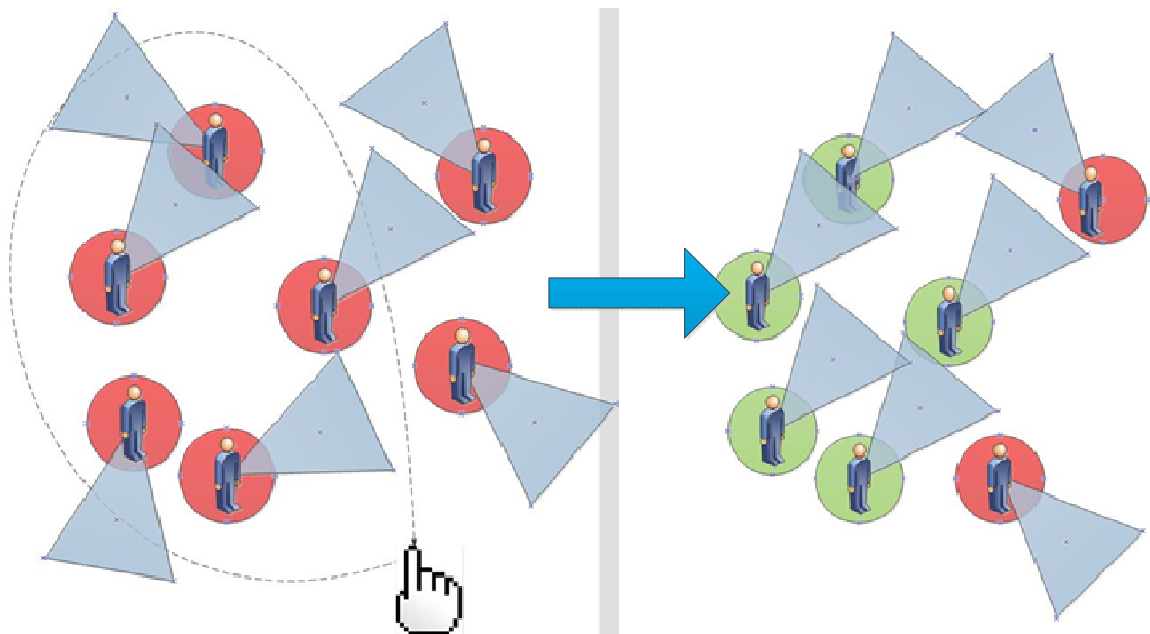


Figure 16 - Example of Lasso Select Gesture

The commands that involved pin-drags are adjusting a unit's field of view and creating a patrol route. A pin-drag requires two separate fingers to execute. It was defined as placing a finger on a selectable object, like a soldier, then placing the second finger down near the vicinity of the first finger and then dragging the second finger in any direction away from the first finger. In the case of adjusting the field of view of a unit this action is quite intuitive. To adjust a field of view the user should place a finger on a unit. Once this is done, the unit's field of view will change to a dark blue color indicating it is ready to be adjusted. The user can then proceed to place his or her finger on the field of view and then drag in any direction to extend and adjust the field of view of that unit. Once the user was finished he or she should lift the fingers off of Planck to confirm the new field of view. To achieve a believable balance, a ratio was established using the distance in pixels from the unit as the denominator of the ratio which will affect how the angle was calculated. The field of view angle was inversely proportional to the field of view distance. If the distance of the field of view was large, then the field of view in degrees will be smaller limiting the probability an enemy will enter the field of view however it increases the range that a soldier could engage an enemy.

The last command, creating a patrol, also used a pin-drag and was similar to adjusting field of view but was slightly different in how the user must execute the gesture. In order for a user to set a soldier's patrol route, the user must first put a finger on a unit. Then place a second finger a smaller circle called a puck. The active radius of a unit was denoted by a green circle that surrounds the unit when a finger has been placed. Once both fingers were placed, the finger on the puck should be dragged away from the unit in order to specify the route which the soldier should patrol along. The route was denoted by fuchsia dots that were drawn as the user drags their finger. Once the fingers were lifted off of Planck, the patrol route was created and the unit began patrolling. The last and final command that a user could give was to stop a unit from patrolling. To do this the user need only tap on a unit that is patrolling and the unit would immediately stop patrolling and hold the position and field of view that they were in when they were tapped. The figures on the following page illustrate how to adjust a field of view as well as create a patrol route.

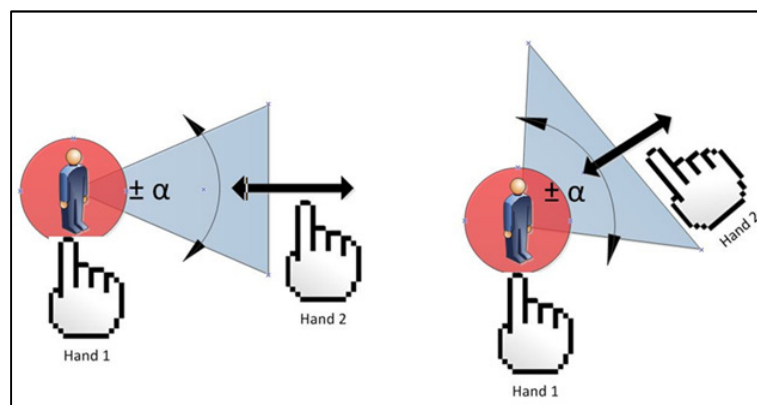


Figure 17 - Example of Pinch Gesture for adjusting Field of View

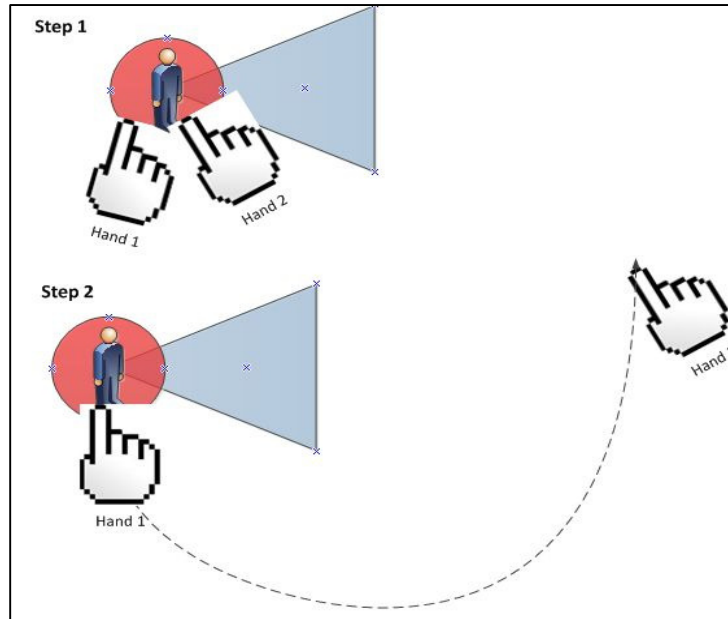


Figure 18 - Example of Patrol Gesture

4.4.3 Graphics Structure

XNA Game Studio was responsible for rendering the required graphics, executing and processing scenario logic, and recognizing user input. XNA is a powerful tool created by Microsoft to aid low budget and hobbyist developers to make games. It abstracts much of the initialization and setup that is required for other graphics APIs like DirectX and OpenGL. One of the most useful things that XNA does automatically is setup a window as soon as the project is created. This window already has a bound win32 handle enabling that window to handle input events as well as being able to receive rendered images. This makes XNA the prime candidate for developing weDefend because of how rapidly small development teams can create publishable software. XNA also makes loading textures, sprites, and models as easy as including a header file in a C++ project. XNA can use a wide variety of the most common image formats and use them as textures; the user can render multiple images onto the screen and have that image be hardware accelerated. XNA also allows the user to extract color data from an image if he or she wants to use the image as a height or bump map, to name a few. All of these texture accesses can be done by including the image file into the project binding it to a C# texture2D variable and then using four to five function calls to initialize and render the image. Models are just as easy to use as textures as long as the three-dimensional model file type is of .x or .fbx format.

Although XNA abstracts away a good deal of graphics setup and data initialization it retains the power of other graphics APIs which provide libraries for matrix and vector manipulations as well as providing functions for rotations, translations, scaling, perspective, and camera operations. Because of this the user must still have some expertise in three-dimensional graphics programming in order to create aesthetically pleasing and logical three-dimensional environments. A basic XNA application starts with two files program.cs and game1.cs. The program.cs handles creating the entry point

or window that the application will use. The game1.cs file will be the main class from which the application will run. All other classes in the application will act as supporting classes that contain all the data and methods needed to create the different elements of weDefend. The game1.cs contains the bulk of the logic and all of the graphics rendering functions. This file contains seven different parts: use statements, class and instance variable declaration, the constructor, an initialize method, a load and unload content method, an update method, and finally a draw method. These parts make up an XNA game and certain parts will be modified heavily in order to achieve user input from the Touch and Fiducial Recognition System.

Use statements are similar to includes in C programming in that they tell the compiler where to find all the pre-existing resources that XNA and weDefend require. The class and instance variable declaration section is where the main weDefend class is instantiated and all the variables necessary are defined. The constructor is where the graphics device and content pipeline are initialized. The graphics device set-up is very simple and only requires that the user create a graphicdevicemanager object and then initializes that object in the constructor. The content pipeline is how XNA is able to easily use textures, images, sprites, sound effects, and models by simply including the resources in the project. The initialize method is where all non-graphics related components such as soldier and insurgent classes are setup. The initialize method is traditionally reserved for settings like screen height and width, full screen mode, and vertex and index buffer setup for primitive drawing to name only a few. The load and unload content methods are used to create or destroy sprite batches which are in charge of holding all the images that will be used as sprites as well as loading all other content such as models, sound files, and graphics effects that contribute to weDefend.

The update method is reserved to run the scenario logic. This method is integral to weDefend because it is where supporting class objects will be updated, created, or destroyed. It will also be in charge of interfacing with the gesture tracker class. The update method should first access the touch and fiducial lists and associate gestures in the list with the proper soldier, field of view, and puck objects. A separate algorithm inside interactive objects will read through the list of active gestures and use their coordinate data to link gesture identification numbers with each object's gesture link ID. Then the update unit state method will update the fields in each object. Only units whose gesture existence Boolean is true will be affected by the update unit state method; this allows the update unit state method to loop through the gestures that are active for that object. This is also where new objects will be created or destroyed depending on what is happening in the scenario. After these methods are done the update function will proceed to update the game state and execute the logic within the active state of the finite state machine. The lose condition is continually checked to see if one-hundred or more enemies have slipped through the defensive line in the action state. Update will also be in charge of switching between the preparation and action phases. This means that all the logic described above will be repeated in a two argument switch case. Update also provides a system timer called game timer so that the developer does not have to deal with that minor aspect of the scenario.

Finally the Draw method is where all the rendering takes place. No updating or manipulation of data other than graphical rotation, scaling, translation, perspective, and camera manipulation should take place in the draw method. This method begins and ends when sprite batches are rendered allowing the user to change textures on the fly or animate sprites by quickly rendering a series of images in a sprite batch. The render function is also in charge of rendering geometric primitives such as terrains and other non-model non-sprite assets. This function will also be in charge of rendering the soldier, wall, field of view, puck, debug, and insurgent objects. In order to do this each object type will have a render method built into the class which prepares all model assets, textures, and sprites as well as initializes all the data necessary to render all the objects. In this way one render method can be called for each object that exists and will be updated continuously since the method will exist inside each instance of soldier or insurgent. This does break the convention of doing all rendering inside the draw method however; since the method is still only called inside of draw it does not totally disregard tradition. This methodology allows the programmer to simply loop through the soldier array and for each soldier that exists call that particular render function and all the objects should be rendered based on the data that is contained in each instance of that particular object.

4.4.4 Code Structure

WeDefend was made up of seventeen classes. Four of these classes are objects which have a role in the scenario, while the others are support classes of the main four classes. The main game1.cs has all the instantiation of our four main classes as well as the finite state machine and the instance of gesture tracker which updates the shared input lists. This class was responsible for holding the majority of the logic and running the scenario in weDefend. The five protected methods are standard methods in any XNA Game Studio application. These five methods are in charge of running the application and are explained in greater detail in the section preceding this one. Initialize is the first of the protected XNA methods. It is in charge of setting up all non-graphic related components like vertex and index buffers, which contain data for drawing primitives, and other parameters that affect the screen. The load and unload content methods are also protected by default and are in charge of binding and sending all textures, images, sprites, models, sound clips, and other assets to XNA's content pipeline. The update function traditionally changes things like position and game data based on mouse and keyboard input. This method was severely overhauled to incorporate Plank's novel input. The update method ran a finite state machine which uses bit flags to check for the state this FSM decides which state weDefend is in. It then updated all the interactive objects if weDefend was in the preparation or action state, by calling the interactive objects respective update methods to link gestures to soldiers. Then the unit specific update methods adjusted all the fields of the existing objects given they have an input gesture linked to them. Once these two methods are done isOld method runs to make sure that touches that are linked to objects can not affect other interactive objects as well as check whether the stamp fiducial has been placed to create a new soldier. The draw method will take care of all rendering and is explained in greater detail in the section above.

The other five methods are used to assist the update method in interfacing with, and assuring the state machine is reset when the reset fiducial is placed. The update methods of each interactive object were passed the two shared input lists that gesture tracker adds touches and fiducials to. With these lists they were able to run the linking algorithm as well as update the fields that are being affected by input that is linked. In order to link an input to an interactive object, first it is checked whether or not the linked Boolean of that object is set. If it is, then it was proceed to search for the touch that is linked and update the object based on the updated information of the touch position being a simple example of such. If the linked Boolean is not set then we proceed to search through the list and see if it is inside the bounding radius of the object, if it is we make the link ID of the object equal to the touches ID and set the linked Boolean. When the method is finished each instance of a soldier will have corresponding gestures that are related to them if any input occurred near the object and would have been updated appropriately. The update method would also update the debug status of all debug objects of the debug state is toggled. The debug state is necessary when testing the software on a prototype that does not have a rear projection acrylic so that touches can be seen. It is also a useful tool for assuring that units are behaving as they should.

The two types of fields in the weDefend class are graphics related fields and scenario logic related fields. The graphics related fields define things like the screen width and height as well as scaling factors for sprites, textures, and models. The scenario logic related fields are the bit flags which run the FSM. The other two fields are linked lists called soldiers and insurgents. These two linked lists contain each instance of soldiers and insurgents. It was used to loop through all the instances of soldiers and insurgents so that they can be updated every time update unit state is called and every time they need to be rendered.

The other classes in weDefend are a plethora of support classes necessary to execute all manner of things from debugging to fields of view. Some of the more important classes are the debug Item, FOV, patrol route, polygon drawer, lasso, gesture tracker, and particle. The debug item class is responsible for creating a series of rectangles which contain text that gives information about each interactive object such as position, angle of the field of view, and whether or not the linked bool set. It also is integral in testing weDefend on a prototype with no rear projection acrylic since it places small black diamonds that denote touches. FOV is a class which contains many of the same functionality as an interactive soldier, however it differs in that it draws a field of view and detects whether or not a touch is inside it with a more complicated geometrical intersection algorithm. Every soldier owns an instance of FOV. Patrol route is similar to FOV because it is an interactive object however its algorithms are almost identical to soldier except for the algorithms that drop waypoints as it is being dragged and then makes the soldier follow those waypoints. Again each soldier owns an instance of patrol route.

The polygon drawer class is a helper class which uses primitive batch to draw things like circles, diamonds, cones, boxes, and other manner of geometrical objects that make up the units, walls and interactive objects in weDefend. Lasso makes use of polygon drawer

in order to draw selection squares on the surface of weDefend. Lasso is used to create player groups. Any soldiers that are inside a lasso when a user releases a lasso receive a random color to denote that those soldiers belong to that user. The particle class is in charge of making soldiers shoot bullets at insurgents. It uses simple physics to calculate where an insurgent is going to be in order to collide with him after a set amount of time. Some variability is also thrown in so that soldiers sometimes miss. Finally the gesture tracker class is a thread that is instantiated in weDefend so that it can write to the shared data structures, more information on this class can be found in part three of this paper.

The soldier class contained many important methods and fields. The soldier class has fields which allow all game logic and graphics to be updated. An integer will represent the attack power constant that never changes for any soldier. The field of view distance defines how far from the unit the field of view must be drawn. The field of view angles theta and phi are used to orient the field of view. Theta is the angle relative to the center of the soldier while phi denotes the angle of the actual field of view which changes based on distance. The screen coordinate point object is two integers x and y which defines where the soldier should be drawn on the screen and where he exists in screen coordinates. The unit ID is an integer which counts up from one and is assigned to every instance of soldier that is created. The patrol route object as described above was owned by the soldier to give the soldier class the capability of being put on patrol routes. This data structure had a start point and endpoint to denote the beginning and end of a patrol route. The gesture link ID is an integer which denotes which touch the soldier is currently linked to. A soldier would fire at the first enemy that enters his field of view and would not stop firing at that enemy until the enemy is either eliminated or exits the field of view at which time the soldier will then choose the next enemy in his field of view to attack. The linked Boolean will be used by the update unit state function to tell if any input needs to be resolved for a particular soldier instance.

The methods for the soldier class are a series of overloaded update and draw functions that update or draw based on what state the game is in and what parameters are needed in each state. The soldier also has a follow patrol route method which it uses to move along patrol routes autonomously based on the list that is returned by the patrol route puck. The update debug method continually sends the debug information from fields of the soldier to the debug item that has been assigned to that soldier. The update cursor method is used to assist in updating the soldier's position fluidly instead of snapping the soldier's position directly to the location of the linked touch.

Insurgents are created by using two classes. The insurgent generator class uses a random number generator to spawn insurgents at a set number of hard coded spawn points all around the map. The percent per update chance of spawning an insurgent is .003 if the random number generator creates a number greater than this value then it will create an insurgent at one of the hard coded routes.

The insurgent class is responsible for handling the logic once an insurgent has been spawned. Insurgents update methods are continually moving them along their patrol route every tick of the clock towards the protected zone. Once they reach the zone they

disappear. This movement as well as whether they have been shot, explained below, is calculated in the insurgent update method. The draw method displays the changes on the screen. The test hit and collision method use simple object collision in order to calculate whether or not an insurgent has been hit by a bullet. If they have then a hit point is subtracted from the hit points field and if the insurgent has run out of hit points the insurgent is killed and removed from the insurgent list.

5. Computational Container System

The Enclosure/Computer System is the brains of project Planck. The computer powers the entire system. The Enclosure provides protection, support, and cooling to all of the components of the multi-touch system. The Computational Container System outputs power to the Control System. It also outputs power to both the IR camera and the IR LED's in the Image Recognition System. The Computational Container System inputs the raw IR image from the camera in the Image Recognition System and delivers these multiple images, frames, to the Vision Detection application that is running on the Event Detection System. Finally the Showcase Application will deliver the final graphical output to the video card in the Computer System to output to the screen for the user to view.

The multiple components that make up the Computational Container System are the enclosure, computer mount, projector mount, acrylic frame, cooling, computer, image display, and a physical fiducial object container. The components can be viewed in Figure 19. The Enclosure provides support and a location for all the components of project Planck. The computer mount provides a mounting location and hardware for the computer to rest on in the enclosure. The projector mount provides a mounting location and hardware for the projector to rest on in the enclosure. The acrylic frame provides a secure mounting position for the LED's to rest in around the acrylic. It also provides a secure location for the acrylic to rest on within the enclosure. The cooling of the enclosure provides stable temperatures for all components mounted inside the enclosure. It is important that no hardware component goes above safe operating temperatures. The computer shall have the latest desktop computer hardware available at the time. The computer needs to be fast enough to run the vision framework and Touch and Fiducial Software Recognition System as well as the showcase software without ever bottlenecking the entire system. The image display provides the output image to the user. The image shall be large enough for multiple users to view and interact with objects on the screen. Finally, the fiducial objects and object container combine the fiducial tag with a real-life object. This not only adds aesthetic appeal to project Planck, but is also easier for the user to grab hold of the fiducials.

During the research and design of the Computer system, it's important to remember the goals, objectives, and specifications and requirements at hand. The goals and objectives for the computer system are: To facilitate collaborative work from simultaneous users; and to remove the need for traditional input in favor of multi-touch gestures that real live objects that are in some way relevant to the application. The specifications/requirements

for the Computer System are: A usable system screen size of 40 diagonal inches; and the enclosure shall be tall enough for an average sized user to user standing up (minimum height 30”, maximum height 38”). What will follow are the research and then the design of each component of the Computational Container System.

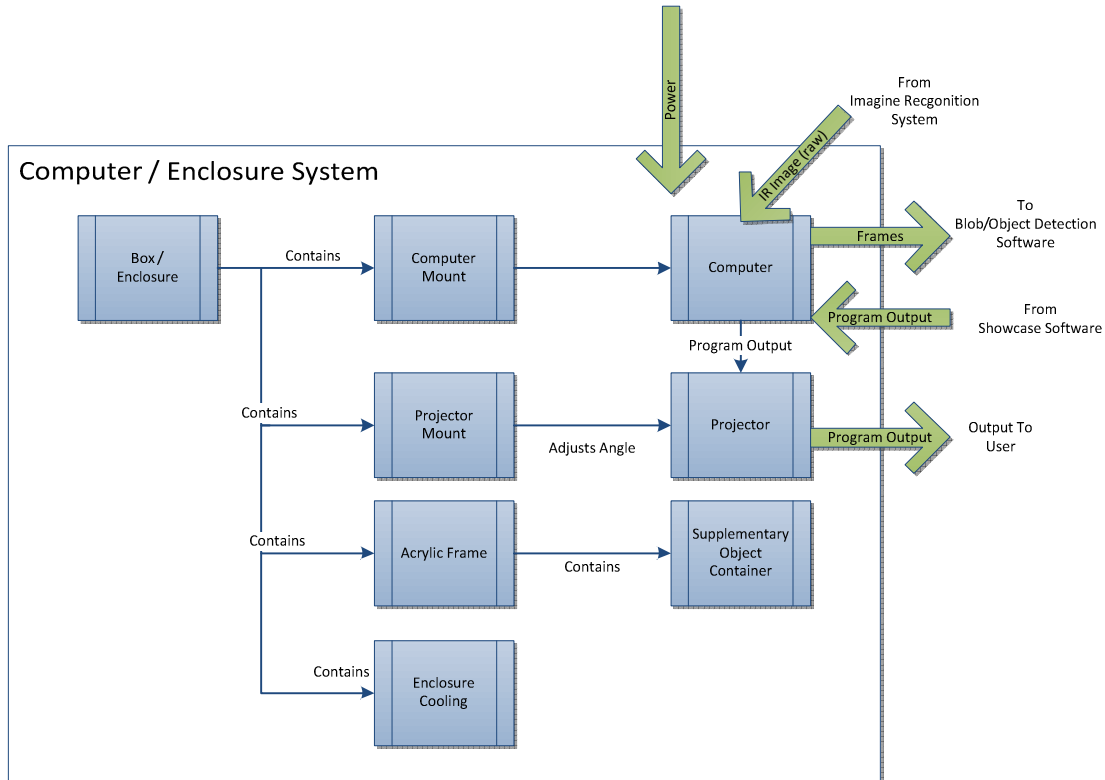


Figure 19 – Top Level Diagram of Computational Container System

5.1 Enclosure

The enclosure is what gives shape to project Planck. The enclosure provides support around all edges of the acrylic and limits the acrylic to an exact location so the user can interact with it. The enclosure houses the image device, computer, and all electronics in our system. The enclosure should be strong enough to support the acrylic and the bodyweight of 4 grown adults leaning over and resting their arms on it. Most wood can achieve this goal, as long as the enclosure is designed properly. The basic design of Planck will be repeated from the lessons learned in the design of the prototype. Other features will be added, but the main concept will stay the same.

The use of a dado blade allows for the wood to interlock with every other piece creating a very strong joint. This takes the guess work out of trying to glue a perfect 90 degree joint. It also prevents the need to reinforce the joint with anything else such as a wooden block or angle iron. This in turn maximizes the space available inside the enclosure. This was beneficial when mounting the projector and other computer components.

Because Planck needs to present a first-rate appearance, the use of a visually appealing wood was implemented into the design of Planck. A high grade veneer hardwood was chosen. It also has the added benefit of being very strong. A high grade veneer hardwood is important so the dado blade does not create divots and imperfections in the edge when being cut.

5.2 Image Display

The purpose of the projection device is to display the images created by the showcase application. This image should be layered underneath the mirror particle acrylic so the user can comprehend the exact spot that he is pressing on the main application. There are two main types of devices available on the market for our application, LCD televisions and Projectors. The benefits and drawbacks of both will be discussed further.

5.2.1 Short Throw Project

A projector projects an image signal provided by the computer onto a screen for viewing. A projector's throw distance refers to the length between the projector's lens and the projection screen. There are two categories of projectors in regards to throw distance, short throw and long throw. Short throw projectors can display the same size image that a long throw projector can, in a shorter distance. This works very well in rooms with limited space. This is great for a multi-touch application due to the limited distance between the screen and the projector inside the enclosure. If the distance is too great, then a user will not be able to use the device comfortably from a standing position.

There are limiting factors associated with short throw projectors however. First, true HD short-throw projectors with resolutions of 1950x1080 do not exist to the mainstream consumer. The two native resolutions within the budget of this project are 1280x800(16:10) and 1024x768(4:3). This means that a very large screen with a resolution of 1280x800 could potentially start to look pixilated and blurry. 1280x800 can achieve the HD resolution of 1280x720, and considering how 1080p was originally created as a cinema format, 1080p may simply be overkill for Planck and its simulation application. Second, the vertical offset of the projector is on average greater than the vertical offset of long-throw projectors. The vertical offset is the vertical distance between the lens and where the projected image starts to be displayed. Short-throw projectors can have a vertical offset of greater than 100%, meaning the projected image does not even begin to be displayed until above the lens. This is because the projector is so close to the screen that it needs to be able to project an image that is out of the viewing angle of the projector itself. This creates complications as the projector would have to be mounted offset of the screen, inside the enclosure. This would increase the overall footprint of the enclosure. An example is shown in Figure 20. Corrections can also be made by tilting the projector or through the use of mirrors. Mirrors will be discussed in more detail later.



Figure 20 - Mounting option of Short Throw Projector with Offset greater than 100%

These two alternatives, however, can produce Keystone Effect. The Keystone Effect is a product of projecting an image onto a surface at an angle causing distortion, as seen in Figure 21. This distortion is best approximated by the function:

$$\frac{\cos\left(\varepsilon - \left(\frac{\alpha}{2}\right)\right)}{\cos\left(\varepsilon + \left(\frac{\alpha}{2}\right)\right)} \quad \text{(Equation 1)}$$

ε is the angle between the screen axis and the central ray from the projector, and α is the width of the focus (Martin). There is software to correct for the Keystone effect but this is only achievable to a certain extent. Plus or minus 30% is a common figure. This is completely dependent on each manufacturer's model. Lastly, short-throw projectors are comparably more expensive. This could be a deciding factor if the budget is smaller than expected.

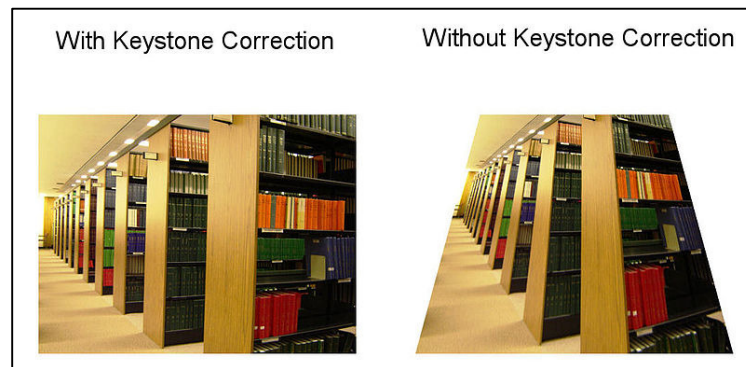


Figure 21 - Keystoning effect (Reprinted under Creative Commons Attribution-ShareAlike 3.0 License)

There are important factors to consider when searching for a short-throw projector that meets the criteria of project Planck best. Dimensions are important because this is the physical footprint in the enclosure. The bigger the footprint, the shorter the distance is between the lens and the screen. The vertical offset is important to note when designing the enclosure. The light output is important so the brightest screen possible for viewing

in all environments can be achieved. Most importantly is the throw distance. Generally speaking, the projector with the shortest throw distance for achieving the desired screen size should be chosen. Lastly, if the projector is to be mounted vertically, it's important to purchase a projector that is designed to do so. Not all projectors have this capability. Disobeying a manufacturer's warning could result in decrease in lamp life of up to 50%. For example, the cost of a replacement lamp for a Hitachi CP-AW250N is \$175. The Hitachi lamp is rated for 5000 hours in economic mode. Mounted vertically, the lamp could be reduced to 2500 hours. There are many variables that lead to reduced lamp life; one is an issue of heat. The projectors were not designed to dissipate the heat mounted in a vertical position and this can cause the projector to get too hot and burn up the lamp.

5.2.2 Long Throw Projectors

Hot mirrors are implemented into the design of multi-touch displays to eliminate the IR light emitted from the projector. A hot mirror is a mirror that reflects IR light while allowing visible light to pass. The hot mirror will be placed directly in front of the projector's lens to filter out any IR light that is emitted and would cause interference with the object/blob detection from the camera. Thankfully, most projectors on the market today don't emit IR light.

The second option is to find a projector that provides the desired 1080p resolution with the shortest throw possible. At the minimal required display of 46", the projector that required the least throw distance was the Vivitek H1082 at 64". With 5'4" inches of required distance between the lens and the projection surface, a mirror would be required to achieve this distance within the enclosure. Also, the brightness of many long-throw 1080p projectors with the throw-distance requirements tends to be on the lower end of the scale. It's rare to see a long-throw projector with greater than 1800 lumens in the budget. Visibility of the screen will be a concern when the multi-touch display is in well-lit areas.

Mirrors can be implemented into the design to fold light from the projector so a long-throw projector can still be possible in a small enclosure. The concept is to bounce the video light output from the projector off one mirror, or several, to ultimately reach the display at the top of the enclosure. Bouncing the projected image around the box creates the distance needed for a longer-throw projector to project the screen size needed to meet the requirements of Planck. This can create the keystone effect as was mentioned earlier. Exact mirror size, placement, and angling needs to be taken into account in order to achieve the desired result. One typical setup, found on Fig. 6, is to lay the projector flat on the bottom of the enclosure and shine the image into a mirror. The light will bounce off the mirror and, if angled properly, will project onto the display acrylic in the desired location. In order to prevent the Keystone effect, the ratio of a given length of a beam of light to all other beams of light should stay equal, with or without mirrors. This can be shown in Figure 22. If $A/B = (A1+A2)/ (B1+B2)$, then the keystone effect should be minimized.

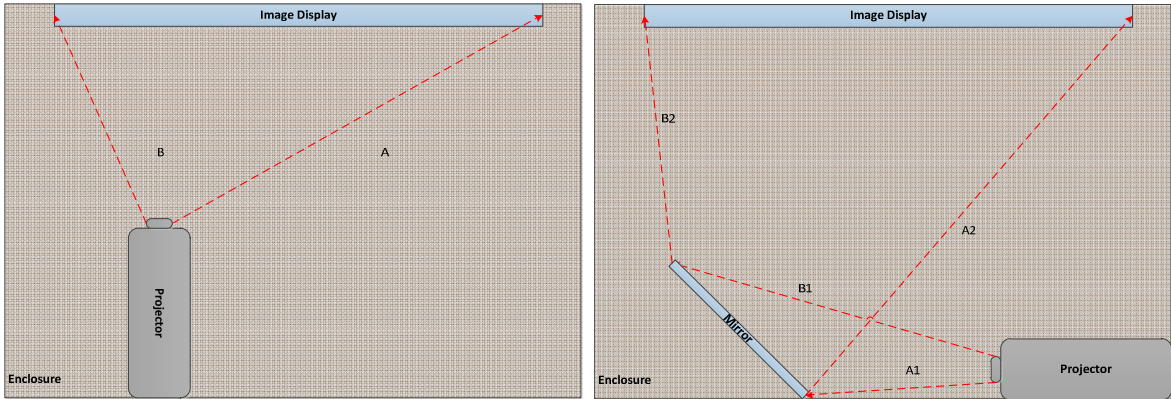


Figure 22 - Minimalizing Single Mirror Keystoning Effect

First surface mirrors should be used to prevent ghosting. A First surface mirror is a mirror that does not contain any transparent layers above the reflective mirror, such as acrylic or glass. These upper layers found on Second Surface mirrors are what promote ghosting of the display image. Most mirrors found in the typical shopping center for cheap prices are all of second surface mirrors. Unfortunately, first surface mirrors are quite expensive. A 20"x30" piece of first surface mirror found online can be as expensive as \$190. Due to the high cost, many experiments have been done to use a form of stripper to remove the backside of the cheaper second surface mirrors. The back-side, usually coated in a paper backing, can be removed with the right stripper. The key ingredient is Methylene Chloride. Once the backing is removed, a First Surface mirror is exposed. This is because, typically, the manufacturer does not coat the backside of the mirror with the protective upper transparent layer. This procedure has been accomplished with mixed results, however. The results are dependent upon the mirror bought, the stripper used, and the quality of the job conducted. Once the backing exposes the First Surface mirror, extra care needs to be given as there is no protective layer and the mirror can be scratched easily. Another mirror that might have to be implemented into Planck is a hot mirror.

5.2.3 LCD Display

The third, and last, option is to use a LED-LCD television. The benefits of using an LCD are that the height of the multi-touch table is completely dependent on the camera's needed viewing distance. The camera has a throw requirement to view a certain size image just as the projector does. Minimizing the throw distance on the projector is only one of two parts in regards to minimizing the enclosure height. Wide angle lenses can be used to shorten the camera's distance requirement. No longer is the projector the deciding factor in the enclosure size. This will be discussed more in the Image Recognition System. The image also has a much higher contrast ratio to that of projectors. Comparatively, 3,000,000:1 in an LCD vs. 3000:1 found in projectors. This results in a much sharper image for LCD's. This is a nice benefit to project Planck as a sharper image can mean the ability for Planck to be capable of being used in well-lit areas. A high resolution of 1080p is easily achieved in today's market. This is a higher

resolution than short-throw projectors can achieve. This would allow a larger viewing screen for the showcase application, weDefend. Lastly, LCD's provide a very bright picture and this would be the best approach in achieving a multi-touch display that is effective during all light operations.

The internals of the LCD would have to be removed from the case and broken down for proper layering. This would involve prying off the outer plastic case. Sometimes hidden screws must be found as well. This is so the camera can see the touch and fiducial detection through the LCD layers. This requires the removal of the case, and complete separation of each component inside the LCD television. Each television is different and can be individually challenging. Tutorials for the separation process are not readily available online either. There are two approaches to effectively layering the LCD television for multi-touch use, mounting the backlight behind the camera, or above the camera. The two layering approaches can be found in Figure 21. The benefits and drawbacks to both approaches are discussed further.

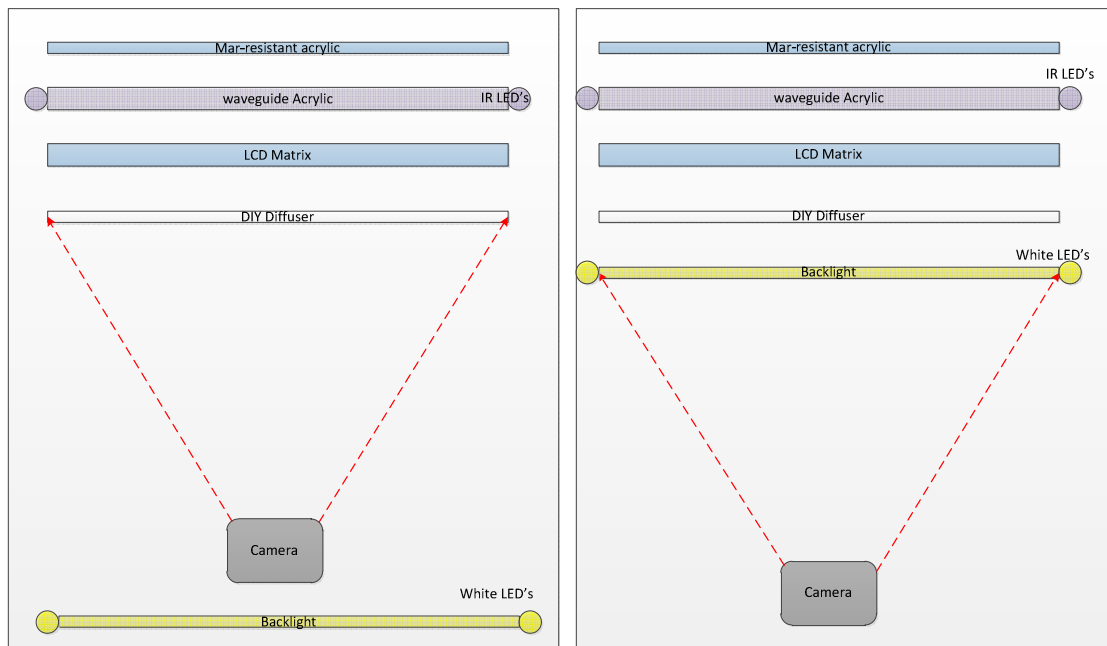


Figure 23 - LCD Layering

The first approach is to mount the backlight behind the camera. This would eliminate any trouble the camera might have when trying to view the touch and fiducials through the backlight. There would be no issue with the opaqueness of the backlight using this method. A cheaper LCD television could also be purchased as LED-LCD would not be of an importance. One drawback is that the backlight is further away from the LCD matrix and may not brighten the LCD matrix correctly, due to loss of light. It would be important to have a completely sealed box and implement a design to minimize the loss of light before reaching the LCD screen. Adding more light sources to effectively brighten the LCD matrix might also be necessary. A second drawback is that the light

source might emit light in the IR wavelength frequency and may distort the image seen by the camera. In this case, the lights would have to be swapped out for non IR emitting lights.

The second approach mounts the backlight above the camera. The benefit to this approach is that backlight rests directly behind the LCD matrix and no issues of illumination should arise. One drawback is that the backlight could provide interference between the camera and the objects/blobs. This could be because of the backlight being too opaque, or the light source transmitting an IR wavelength. A second drawback in using an LCD that allows mounting the backlight in front of the camera is finding an LCD television uses LED edge-lit technology. Both RGB dynamic and full-array LED technologies block the camera's view of the object/blob detection, as it is mounted directly below the backlight. Edge-lit technology staggers white LED's around a special acrylic backlight to evenly illuminate it. This is much like mounting the IR LED's around the EndLighten acrylic. This method would not block the camera's view. IR radiation can emit from the stock white LED lighting but a simple solution would be to swap out the LED's for a new non-emitting IR strip. The backlight acrylic has also been known to have a coating on the back-side of the acrylic that promotes reflection and further illuminates the matrix screen. This however blocks the camera's view. Sometimes this screen can be removed, but not all of the time. If the backlight cannot be re-used, EndLighten acrylic can be substituted.

A drawback for both approaches is finding an LCD with long enough ribbon cables to extend the PCB boards out of the camera's view. The Flat Flexible Cable (FFC) has been implemented into some the LCD's and is notorious for breaking with even the slightest touch. These cables should be avoided when shopping for a LCD suitable for the project. If at any point a part is damaged on the PCB board, the expense will be high to fix, if possible. And that is a risk that would need to be taken if this method was used.

5.2.4 Comparison

A table combining all three categories was created to aid in finding the best choice for project Planck. What's important to note from the table is that a comparable size LCD is about the same cost as a short-throw or long-throw projector. The cost ranges from approximately \$1,000 to \$1,400 for either solution; short throw, long throw, or LED-LCD. The benefits of a short-throw projector can easily be seen by noting the throw distance ranges from virtually 0" to 20" for a screen size of 46". This is a very short distance when compared with a 62" throw from the long-throw projectors, over 3 times as much. Two issues not to forget about when choosing a projector is the vertical offset of the projected image and if the projector can be mounted vertically or not. Surprisingly, there was no direct correlation found with a short-throw projector and having a longer vertical offset. It seems to be dependent upon the design of the individual projector by the company. This is also true regarding the projector being mounted vertically. No one recommends this mounting position, as it shortens the life of the lamp. Some manufacturers give the warning and allow it; other manufacturers prevent the projector from turning on if mounted in this position. One promising candidate is the Hitachi CP-

AW250N. This short-throw projector has gotten great reviews from users over on NUIgroup forums. It offers native resolution of 1280x800, an excellent throw ratio, and is fairly bright. It is, however, a little on the pricier side and has a large vertical offset. Luckily, it's still within the budget and the enclosure can be designed to accommodate the large offset.

5.3 Computer

The computer is the brains of the operation. The computer runs the object detection software, the main application, and the camera that sees the blobs/objects. The computer should never be the source of the bottleneck in Planck. Not a single piece of hardware in the computer should ever reach full utilization when running the showcase program. Because of this goal, the recommended requirements of the most intense software/hardware piece of the system should be well exceeded. The items that could potentially use the most resources are the object detection software, the main application, and the camera. The recommended system requirements of each software component will be checked to find the greatest minimum requirements.

There are no limits to the operating system of the Vision framework chosen. Linux, Windows, and Mac OS X are all supported. The design team of project Planck is most experienced with Windows so it's important to pick out hardware that will run at least on Windows with no driver issues. Since Windows 7 is the newest version out, it's important to meet the minimum requirements of this. This can be found in Table 4.

The XNA API library has a minimum requirement of a video card that's capable of DirectX 10.0 or later. XNA has the same minimum requirements for CPU, memory, and storage space as Windows. The 2x camera has a recommend CPU clockrate of 2Ghz, CPU core count of 2, 2GB of main memory, and no storage space or graphics API requirements. Windows 7 32bit has a recommended CPU clockrate of 1Ghz, 1 CPU core, 1GB of main memory, 16GB of storage space, and DirectX 9 graphics API. Windows 7 64bit has a CPU clockrate of 1Ghz, 1 CPU core count, 2GB of main memory, 20GB of storage space, and DirectX 9 graphics API. CCV 1.4 has a recommended CPU clockrate of 2Ghz, 2 CPU core, 2GB of main memory, no requirement on storage space, and a minimum of DirectX 9 graphics API. ReactIVision has a recommended CPU clockrate of 1.4Ghz, 1 CPU core, 1GB of main memory, 1GB of storage space, and a minimum of DirectX 9 graphics API, and XNA library only has a requirement of DirectX 10 for the graphics API. This is can be viewed in Table 4 below. Another factor to be considered is the resolution that the showcase application will run in. 1920x1080 is extremely graphically intense on the GPU. If the application is to run at this resolution, a high-end video card needs to be purchased. Comparatively, if the application is to run in 1280x720, than a cheaper video card can be purchased

The total minimum requirements can be found at the end of the chart in Table 4. This is the bare minimum system that should be purchased for Planck. It's important to calculate in a big buffer to ensure that the computer will never be the bottleneck in the system. An approximation of at least a 25% performance increase across the board is recommended

for Planck. A quad-core is recommended due to ReactIVision’s use of multi-threads. A gaming video card capable of at least DirectX 10 is also recommended to handle the main application designed in XNA. DirectX 11 video cards have been on the market so long, however, the cost of upgrading to DirectX 11 is marginal and well worth the price increase. A minimum of 4 gigabytes of memory is recommended for a quad-core CPU due to their demanding prefetching scheme.

Table 4

Minimum System Requirements

	CPU clockrate	CPU core count	Memory	Storage Space	Graphics API
2x cameras	2 Ghz	2	2 Gb	N/A	N/A
Windows 7 32bit	1 Ghz	1	1 gb	16 gb	DirectX 9
Windows 7 64bit	1 Ghz	1	2 gb	20 Gb	DirectX 9
CCV	2Ghz	2	2Gb	N/A	DirectX9
ReactIVision	1.4 Ghz	1	1gb	1 Gb	DirectX 9
XNA Library	N/A	N/A	N/A	N/A	DirectX 10
Total requirements	2 Ghz	2	2 Gb	20 Gb	DirectX 10

5.4 Enclosure

There are many components that will need to be mounted inside the enclosure. Details that need to be taken into consideration are as follows: The method that the components will be mounted; how the components will be cooled; and the location in the enclosure the components will be mounted. Hardware will have to be installed for methods of installing the components. Fans will have to be installed to provide adequate cooling to the components. Finally, diagrams will have to be made to map out the location of all components in the system.

5.4.1 Component Mounting

The computer will be mounted inside the enclosure of the system. The computer hardware will be screwed down directly to the inside floor or the wall of the enclosure. Since it is already protected from outside elements, there is no benefit to mounting the computer hardware inside a typical computer case. To reduce the contact between the PCB board and the wood, small wooden or acrylic pegs will be mounted to the inside floor of the enclosure. The motherboard will be directly screwed down on-top of these pegs. There are no detrimental reasons to mount the computer hardware in a certain spot of the enclosure, as there are no requirements associated with this. It’s important to give other devices priority on location. The computer could be mounted on the floor of a

corner of the enclosure, but it could not interfere with the location of the camera or projector. The hardware could also be mounted directly on an inside wall, but a potential issue arises when inserting dedicated cards into the motherboard, such as the video card. Cards that insert into the PCI bus are designed to be supported at multiple points. This would not be a possibility if mounted on the inside wall. The PCI port would support the entire weight of the card and could be a risk concern.

The projector will need to be mounted on a platform that swivels on the vertical axis. This allows the projector to tilt for precise calibration of the image being projected. Figure 24 shows how the platform will be designed. The projector could be mounted on a slate of wood which has hinges attached. The hinges attach to the floor of the enclosure and allow the platform to swivel 180 degrees on the vertical axis.

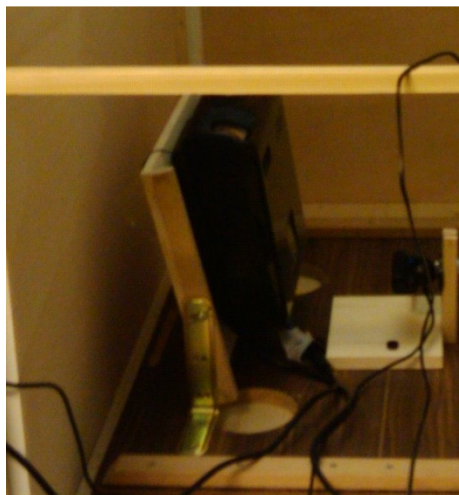


Figure 24 - Projector Swivel Platform (Reprint pending approval)

5.4.2 LED Frame

The enclosure also needs to house the IR LED's that will illuminate the acrylic. In the prototype Phloe, a frame was fabricated with thin pieces of wood encompassing the cut-out opening for the acrylic display. Evenly spaced holes on the frame would house LED's. The end result is a rectangular wooden frame, resembling something much like a picture frame, with LED's mounted inside evenly spaced holes along the entire perimeter of the wooden frame. This finished frame would fit snugly between the acrylic and the enclosure. An example of the wooden frame used in the prototype can be viewed in Figure below.



Figure 25 - LED Frames

Another method is to mount all LED's to a PCB board that is mounted inside a channel around the acrylic. The LED's would be soldered to even intervals on the PCB board. The PCB board would be the exact dimensions as the wooden frame, just large enough to fit inside the wooden channel of the enclosure. The PCB would add needed structure to the LED's so the position and the angle, relative to the acrylic's side, is held. Another benefit is the vastly reduced exposed wiring. In the creation of the prototype, there were many exposed wiring. Exposed wiring promotes accidental breakage, makes it harder to conduct maintenance inside the enclosure, and adds unneeded complexity when conducting maintenance on the LED's.

5.4.3 Component Cooling

The hardware components inside the enclosure will be cooled by fans. There are many different fans of sizes and type out on the market today. Computer fans that run on 12v are the most reasonable choice as a computer power supply will be our main source of power. The fan chosen must have the fourth wire, Power Width Modulation (PWM). There needs to be at least 2 fans in the system, an intake and exhaust. The fans will be mounted on the side of the enclosure where an opening will be drilled. Since heat rises, it makes the most sense to put the exhaust fan higher than the intake. 120mm fans will be chosen because of the lower frequency of sound they produce as well as the higher amount of air they can push. There is a direct correlation between bearing choice and noise output. There are four main types of bearings here; Sleeve, Rifle, ball, fluid, and magnetic. These are compared in Table 5 below.

Table 5

Fan Comparisons

Bearings	Cost	Lifetime	Noise	Total
Sleeve	5	2	4	11
Rifle	4	6	4	14
Ball	3	8	3	14
Fluid	2	10	5	17
Magnetic	1	8	4	13

Cost and noise are given a score 1-5, 1 being the lowest. Lifetime is given a score out of 10, 1 being the lowest, based on its high priority. The goal is to prevent frequent maintenance of the system. Noise is second most important as, ideally, the system should be as quiet as possible to prevent distractions. In reference to Table 5, it is shown that sleeve and rifle bearing fans both have the least life. These are eliminated for that reason alone. Ball, fluid, and magnetic bearing fans would all probably be acceptable in meeting the requirements of the system. Fluid bearings are the highest scoring fan however. This will be of the highest demand when searching for a fan for the enclosure of the system.

6. Control System

The purpose of the control system will be to regulate the power to other devices. The benefits of this are twofold. First, this allows for greater control of the sensitivity of the device through the hardware, so the software calibration will not be completely necessary when the device changes lighting conditions. Second, this allows for control over the fan speeds, which can reduce the ambient noise and power consumption when the device is not being fully taxed.

The control system as a whole, is not a required system in the project, but supplementary. The touch screen could function with this system entirely removed. Because of this fact, most of the components within this system were chosen to be able to be designed and implemented without exhausting exorbitant amounts of resources.

6.1 Power

Power for the device can come from multiple sources. The computer and projector within the device will have their power provided by their own power sources. Without modifying the underlying hardware within those devices, the power sources cannot be changed, and that is not within the scope of the project to do. It is possible that a system can be built that combines all of these internal power sources into a single external power source. This can be as simple as adding a surge protector within the enclosure. A surge protector would not only combine all of the separate cables into a single cable leaving the

system, but it would also add a layer of protection to the system. But, if the end user connects the final project to a surge protector, creating a daisy chain, this could potentially create a fire hazard.

The LEDs and fans on the other hand can be powered from multiple sources. It is possible that they can be powered using their own power source. The simplest way to do this would be to use a linear power supply. This would require minimal additional hardware and would give great control over the voltage that was being fed into the system. Alternatively, a power regulator could be partially or entirely built to the specifications we desire. Both of these options do require additional hardware to be purchased, as well as having the disadvantage of being a 3rd power cord going through, and possibly out of the device.

The final option, which is what is used in Plank, is to use what is already available. The computer will have a power supply and additional rails that can be used to power additional components within the device. These rails function on 12 volts. Depending on the wattage of the power supply selected, the amperage that each rail can output will vary, but multiple rails could be used to supplement. Many of the power supplies considered have the capability of outputting their entire amperage over a single 12 volt rail.

6.2 Pulse Width Modulation

Control over the speed of the LEDs and fans can be accomplished through analog means or digital means. Using analog means has several disadvantages. Because the LEDs function as diodes, they have a minimum, non-zero voltage which is required to turn them on. This would have to be adjusted for within the circuit. Also, the voltage through the LEDs is depending on the current. This means that adjustment of the LEDs would be non-linear when using a potentiometer alone to control the voltage. Analog signals are more susceptible to noise than equivalent digital / pulse width modulated signals. Last, this has been known to cause excessive wear on the LEDs due to creation of heat.

The speed of the fans should also be dynamic. Under varying loads, or when the device is in an idle state, it will produce noticeably less heat than in a state of heavy usage. By varying the speed of the fans, the resulting noise output can be significantly decreased. Fans come in varieties with 2 pins, 3 pins, and 4 pins. The optional 3rd pin adds feedback based on the speed the fans are currently running. This feedback can be interpreted by a computer or another device to vary the speed by varying the voltage over the first two pins. The optional 4th pin allows for low current pulse width modulation to act as a control signal for the fan (Burke, 2004).

A simple way to achieve pulse width modulation is through a 555 timer. The circuit to use a 555 timer as pulse width modulator is a well-known design, which would not require any additional design for the project. It could just be implemented according to the specifications required. This is the route that other projects, such as the Poker Table, went with. There are disadvantages to the use of the 555 timer though. Most notably is the fact that it is manually controlled. The device would be controlled by the difference

in resistance over a potentiometer, but that potentiometer would have to be controlled by the user of the device. In the case of the fans, the speed of the fans may need to be controlled automatically based on the temperature within the device. Even if this temperature is displayed, controlling the speed of the fans may be cumbersome and inefficient for the user of the device. There would also need to be a separate 555 timer circuit controlling each component that needs to be regulated. This can have a significant increase on the number of components within the device.

In order to get automation of the control, a processor is going to be required. In this case, a microcontroller would be sufficient. But, the specifications of the project do not require very high requirements of the microcontroller, so there are several available options. There are several different brands and types of microcontrollers out on the market that would be more than sufficient for these relatively low specifications. For the purposes of this project, familiarity and simplicity are two major factors in the design of the control system. Spending time learning a new language or programming with an unfamiliar device for purposes of supplementary systems would have inhibited work on other systems. Because of these factors, an MSP430 will be used for the project.

In order to control the LEDs, a potentiometer would need to be hooked up to an analog to digital controller within the microcontroller. For the fans, a temperature sensor may be added to automate the speed. If the temperature sensor is not used, then a potentiometer would also need to be used to control the fans. In both cases, another analog to digital controller would be required. To add a degree of expandability within the project, it would be beneficial for there to be at least 4 channels on the analog to digital converter. There are also several types of analog to digital converters that come on the MSP430 chips.

The cheapest method is to use onboard comparators. These comparators can tell the difference between the input voltage and a set voltage. This can then be used to infer the voltage. But they can only tell if the voltage has increased or decreased. SAR analog to digital conversion functions in a similar fashion. Every time a sample is taken, the current voltage is held. The new voltage is then compared against the previous voltage. The final method is Sigma-Delta analog to digital conversion. These use additional digital components to reduce the effects of noise.

The pulse width modulation would be handled through the general purpose output pins. There would need to be one of these one of these for the LEDs and one for the fans. As with the analog to digital converters, it would be beneficial to leave some room for expandability within the project, so at least 4 general purpose output pins would be required.

An MSP430G2231 fit our specifications. It comes with a 10-bit SAR analog to digital converter, comes in an easy to use, DIP packaging, and has 10 pins which can be used as either analog to digital inputs or general purpose outputs (Texas Instruments, 2011).

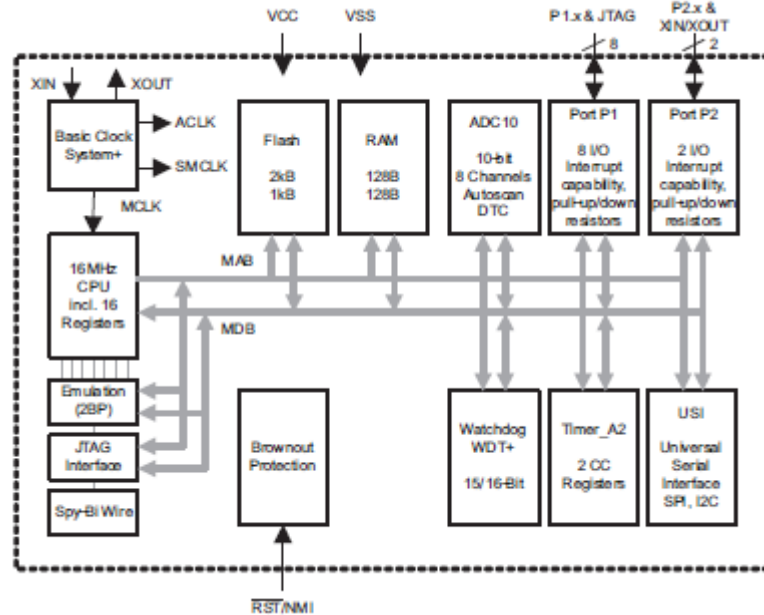


Figure 26 - Functional Block Diagram of MSP430G2231 (Reprint pending approval)

6.3 Temperature Sensing

The amount of heat that must be dissipated from the device varies based on how much the current workload of the device is, as well as conditions of the room in which the device is operating in. All of the internal components may have their lifetimes noticeably decreased by the temperature within the device. Because of this, there needs to be a significant amount of heat dissipation.

Increasing the amount of heat dissipation also increase the ambient noise that the device is producing, so it is important that the speed of the fans be adjusted based on the temperature within the device at a given time. In order to do this, we will need at least one temperature sensor.

There are two major producers of heat within the device. They are the projector and the computer. If a single temperature sensor is used, it would need to be positioned between the two devices, so that it could accurately measure the temperature between the two. A more ideal setup would be to have two temperature sensors, one on or near each device. If either of the devices increases passed a specified safe temperature, then heat dissipation could be increased.

Temperature sensors also come in several types. The primary variants are local temperature sensors and passive infrared temperature sensors. Local temperature sensors measure the temperature locally. Whatever the sensor is placed on is the temperature that they measure. Passive infrared temperature sensors measure the temperature of a surface remotely, using infrared. Since the temperature being measured is local to the two main heat sources, a local temperature sensor would give the most accurate reading.

The MSP430 microcontrollers also optionally come with built in temperature sensors. But the locations that are being measured are not going to be the location that the chip is actually going to be in. This means that the built in temperature sensors would not be ideal for our implementation. There are also digital temperature sensors with built in analog to digital converters or other components which output the temperature in a digital fashion. Since the plan is to utilize the built in analog to digital conversion on a microcontroller, an analog temperature sensor was sufficient.

The maximum temperature that the temperature sensor should be able to read should be at least 20% higher than the maximum operating temperature of the components within the device. The maximum operating temperature is expected to be lower than 70°C, so the maximum temperature must be at least 85°C. The device is not expected to be operating in conditions below freezing, so the minimum temperature can be at or above 0°C.

It is also important that the output voltage scaling is high. With a low output scaling voltage, there is an additional source of error caused by the accuracy of the analog to digital converter. Because of this, an LM37 type temperature sensor was more ideal than a LM35 or LM36. The LM37 has a scaling of 20 mV/°C, compared to 10mV/°C of the LM35 and LM36. The LM37 also has an offset of 0 mV at 0°C (Analog Devices, 2010).

Table 6

Comparison of Temperature Sensors

	Scale (mV/°C)	Output at 25°C (mV)	Output at 50°C (mV)	Output at 75°C (mV)
LM35	10	250	500	750
LM36	10	750	1000	1250
LM37	20	500	1000	1500

7. Image Recognition System

The purpose of the Image Recognition System is to collect the image data from the user and output it in a format that is efficient for the computer to use. The data that is collected needs to be filtered of any erroneous information so that the relevant information can be obtained accurately and efficiently by the Object Detection System.

A common method for filtering out erroneous information is to work within the infrared domain. This is done by removing visible light, and increasing the contrast of the objects you wish to view. There are 3 primary components to achieve this. The first is infrared LEDs. These act as the source of the infrared light. The second is an acrylic surface which refracts, reflects, or diffuses the infrared. The final piece is an image sensor, which actually receives the image and sends it to be analyzed by the Object Detection Software.

7.1 Types of Image Recognition

Multi-touch systems have been around for decades. The earliest ones were being made in 1980s. The currently exist on many laptops, tablets, and phones. But not all of the technologies are scalable to a large table. Also, not all of the technologies would allow for the feature set required for this device. So, the type of image recognition that is used will have a significant impact on the capabilities of the device.

Capacitive touch, Resistive touch, and Electromagnetic Resonance (EMR, Wacom) are technologies that are used in smaller devices like laptop computers and cell phones, but require very specific technologies which are expensive to scale up to large tables.

7.1.1 Frustrated Total Internal Refraction (FTIR)

The type of image recognition that originally started the research and development of these technologies for large screens was called Frustrated Total Internal Refraction (FTIR). The theory behind this is that infrared entirely refract within the acrylic layer. When an object applies pressure to the acrylic surface, the infrared can no longer refract completely and instead exits the surface out the bottom. These disturbances are then caught by a camera below.

The setup of this type of system usually involves an active layer of acrylic. This acrylic is not any special type of acrylic and most acrylic will work. The infrared LEDs are housed in or around the acrylic. A compliant surface is made above the acrylic. This consists of a material used as a diffuser being attached to top surface of the acrylic by a very think rubbery layer. A projector or LCD is then placed below the acrylic to project an image, along with a camera that reads the touch events (NUIGroup).

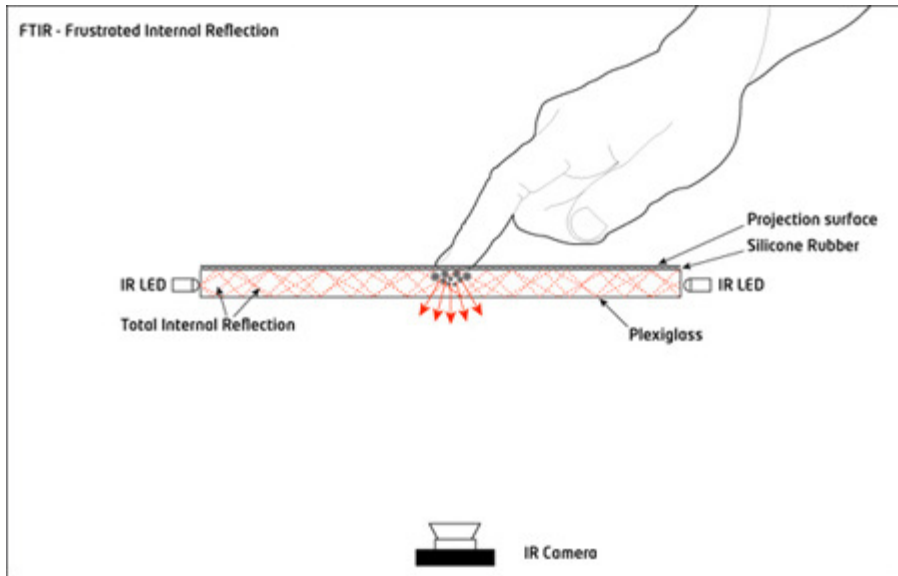


Figure 27 - Illustration of Frustrated Total Internal Reflection (Reprint pending approval)

The advantage of this method is that it delivers a very high contrast ratio for touches on the surface. It also is relatively unaffected by outside disturbances. This makes it a very reliable system for interpreting touch events. But it is not without its disadvantages. Because FTIR relies on pressure being applied to the surface to cause a disturbance, fiducials and other objects cannot be recognized. Also, the surface does not properly pick up dragging. This requires that there is another layer on top. This layer must be free of air bubbles, thin, and made of a malleable material. This layer can be difficult to properly place down and an unsatisfactory layer can have significant effects on the output image.

7.1.2 Front/Rear Diffused Illumination (DI)

A second method was developed that could see objects more distinctly on the surface. This was called DI (Diffused Illumination). The theory behind DI is rather simple. Infrared light would be shined directly at the surface. The light would travel through the acrylic. Any object above the surface would then reflect the light back down at the camera.

There are actual several different setups that work for DI and the design for DI tables is not concrete. Depending on the size and shape of the surface you are using, different numbers and types of infrared lights can be used. The lights also do not need to illuminate the surface from below. They can illuminate the surface from above and the resulting image would be inverted. Acrylic is also not a requirement. Because the surface only needs to have infrared light travel through it, any surface that is clear will work. A normal DI setup utilizes a layer of clear acrylic as the active layer that reflects the infrared light. A diffusion layer is then placed on top of this surface. Multiple high powered infrared light sources illuminate the bottom of the surface attempting to create

an even covering of light. It's critical that the diffusion layer still lets a significant amount of the infrared light through the surface. Light from fingers and objects are reflected more when they are close to the surface than they are when they are far away (NUIGroup).

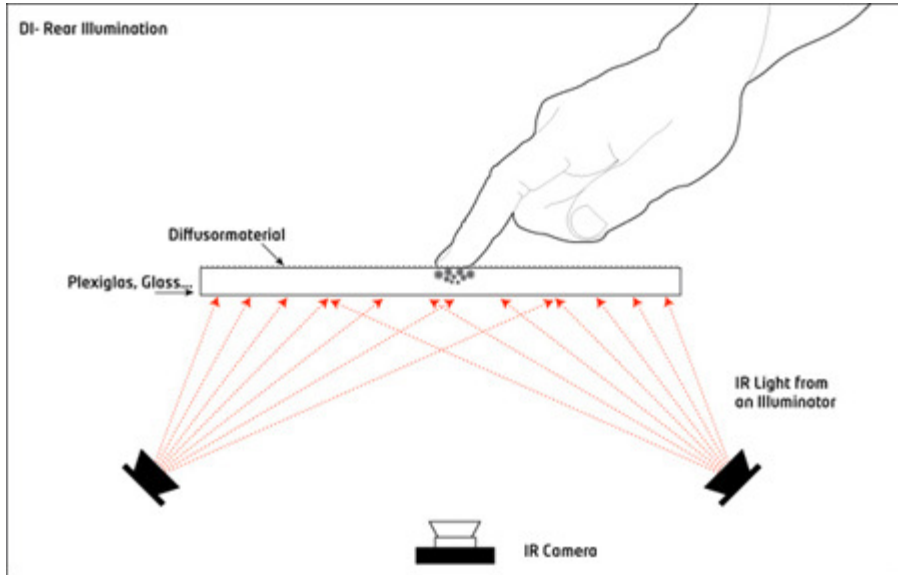


Figure 28 - Illustration of Diffused Surface Illumination (Reprint pending approval)

The primary advantage over FTIR is that objects can be recognized on the surface. The device is not limited to only touch. The lenient materials in the design also mean that the device can be designed to be more functional. Materials that may be easier to drag your finger across, or may be more resilient can be used. It does have several notable disadvantages. Because it relies on simple reflection, anything near the surface could potentially give off infrared and cause a disturbance. The Object Detection Software also needs to be calibrated to detect objects at the correct distance and there is not a high contrast between objects on the surface and objects near the surface. Because of the setup of infrared lights, these devices are also known to have an uneven distribution of infrared light which must be calibrated for as well.

7.1.3 Diffused Surface Illumination (DSI)

Rather than directing the infrared light directly at the surface, it would be more effective if you could evenly disperse the light over the entire surface. Using a special type of acrylic called edge diffused acrylic, DSI (Diffused Surface Illumination) creates an evenly dispersed layer of infrared, which can then be read back by a camera.

A typical DSI setup would include an active layer of edge diffusing acrylic such as EndLigthen or ELiT. Infrared LEDs are housed around the acrylic. Its important that no extra infrared light escapes from the LEDs without passing through the acrylic first, or it can create hot spots around the edges of the device. A diffusion layer is placed over the

top of the surface. A projector and camera are placed below the surface and the camera reads the touch events. If the acrylic of an FTIR table is replaced with the edge diffused acrylic necessary for DSI, then a FTIR table can be converted to DSI. Similarly, a DI table is easily converted into a DSI due to the edge illuminating acrylic not interacting severely with the method DI uses to view touch events (NUIGroup).

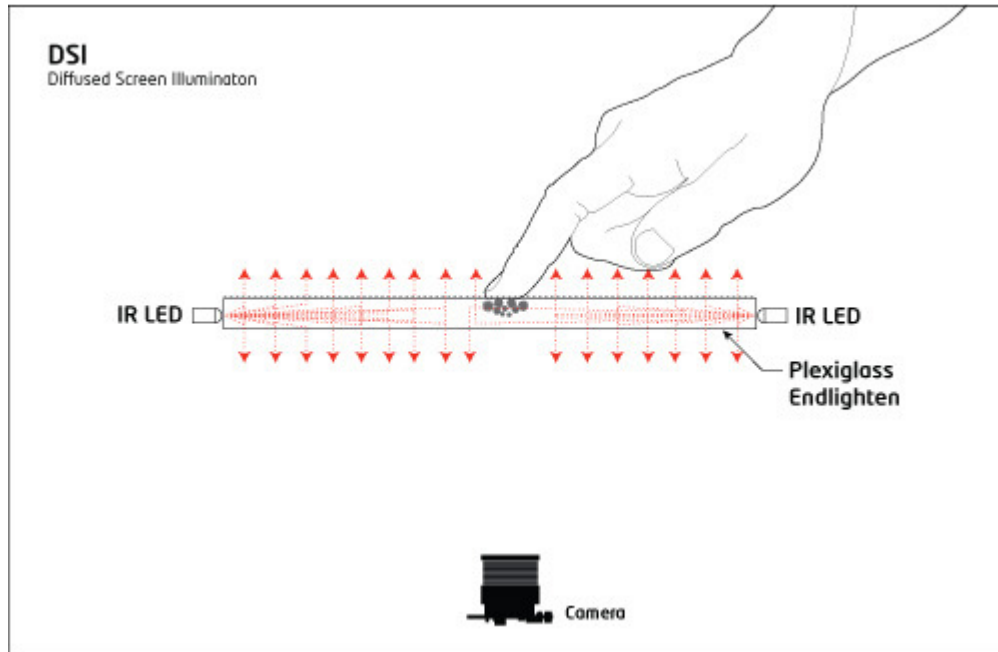


Figure 29 - Illustration of Diffused Surface Illumination (Reprint pending approval)

Edge diffused acrylic has metallic dust mixed into it which reflects the infrared light coming in through the edges. This light then creates an even layer illuminating the bottom and top of the acrylic. Like DI devices, DSI devices allow for object detection. But, because the source of the light is centered directly around the acrylic surface, the contrast of objects near the surface is increased. And because the light is evenly distributed, you do not need to worry about the uneven distribution seen in DI devices. Like DI, the surface is still susceptible to outside interference. It also requires a specific type of material to function which is more expensive than the alternatives offered by FTIR and DI.

7.1.4 Interpolated Force Sensitive Resistance

Some methods do not utilize infrared light at all. A newer technology that has been introduced uses glass or acrylic that has been coated in a material that has force sensitive resistance. The two layers of materials are also configured with a set of parallel electrodes, with each layer's electrodes being perpendicular to each other. The higher the pressure on the surface, the more current can travel through the electrodes near the source of the pressure. This information can then be gathered to pinpoint the location of the touch as well as the amount of pressure used (Edwards, 2010).

This has an advantage of being thinner than the infrared technologies. It also gives another degree of information that can be used to create new and intuitive gestures and inputs. The disadvantage of this is that it relies on pressure, so object recognition may be out of the question, unless the objects are of suitable weight. It also requires material that has been covered in parallel lines of electrodes, which does not come pre-manufactured. These restrictions make IFSR less than ideal for applications within this project. But, the technology may have usefulness in other applications where the ability to read pressure may be a valuable research. An example of this would be in art, where the pressure of a brush stroke will affect the expected result. Currently, only Wacom devices offer this sort of functionality with their use of Electromagnetic Resonance (EMR).

7.1.5 Optical Imaging

Optical imaging has been a popular technology on large scale multi-touch projects for a long period of time. It utilizes a small number of cameras at the corners, just above the surface to be touched. By using simple optical algorithms, the location and size of an object touching the surface can be triangulated.

As the size of the screen increases or decreases, the cost of Optical Imaging remains relatively constant. On very large scale applications, it requires cameras with higher resolution imaging, but a high level of accuracy can be achieved even at currently large screen sizes. This makes it an ideal solution for most of today's current market. But, it is also still limited only to touch. Although it can tell the size of the object touching the surface and does not require any pressure, it cannot tell the shape of the object. Because of this, the size of the object has minimal usefulness.

7.1.6 Kinect (3D Imaging)

For Microsoft's gaming system, the Xbox 360, they created a specialized array of cameras that can create a true 3d image. The hardware within the Kinect can also estimate the position of the user using skeleton when human shapes are being recognized. This allows for interaction with a device without touch it at all. Gestures instead can be made using all or part of the body, as well as relative distance to the screen.

The Kinect technology isn't truly a multi-touch technology. But, there are many applications where it can give an intuitive user experience that allows for interaction by multiple users. These are many of the features that this project aims to do. The software algorithms for use with Kinect have not yet been fully realized and may require knowledge of complex 3d imaging that may be outside of the scope of the project.

7.1.7 Conclusion

For the purposes of this project, the primary concerns are the accuracy of the device and the ability to do multiple types of interactions. Because of this, Diffused Surface Illumination was chosen. It offers the ability to do object recognition through the use of fiducials as well as offering the ability to read touches and common touch gestures such as dragging, pinning, and pinching.

7.2 Active Material

Because DSI requires very specific acrylic material, there are not very many options for the acrylic that actually facilitates the touch sensing. The layer needs to be an edge illuminated acrylic. This acrylic has metal dust mixed in which disperses the light that enters from the side over the entire surface. The amount that the light is dispersed is effective by the amount of metal added. This means that a material that disperses better will also cost more.

Other projects have had significant success with the EndLighten material, which is manufactured by Evonik. Evonik makes several types of all of the acrylic that will be required for the project at competitive prices, so it is an ideal supplier. The EndLighten material comes in several thicknesses and grades.

The thickness of the material does not have significant effect on the the performance of the acrylic according to manufacturer specifications. It will affect the surface area that can be illuminated and the sturdiness of the acrylic. The size of the LEDs, with their mounting, is expected to be in the range of 6 to 8 mm. The Acrylic comes in 6mm, 8mm, and 10mm. To allow for maximum absorbence from the LEDs, a material thicker than the size of the LEDs is ideal. The size of the table is also exceptionally large. To ensure that there is no bending over the material, due to the weight of all of the layers of the acrylic as well as people possibly putting large amounts of force on the table, the maximum thickness of EndLighten will be used.

The grade of the material effects how much light is dispersed through the surface. A higher grade of material will allow for light to penetrate deeper, but may also require brighter infrared LEDs. Tests from other projects have shown that XL, which is normally rated for up to 24” may only obtain up to 17” usable for DSI applications. Due to the size of the table, XXL will be required.

7.3 Diffuser

The projection layer is the layer you actually see the projected image on. Although this may not seem like it is directly related to the touch system, it actually has a significant impact on the performance of the touch system.

By necessity, the projection layer disperses light so that an image can be displayed on the surface. The more light that is being dispersed; the better the image quality can be viewed. This has the negative effect of dispersing input touches and objects. This creates a need for balance between picture quality and touch sensitivity. Although the dispersion of light is in most situations bad, it can also be good for touch sensitivity. Objects that are further away from the surface will be dispersed more, so this creates better differentiation on an objects distance from the surface. The way that the material disperses light may also have an effect. Many rear projection materials do not evenly disperse the light. This creates hotspots around the outside of the projection which may interfere with the ability to read inputs at those locations.

Rear projection material can be broken up into two primary types: film and acrylic. The film materials can be rolled out over a surface and cut with simple tools. They have the advantage of being very thin, cheap, and flexible. These also tend to have a adhesive side that can be adhered to the acrylic touch service. Because of how thin they are, these are known to have hot spot around the outside from the projector and obtain less ideal blobs than the acrylic materials. Acrylic rear projection materials are thicker and more expensive. These tend to come in the range of 5mm or more in thickness. They are sturdier, though they are also known to be soft and scratch easily. These have the advantage of being less susceptible to hotspots from the projector. Tests from other projects have also shown that they produce better quality object detection that the film materials.

The specifications for rear projection acrylic are given in the form of transmission rate, gain, and half-gain. The transmission rate is the amount of light that passes through the material. A high transmission rate means that very little light is dispersed, and the image quality will suffer. A very low transmission rate can also cause its own problems. It may darken the image, as well as obscuring objects and touch events. The gain is the amount of the dispersed light when compared to light reflected off of a Lambertian surface. The half-gain is the angle where the luminance reaches fifty percent of what a Lambertian surface would produce. A Lambertian surface is a surface that follows Lambert's cosine law. This is a surface, that when light reflects off of it, it is dispersed in a fashion that when viewed from an angle, the cosine of the luminance changes at the same speed as the cosine of the area of the surface. This gives the effect that the surface is illuminated equally from any viewing angle. A surface with a higher gain will have a lower viewing angle. But, a surface with a lower gain will not necessarily have a high viewing angle. This is why the half-gain is also important (Salmon).

The Evonik 7D006 material was chosen as the rear projection material for this project. It gives a very clear projected image as well as still supplying the accurate blob detection required. It is also not as expensive as the 7D512 and 7D513 alternatives. It was also chosen because it has been highly tested and other projects have shown to get the quality of object detail that required. This is the same material that was used in the Microsoft Surface, which is the only mass produced multi-touch screen of equivalent scale.

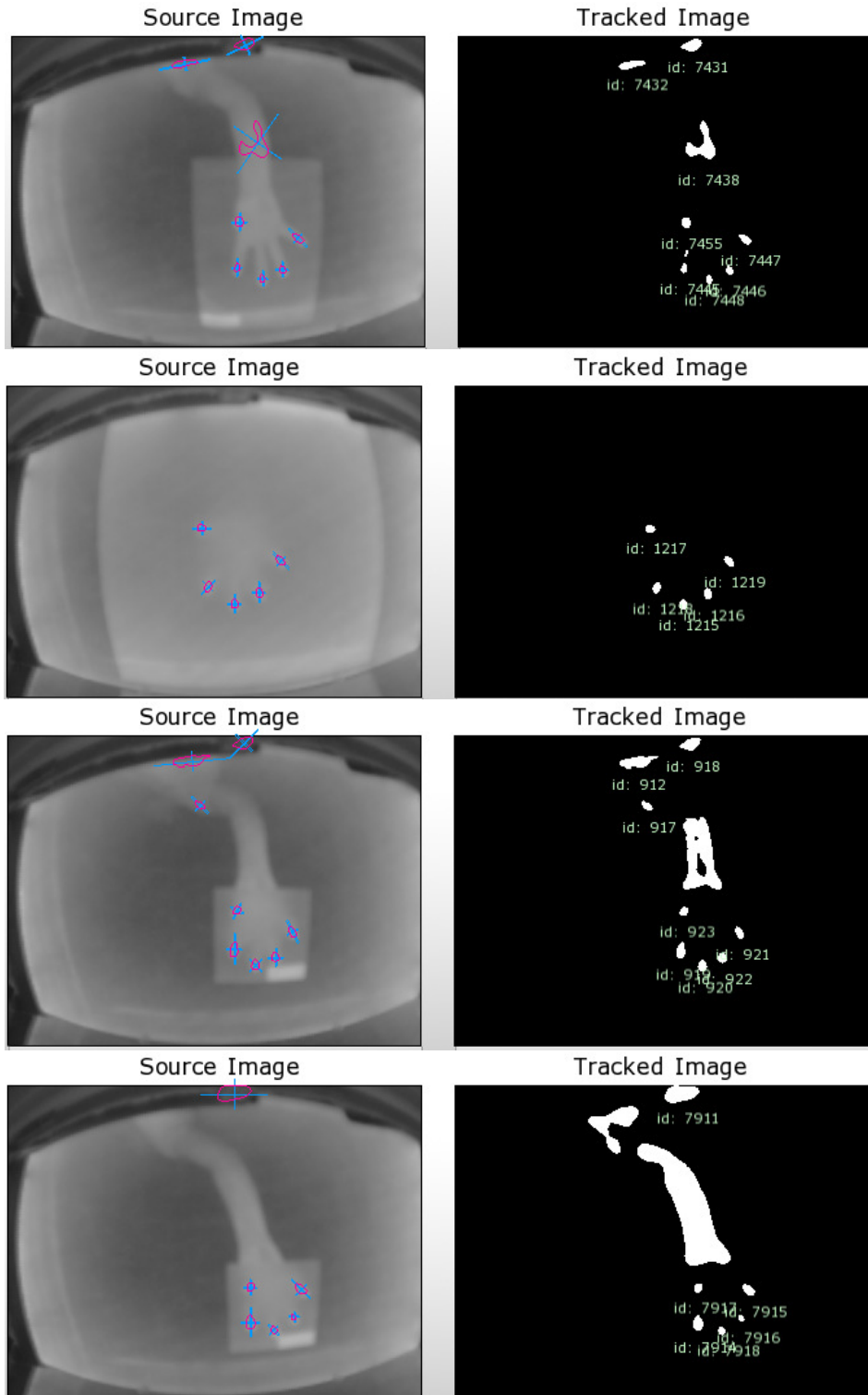


Figure 30 - Inclusion of Various Diffuser Materials on DSI Table. From top to bottom, 0D002 Colorless, 7D512 Light Grey, 7D006 Grey, 7D513 Dark Grey (Ramseyer) (Reprint pending approval)

7.4 Abrasion Resistance

The rear projection material, as well as the active layer, is known to be soft and scratch easily. Due to the nature of the device, it will have constant interaction which could damage the relatively expensive surfaces. This required that there is a third layer of material on top to protect the other two.

Evonik 0A000 MR2 material will be used for this purpose. This is clear acrylic which has a special abrasion resistant coating. Because it is clear, it will have no effect on the projected image or the image detection. It also is shown to have a significantly better resistance to wear as well as being only a third of the cost to replace compared to the EndLighten material.

Evonik also produces EndLighten which has been coated with the same abrasion resistant coating. But, due to the rear projection material being on top of the EndLighten, this layer should not be in contact with the users of the device. For this reason, it is not necessary or optimal to use the EndLighten with this coating.

7.5 Camera

The camera will actually take the picture and transfer that to the computer to be interpreted by software. It is imperative that the frame rate remains high. The goal would be to be as close to 30fps as possible. In order to this, the camera needs to have an efficient sensor that can receive a quality picture in low light conditions. If the sensor is not satisfactory, this can be compensated by raising the exposure on the camera, but this will lower the frame rate. The exposure on many of these cameras can go as high as 1/5 second, but that would lower the maximum frame rate of the camera to 5 frames per second. The other option is to increase the gain on the camera. Raising the gain on the camera is lowering the amount of light that needs to be collected. Lowering the amount of light also means increasing the susceptibility to interference. The resulting picture can come off as “grainy”. This graininess may result in static and artifacts in the picture. The large majority of these artifacts are very small that they can be reliably be removed by software, but only to a certain extent. The issue comes not with the artifacts resulting in unwanted touch events, but rather in the removal of the artifacts resulting in the wanted touch events to be obscured.

There are two major types of cameras that can be used for this application. The first of which is FireWire cameras. The name FireWire specifically refers to the interface in which these cameras interface with the computer, but the majority of these cameras are used for specialty applications. The majority of these cameras are built to be parts of system, similar to how it would be used for this project, rather than as a separate device. Because of this, they tend to be very simple and industrial in design. They also offer significant amounts of control over the type of sensor, lens, and format. This, in theory, could result in better performance. These devices are not designed to be looked at by a user, but rather to give data that can be processed by a computer. Because of this, they

tend to be lower frame rate and resolution than other consumer level cameras. But this also means that they tend to output to the computer raw images as the sensor sees them, with all the processing able to be done by the software. The difference in processing power of the computer would allow for much better processing to be completed.

The second option is to modify a traditional webcam. Webcams are, by design, marketable to the average consumer. This means that the features they focus on are maintaining a frame rate over 30 frames per second, higher resolutions, and convenience. Many of the features on these cameras would have to be disabled since they are counterproductive to receiving quality images. Of primary concern is the automatic focusing and automatic gain control. Once the settings in the device are calibrated to get the optimal picture, every change in the focus and gain could obscure touch events. Many of these webcams also may not have access to manually change these settings, which would make them unusable for the project. Another problem introduced in these cameras is the fact that they are shipped with infrared cutoff filter. Since the cameras are built for visible light and to display as a human eye would see, not as the device itself sees. Depending on the camera, this may be an easily removable filter that can be screwed on and off, or it may be glued or cemented in place, which may require that it is broken off, which may potentially damage the image sensor. The output format of these cameras also tends to be a processed video format. This presents three problems. First, the image may be processed with filters that affect the quality of the image for the purposes of sensing infrared. Second, the video format would be compressed and “lossy”. This means that quality would be lost between what the camera's sensor is capable of, and what the actual camera is outputting. Third, the video format would require decoding and additional computer resources to process.

Of major concern with the camera is that barely any of the datasheets for these cameras lists how these cameras will behave in the conditions needed for this project. Even for the FireWire cameras, this project will be utilizing them in a way that they were not originally intended to be used. Decisions on the camera will be made largely from interpolated data and experiences of other users. It will also likely be the first time the camera chosen will be used for this kind of purpose. Many of these factors will be compensated for with the LEDs. By producing more infrared light, the gain on the camera can be much lower. By using a wavelength closer to visible light, it is more likely that the camera chosen will be able to see the infrared light.

Logitech offers several cameras which use the Universal Video Class specification for their drivers. This specification does not require drivers on most systems. It also offers the ability to change several of the cameras features in a standard way. They allow for all of the auto adjustment features to be turned off and allow for the resolution, exposure, and aperture to be adjusted manually. Many of the cameras also allow for output to be given in YUYV or Bayer raw formats. They also come equipped with wide angle lenses which will allow for a very shallow placement within the enclosure without distortion.

7.6 Optical Low Pass Filter

While cameras are capable of gathering useful information, they also gather significant amounts of erroneous information. Due to this reason, it is necessary to filter out as much of this information as possible. The camera is already incapable of seeing infrared light below a certain frequency; this makes the camera high pass by design. An additional filter can be placed on top of the camera that acts as a low pass filter, filtering out higher frequencies.

There are two cheap and economical ways of doing this, both of which utilize materials that may be able to be readily available or cheap to purchase. The first is to utilize a floppy disk. The magnetic coated plastic that is used to store information has the side effect of being a low pass filter. This material can be cut and placed over the camera's lens. The second is to use developed photonegative. The photonegative must be developed in order to properly function as a low pass filter. But, if the photonegative has an image on it, or was used to take a picture, then that resulting picture will prevent the filter from being an even low pass filter. The actual specifications on these homemade filters will vary and are not firmly known. The cutoff frequency may vary from one disk to another or depend on the development process of the film.

A more reliable method of filtering would be to purchase an optical low pass filter. A cold mirror was chosen for this application. The cold mirror acts as an optical low pass filter by reflecting cold light (less than 750nm) and allowing warm light (greater than 750nm) to pass.

7.7 LEDs

LEDs have many features which will affect not only the performance of the device, but also the effect how complicated the design will be. The main issues effecting performance are the wavelength of the LEDs and the brightness of the LEDs. Most cameras that were considered are designed for the consumer and not specialized applications. Because of this, they are designed to see visible light and may not be able to see the entire infrared spectrum. This means that it is safer to use a wavelength of LED that is closer to visible light rather than being deep into the infrared spectrum. According to the CIE, visible light is in the range of 380nm to 780nm in wavelength. Mid infrared, which is normally thought of with thermal imagery, starts at 1400nm (Schulmeister).

The brightness of the LEDs effects how deep the infrared light can travel into the acrylic. This is affected not only by the actual brightness of the LEDs, but also by the spacing of the LEDs. The brighter the LEDs are, the more outside interference can be reduced. This effects whether or not the table can be used in daylight or well-lit rooms. If too much infrared light is produced, this can cause blowout on the camera, which is where things are too bright for the camera to properly recognize them. Also, the more closely

spaced the LEDs are, the higher the cost and the higher the power usage, which could affect the marketability of the device.

The factors that affect the design specifically are the physical characteristics of the LED. The voltage being applied to the LED chains from the power supply is 12v. This requires that the LEDs be placed in chains, which a certain number of them in series with a resistor to apply the correct current through the LEDs. The voltage required for each LED will affect the number of LEDs which can be placed in each chain. The current going through the LEDs will be constant, with the voltage being modulated. Due to the number of LEDs, this power consumption can also get quite high.

The LEDs are also rated for a half-angle that they illuminate. It is important for the device that the illumination is kept within the acrylic as much as possible. If infrared is able to travel directly towards the objects touching the device and not travel through the acrylic, this can create hot spots around the edge of the surface. But, if the illumination is too narrow, then it will not disperse properly through the acrylic. This can be counteracted by creating a cover over the top of the LEDs that an overlap onto the acrylic so that any excess infrared light is reflected back towards the acrylic. The way the LEDs are mounted may also have an effect on the dispersing of light. An efficient way to mount the LEDs must be made so that they are perpendicular to the acrylic edge so the maximum amount of light enters the acrylic. Because of this, surface mount LEDs on PCBs may be the most reliable format. The area housing the LEDs will also be covered in a reflective or metallic material in order to reflect as much light as possible back into the acrylic.

8. Design Summary

Five systems are in the design of project Planck. These five systems include: the Touch and Fiducial Recognition Software System, the Computational Container System, the Showcase Software System, the Control System, and the Image Recognition System. Figure 31 illustrates the composition of these systems as seen below.

The Image Recognition System gets input from the user and delivers this raw image information to the Computation Container System which contains the computer. This system delivers frames to the Touch and Fiducial Recognition Software System. These are interpreted within software and the gesture object data is delivered to Showcase Software System. From here, the gesture information is used to influence the game logic and this in turn is shown on-screen. The system that displays the information back on the screen is the Computational Container System. Specifically, it uses a short throw projector to do this. The images on-screen are then used by the user to help them decide what they would like to do next.

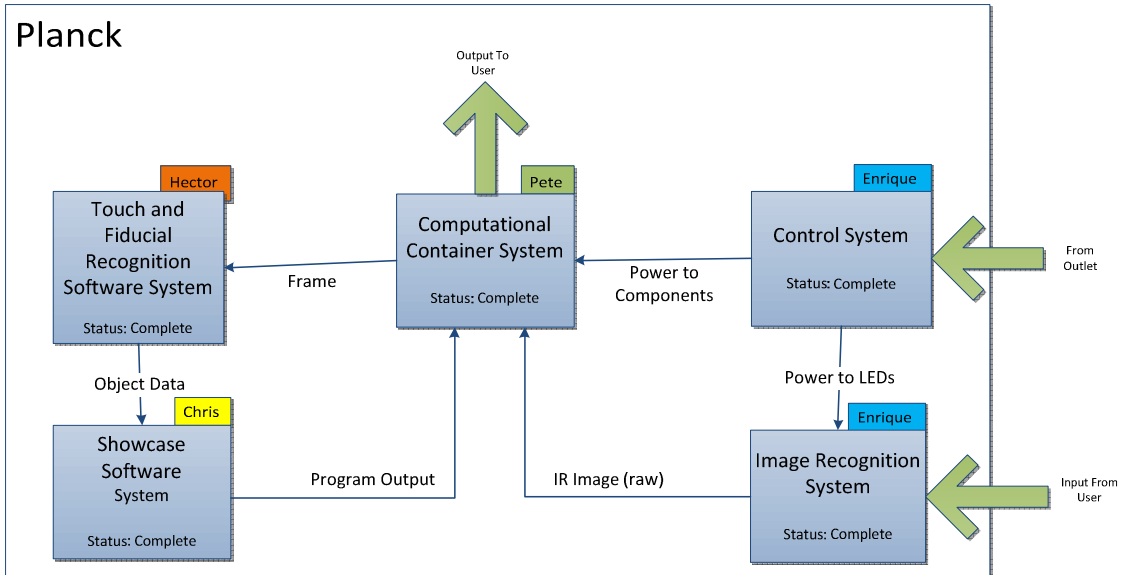


Figure 31 - Top level block diagram

8.1 Touch and Fiducial Recognition Software System

Figure 32 illustrates how the TFRSS system interfaces touches and fiducials with the showcase application. GestureTracker is instantiated as a thread of the showcase application, weDefend. Once GestureTracker is instantiated it begins listening to port 3333 for communications from CCV. The communications received from CCV will come as either touch objects or fiducial objects. If a touch object is received and it isn't in the shared data structure, it is considered a new touch and is added to the shared data structure. If the newly received touch from CCV is found in the shared data structure, its position and other identifying information are updated in the shared data structure with the new properties of that object. If GestureTracker receives notification from CCV that a touch has been removed from the surface, the corresponding object in the shared data structure is removed.

Similarly for fiducials, as CCV notifies the GestureTracker instance that fiducials are removed, moved, or added, those objects are removed, updated, or added respectively to the shared data structure.

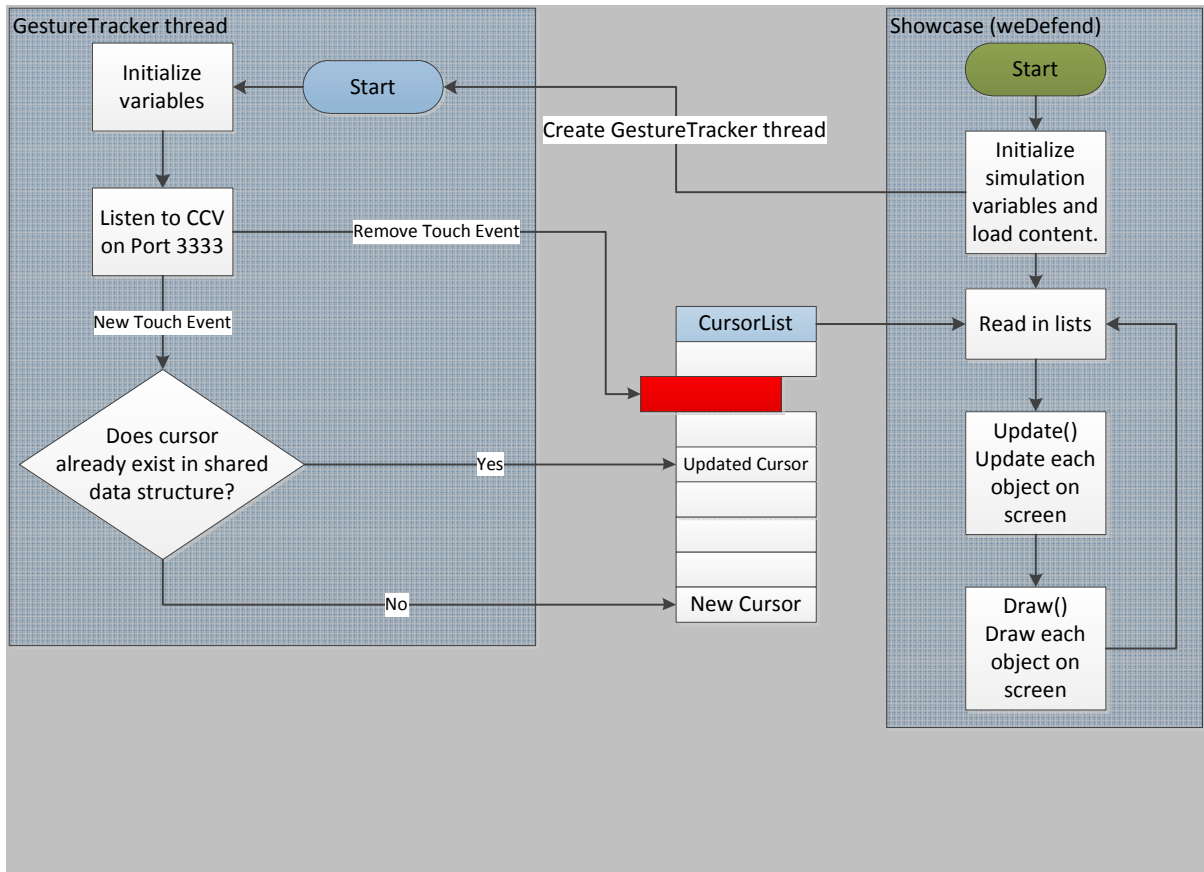


Figure 32 – High Level Block Diagram of Touch and Fiducial Recognition Software

The showcase application has two methods show in the class diagram figure below that allow it to pull the list of touches and fiducials from the GestureTracker thread. The WeDefend showcase application uses the data in these two lists as input to the graphical application. WeDefend constantly pulls this list in order to update the graphical objects based on the touch and fiducial data that GestureTracker updates in the shared data structure. The two methods used by weDefend to acquire the shared data structure lists are `getCursorList()` and `getObjectList()` and can be seen in the Figure 33. Since adding, updating, and removing touch and fiducial objects from a shared list violates the atomicity of the GestureTracker thread, locks were implemented to give priority to writing into the shared data structure. `getCursorList()` and `getObjectList()` can acquire the lock when they are run in order to read the list but updating the shared list takes priority over them delivering new data to weDefend. The latency assumed by this process is minimal and has been mitigated by including a very fast processor/memory structure in hardware. The typical number of objects being handled by Planck in general is only in the 30's so performance bottlenecks really aren't an issue. Latency on the device given this input paradigm is minimal and doesn't affect usability.

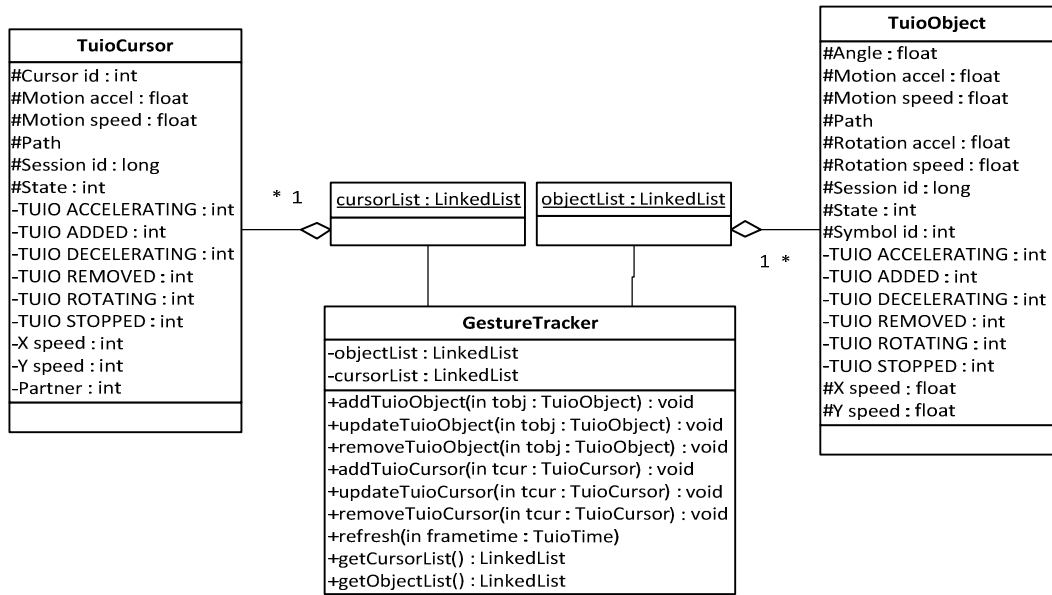


Figure 33 - Touch and Fiducial Recognition Software System Class Diagram

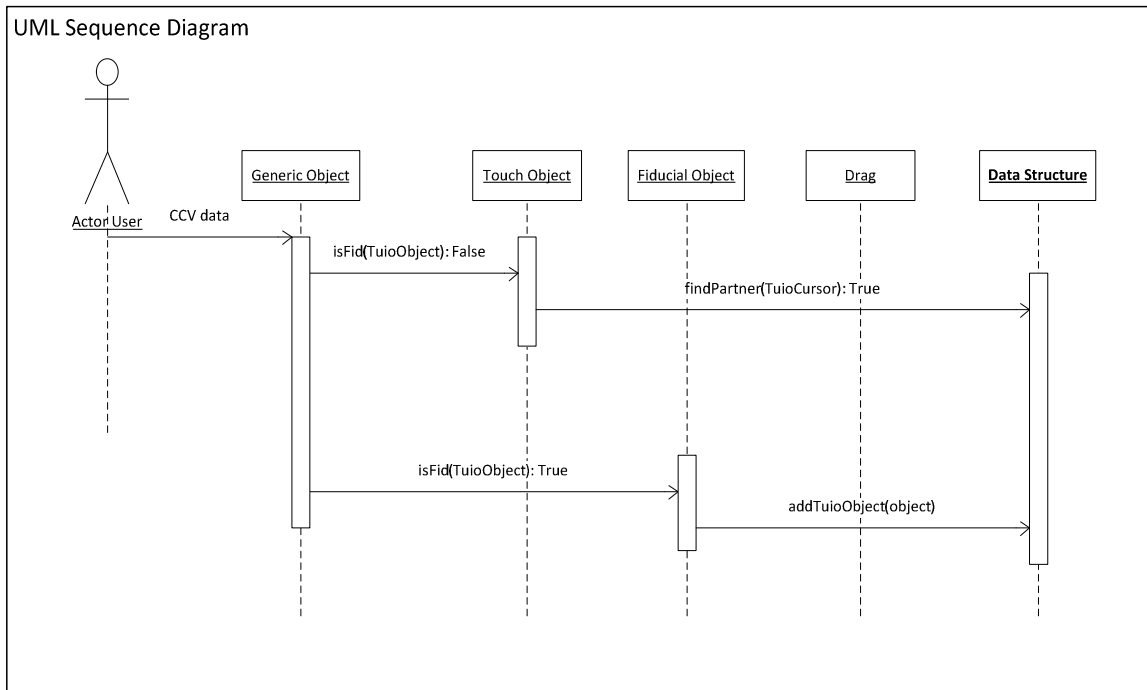


Figure 34 - Sequence Diagram of Touch and Fiducial Recognition Software

The UML Sequence Diagram in Figure 34 shows the life of every process in the application and how they interact with one another and objects in the application. It also

shows how they interact with other processes. The UML sequence diagram is useful to project Planck as it helps understand the flow how Cursor and Object processes travel through the Vision framework, gesture framework, and the showcase application. This is useful when the developers of project Planck begin writing the gesture framework and the showcase application, weDefend. The process of how a user touches a screen to the touch being added to the shared data structure will be illustrated in Figure 34.

8.2 Showcase Software (weDefend)

Figure 35 gives a brief introduction to the layout of the Showcase Software System. The Gesture Recognition Module is a threaded class that will receive TUIO messages and parses them into gestures. The Application included the game logic function and called the drawing function. It also contained many support classes needed for the game to run. The support classes help apply logic to all of the objects on screen. Window Creation, Graphics Processing, Shaders, and Model/Texture Processing are all graphics related tasks and will be used in the drawing of 3 dimensional objects on-screen.

The Use-case UML diagram shows the interaction the user has with the interface of weDefend. The goal of the use-case diagram, from a developer standpoint, is to realize the graphical overview of the functionality that weDefend provides to the users. The diagram illustrates not only where the user is forced to start in the system, but his available options at each step in the showcase application. The game initially starts at a title screen asking the user to place down a preparation mode fiducial. This triggers a state change to the Preparation mode in Planck. Once inside the preparation mode, the user has a wide variety of different options available to him. Initially, the user only has two options the user can take. One option the user can stamp his first unit anywhere on the surface using the Stamp Unit Fiducial object. The other option is to stamp the Action Mode fiducial object anywhere on the surface and the simulation will begin. This is where the objectives of the scenario can be achieved or failed.

Now that the first two possible steps have been illustrated, the continuation of each of the previous steps will be expanded. If the user decided to place the Stamp Unit fiducial, the user then has three new options available to him. The first, he can execute the Field of View (FOV) gesture on the unit. The second, he can execute the Patrol gesture on the unit. Third, he can execute the Lasso Group Pin gesture on the unit. If the user decided to stamp the Action Mode fiducial and enter the start of the simulation, then the user would only be able to execute a FOV gesture or stop a patrol. The user cannot go back to Preparation Mode. Once one hundred insurgents reach the protection zone the user will be presented with the game over screen at which time he or she must place down the reset fiducial to be brought back to title screen or if they become bored they can reset to the title screen using the reset fiducial in the action mode.

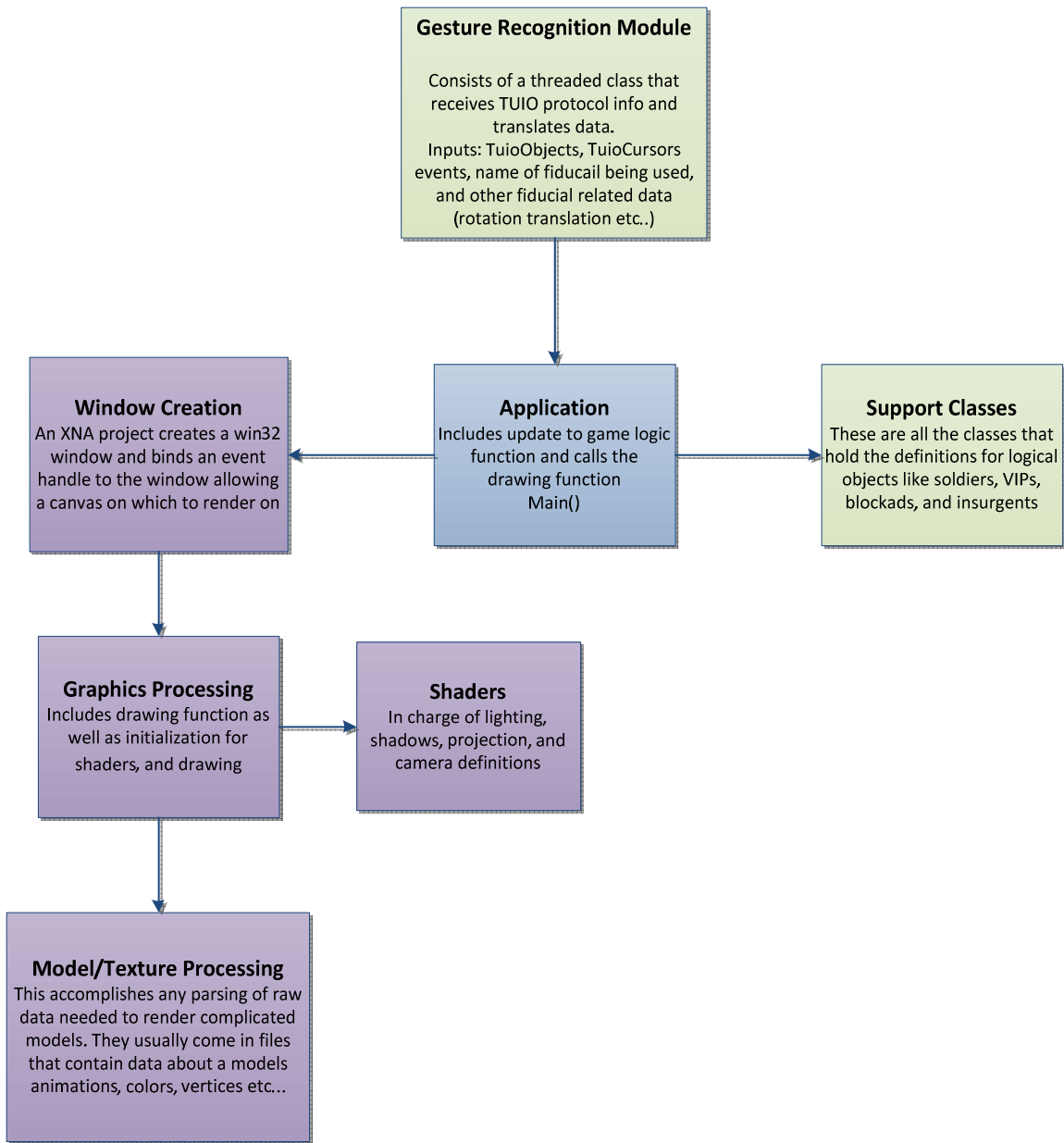


Figure 35 - Block Diagram for weDefend

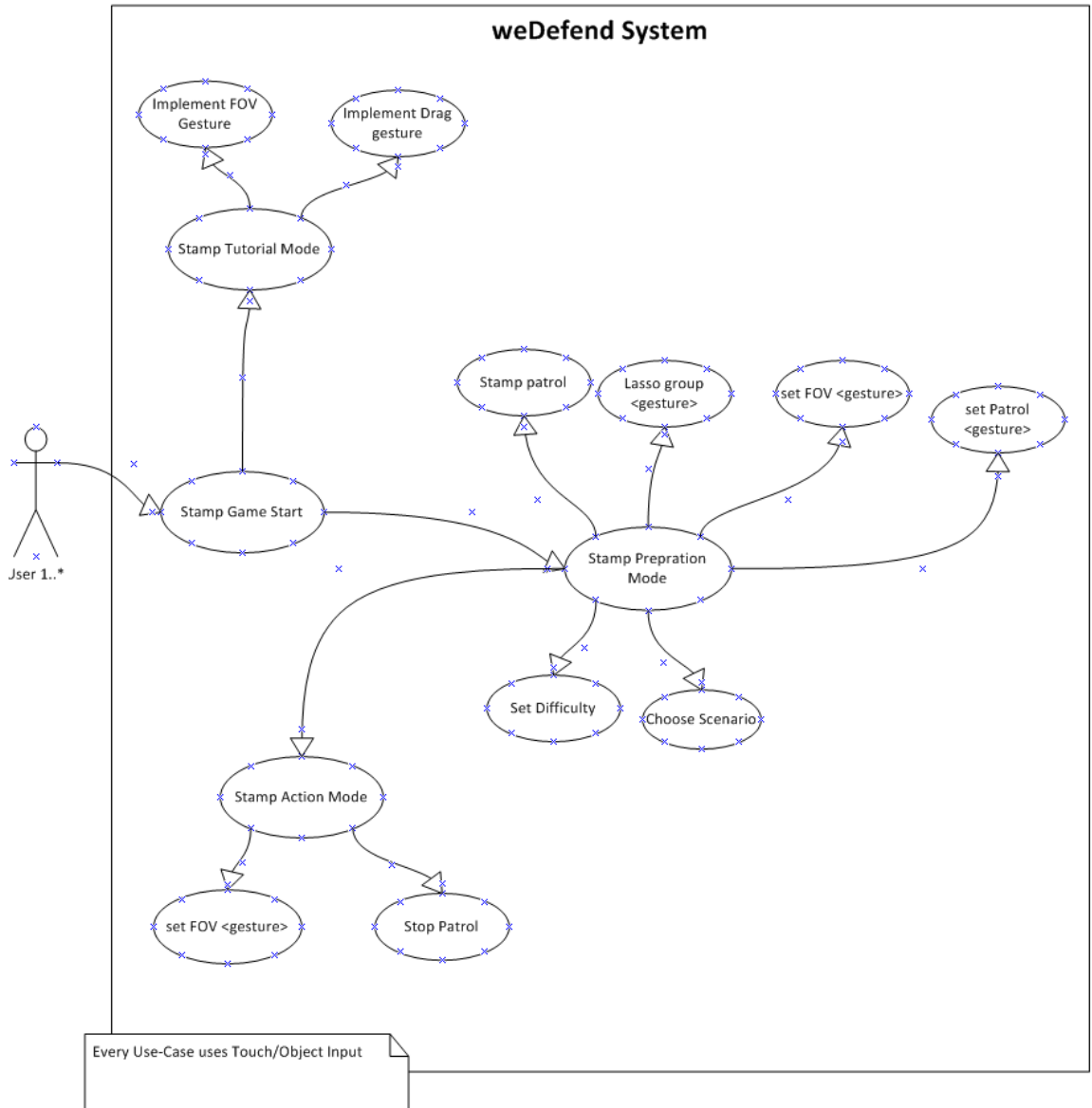


Figure 36 - weDefend Use Case

The UML State Diagram shows all the different states of an application. This can be used to realize flow of the application and show how the different states affect both the user and objects in the application. Four states were realized in the writing of the State diagram. When weDefend is booted up it is in the title screen or main menu state. This is the initial state of weDefend. From here the user can put down the preparation mode fiducial which will send weDefend into the preparation state. Once the user enters Preparation mode, he has a wide variety of options available to him. This can be viewed in the Use Case diagram. At any time the user can return to the start of the application by using the reset fiducial. The user must stamp the Action Mode fiducial from preparation mode to enter Action mode and execute the start of the simulation. Once he enters the Action mode, he cannot return to the Preparation mode. Inside the Action Mode, the user has a few options that are available to him. This can be viewed inside the Use Case

diagram as well. Finally, when the simulation has ended, whether the user won or lost, the application will enter its final state, game over.

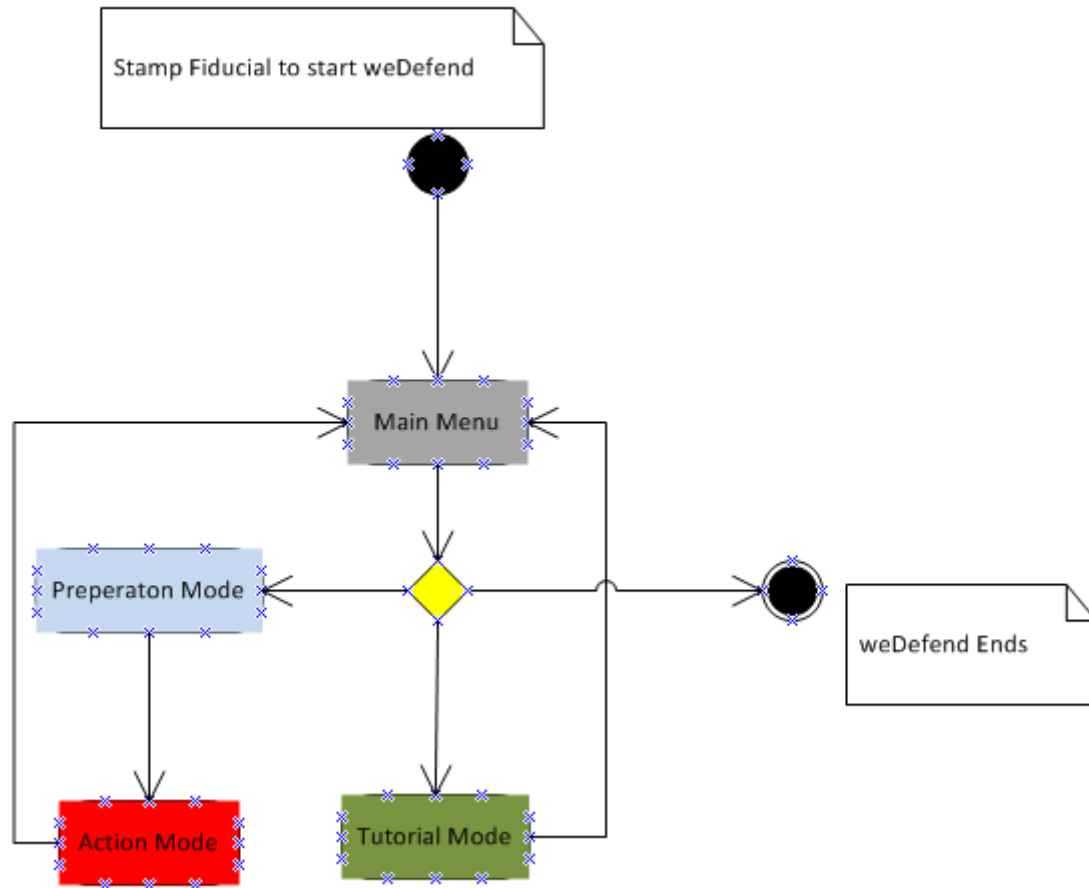


Figure 37 - weDefend State Diagram

weDefend is made up of seventeen classes. Four of these classes are objects which have a role in the scenario, while the others are support classes of the main four classes. The main game1.cs has all the instantiation of our four main classes as well as the finite state machine and the instance of gesture tracker which updates the shared input lists. This class is responsible for holding the majority of the logic and running the scenario in weDefend. The five protected methods are standard methods in any XNA Game Studio application. These five methods are in charge of running the application and are explained in greater detail in the section preceding this one. Initialize is the first of the protected XNA methods. It is in charge of setting up all non-graphic related components like vertex and index buffers, which contain data for drawing primitives, and other parameters that affect the screen. The load and unload content methods are also protected by default and are in charge of binding and sending all textures, images, sprites, models, sound clips, and other assets to XNA's content pipeline. The update function traditionally changes things like position and game data based on mouse and keyboard input. This

method was severely overhauled to incorporate Plank's novel input. The update method was run a finite state machine which uses hexadecimal codes and logical and operations to check for the state this FSM decides whether weDefend is in the preparation phase or the action phase. It would then update all the interactive objects if Plank was in the preparation or action state by calling the interactive objects respective update methods to link gestures to soldiers. Then the unit specific update methods would adjust all the fields of the existing objects given they have an input gesture linked to them. Once these two methods are done isOld method runs to make sure that touches that are linked to objects can not affect other interactive objects as well as check whether the stamp fiducial has been placed to create a new soldier. The draw method will take care of all rendering and is explained in greater detail in the section above.

The other five methods are used to assist the update method in interfacing with, and assuring the state machine is reset when the reset fiducial is placed. The update methods of each interactive object were passed the two shared input lists that gesture tracker adds touches and fiducials to. With these lists they were able to run the linking algorithm as well as update the fields that are being affected by input that is linked. In order to link an input to an interactive object, first it is checked whether or not the linked Boolean of that object is set. If it is, then it was proceed to search for the touch that is linked and update the object based on the updated information of the touch position being a simple example of such. If the linked Boolean is not set then we proceed to search through the list and see if it is inside the bounding radius of the object, if it is we make the link ID of the object equal to the touches ID and set the linked Boolean. When the method is finished each instance of a soldier will have corresponding gestures that are related to them if any input occurred near the object and would have been updated appropriately. The update method would also update the debug status of all debug objects of the debug state is toggled. The debug state is necessary when testing the software on a prototype that does not have a rear projection acrylic so that touches can be seen. It is also a useful tool for assuring that units are behaving as they should.

The two types of fields in the weDefend class are graphics related fields and scenario logic related fields. The graphics related fields define things like the screen width and height as well as scaling factors for sprites, textures, and models. The scenario logic related fields are the bit flags which run the FSM. The other two fields are linked lists called soldiers and insurgents. These two linked lists contain each instance of soldiers and insurgents. It was used to loop through all the instances of soldiers and insurgents so that they can be updated every time update unit state is called and every time they need to be rendered.

The other classes in weDefend are a plethora of support classes necessary to execute all manner of things from debugging to fields of view. Some of the more important classes are the debug Item, FOV, patrol route, polygon drawer, lasso, gesture tracker, and particle. The debug item class is responsible for creating a series of rectangles which contain text that gives information about each interactive object such as position, angle of the field of view, and whether or not the linked bool set. It also is integral in testing weDefend on a prototype with no rear projection acrylic since it places small black

diamonds that denote touches. FOV is a class which contains many of the same functionality as an interactive soldier, however it differs in that it draws a field of view and detects whether or not a touch is inside it with a more complicated geometrical intersection algorithm. Every soldier owns an instance of FOV. Patrol route is similar to FOV because it is an interactive object however its algorithms are almost identical to soldier except for the algorithms that drop waypoints as it is being dragged and then makes the soldier follow those waypoints. Again each soldier owns an instance of patrol route.

The polygon drawer class is a helper class which uses primitive batch to draw things like circles, diamonds, cones, boxes, and other manner of geometrical objects that make up the units, walls and interactive objects in weDefend. Lasso makes use of polygon drawer in order to draw selection squares on the surface of weDefend. Lasso is used to create player groups. Any soldiers that are inside a lasso when a user releases a lasso receive a random color to denote that those soldiers belong to that user. The particle class is in charge of making soldiers shoot bullets at insurgents. It uses simple physics to calculate where an insurgent is going to be in order to collide with him after a set amount of time. Some variability is also thrown in so that soldiers sometimes miss. Finally the gesture tracker class is a thread that is instantiated in weDefend so that it can write to the shared data structures, more information on this class can be found in part three of this paper.

The soldier class contained many important methods and fields. The soldier class has fields which allow all game logic and graphics to be updated. An integer will represent the attack power constant that never changes for any soldier. The field of view distance defines how far from the unit the field of view must be drawn. The field of view angles theta and phi are used to orient the field of view. Theta is the angle relative to the center of the soldier while phi denotes the angle of the actual field of view which changes based on distance. The screen coordinate point object is two integers x and y which defines where the soldier should be drawn on the screen and where he exists in screen coordinates. The unit ID is an integer which counts up from one and is assigned to every instance of soldier that is created. The patrol route object as described above was owned by the soldier to give the soldier class the capability of being put on patrol routes. This data structure had a start point and endpoint to denote the beginning and end of a patrol route. The gesture link ID is an integer which denotes which touch the soldier is currently linked to. A soldier would fire at the first enemy that enters his field of view and would not stop firing at that enemy until the enemy is either eliminated or exits the field of view at which time the soldier will then choose the next enemy in his field of view to attack. The linked Boolean will be used by the update unit state function to tell if any input needs to be resolved for a particular soldier instance.

The methods for the soldier class are a series of overloaded update and draw functions that update or draw based on what state the game is in and what parameters are needed in each state. The soldier also has a follow patrol route method which it uses to move along patrol routes autonomously based on the list that is returned by the patrol route puck. The update debug method continually sends the debug information from fields of the soldier to the debug item that has been assigned to that soldier. The update cursor method is used

to be small enough to fit through a doorway. The enclosure had to be tall enough to use while standing. The enclosure had to stay cool enough to keep all the inside hardware from overheating. Finally, the design of the screen had to resemble that of a touch-screen smart-phone. This is detailed as an acrylic sheet that goes wall to wall on the surface of the enclosure.

The enclosure will be designed to be tall enough for the average user to run the application standing up. With no real data to base our potential client's height off of, estimation was needed. From personal experience, an average height of 5'10" for males is appropriate and 5'3" for females. . Because of this, the table was designed for an average height of 5'7". The group found that a total height of 36" was appropriate as it allows users taller than 5'7" to still touch the screen without having to bend over and allow users smaller than 5'7" to still comfortably reach the screen. A total screen size of 46" diagonal was chosen as its big enough to comfortably use the main application but not too big where transportation would become a concern.

8.3.1 Image Display

Planck will use a short-throw projector to provide the display image for viewing the showcase application. Much research was done on the image display but the feasibility of a short-throw projector could not be over-looked. No mirrors need to be used to maximize throw distance; the enclosure height is not of concern as short-throw projectors can achieve a big image in a small space; and there is no risk of damage as no hardware components needs to be removed and re-layered as in LCD televisions. The short-throw projector meets the requirement of a resolution at least 1280x720. There are no other drawbacks to a short-throw projector.

Short-throw projector choices were narrowed down to three. While previous multi-touch users have disregarded manufacturer's specifications and mounted the projector vertically, the decision was made to obey all manufacturers' specifications. They address these issues for a reason, because they do not feel their product will work the way they intended it to. This decision ruled out the Mitsubishi WD380U-EST. The Hitachi CP-AW251N was chosen as the projector for Planck over the Sanyo PDG-DWL2500 for many reasons. This list includes greater feedback from the NUIgroup community; less power consumption; smaller dimensions; LCD technology over DLP, which is currently favored for its sharper image; less weight; and it comes with better built-in support to correct Keystone digitally.

The projector was mounted vertically against the wall of the enclosure. Collecting information from the manufacturer's manual, the vertical offset and throw distance required to achieve a 46" diagonal display were found. This information can be found in Figure 39. Using trigonometry, the angle that defines the vertical offset was calculated. This is seen as α in Figure 39. The enclosure height required is 13.59 inches. The border, defined as the distance between the edge of the enclosure and beginning of the display image, must be 8.9" down one single side of the length of the enclosure.

With the vertical offset angle known, the measurements can be calculated when the projector is tilted, to minimize the vertical offset. A modest tilt of 11.52° shortens the border to 8.6" and increases the height of the enclosure to 17.15". The measurements and view of the tilt can be found in Figure 40. While the height is of little concern, the border is barely minimized. This is due to the greater width the projector takes up from the tilt. A greater tilt would reduce the border distance more but the ability to correct the Keystone image becomes a concern. Of great importance is the fact that manufacturer states all measurements account a chance of $\pm 8\%$ error. Because of this, more time will not be put into finalizing the exact tilt of the projector. Instead a precise tilting mount will be constructed for the projector. The projector will be mounted and the exact tilt required will be determined at the time of install.

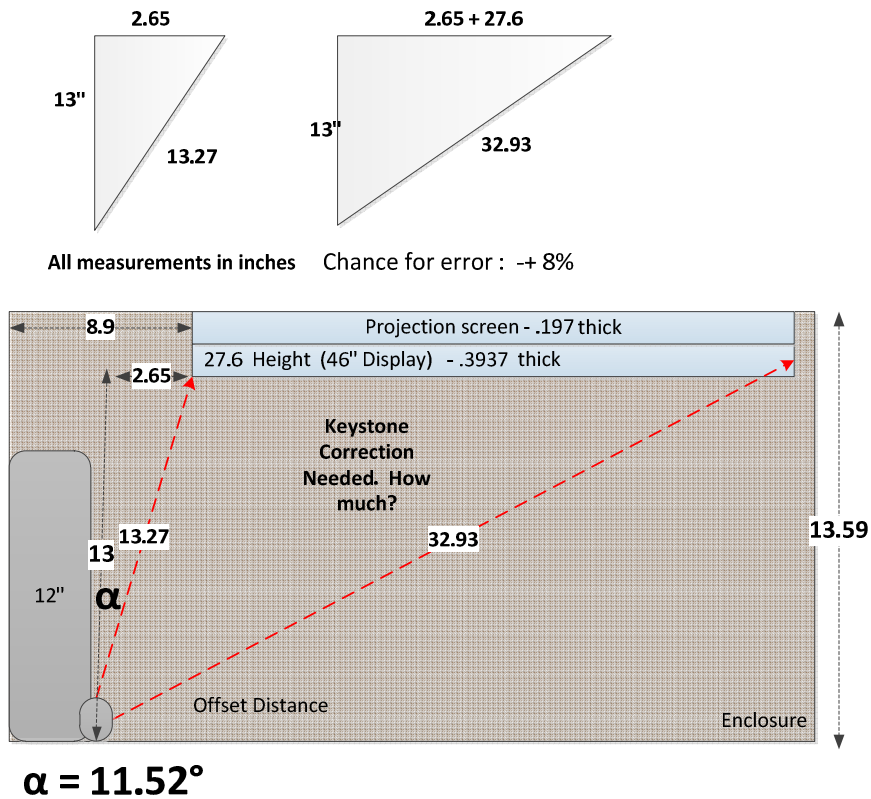


Figure 39 - Projector Placement

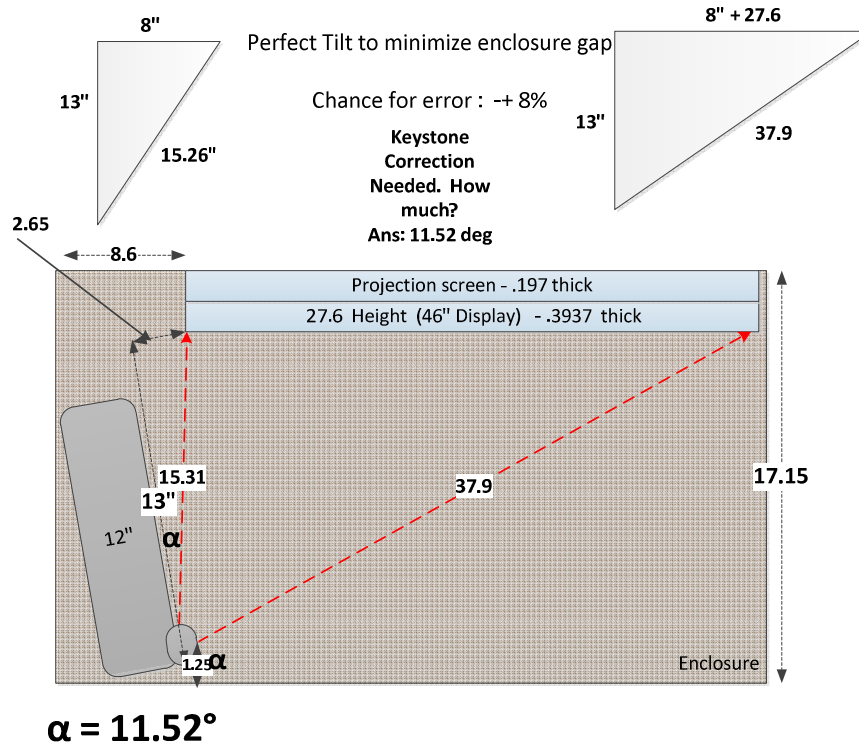


Figure 40 - Titled Projector Placement

8.3.2 Computer

The computer that was chosen for the system has a quad-core i7 Intel. The video card supports DirectX 11. To effectively take advantage of Intel's aggressive prefetching scheme, the computer has at least 4GB's of main memory installed on the system. The hard-drive should be a SSD to boot up with minimal delay for faster use. Figure 41 shows the computer build that was bought for the project. An Intel quad-core 2nd generation i7 2600K was picked and paired with a compatible Asus ATX motherboard. Current ATX boards only support memory at speeds of 1333mhz so 8gb's of G.Skill was picked as prices have dropped so rapidly on memory that it's getting hard to find anything less at a cost-savings. An additional 4gigs of memory only helps future-proof the system and is well worth the additional \$12 cost of only 4gigs of RAM. A budget friendly DirectX 11 video card was chosen to provide to power the graphics of the main application. A Cooler Master 700W Bronze efficiency Awarded power supply was chosen to power the computer, IR LED's, and enclosure fans. While a 450-500W power supply would probably be enough to power the computer itself, it was important to take into consideration the power draw from the fans and IR LED's. It's also important to note that power supplies consistently don't deliver the wattage claimed. Starting with an estimated wattage of 500W for the computer, another 100W allocated to the LED's and enclosure fans, and a 15% chance of error leaves a total power requirement of 690W. A 700w power supply meets these requirements. Finally, a Solid State Device hard drive was chosen because of their faster read/write speeds over a magnetic hard drive. They

are also more reliable than the failure-prone magnetic hard drives. An OCZ Technology Vertex3 Series 120gig was chosen for their claimed read time of 550mb/s and write of 500mb/s. This boots up Windows 7 faster than a magnetic drive allowing the user to start using the showcase application more quickly. This is because the OCZ SSD has an average read throughput of 500mb/s vs. 108 mb/s for a Western Digital Caviar black with 64mb cache. This is important as the user should have to wait a minimal time as per specification.







Qty.	Product Description	Savings	Total Price
1	 <p>COOLER MASTER Silent Pro M700 RS-700-AMBA-D3 700W Power Supply Item #: N82E16817171037 Return Policy: Standard Return Policy</p>		<p>\$439.99 \$109.99</p> <p>ADD TO CART</p>
1	 <p>GIGABYTE GeForce GTX 460 (Fermi) GV-N460OC-1GI V3 Video Card Item #: N82E16814125412 Return Policy: VGA Standard Return Policy</p>		<p>\$139.99</p> <p>ADD TO CART</p>
1	 <p>OCZ Vertex 3 Series - MAX IOPS Edition VTX3ML-25SAT3-120G 2.5" MLC Internal Solid State Drive (SSD) Item #: N82E16820227714 Return Policy: Limited Replacement Only Return Policy</p>		<p>\$299.99 \$224.99</p> <p>ADD TO CART</p>
1	 <p>G.SKILL Ripjaws X Series 8GB (2 x 4GB) 240-Pin DDR3 SDRAM DDR3 2133 (PC3 17000) Desktop Memory Item #: N82E16820231468 Return Policy: Memory Standard Return Policy</p>		<p>\$59.99</p> <p>ADD TO CART</p>
1	 <p>ASUS P6X58-E PRO ATX Intel Motherboard Item #: N82E16813131755 Return Policy: Standard Return Policy</p>		<p>\$219.99</p> <p>ADD TO CART</p>
1	 <p>Intel Core i7-2600 3.4GHz LGA 1155 95W Quad-Core Desktop Processor Item #: N82E16819115071 Return Policy: CPU Replacement Only Return Policy</p>		<p>\$299.99</p> <p>ADD TO CART</p>
Grand Total:			\$1,054.94

Figure 41 - Detailed Computer Components

8.3.3 Enclosure

The enclosure was made out of ¾” thick Oak veneer. Oak was chosen because of it being a strong hardwood and it being visually appealing. Veneer was chosen because of its cost effectiveness and because it comes in large 4’x8’ sheets. Solid hardwood comes in much smaller sizes which required a lot more gluing and is more expensive.

The enclosure was constructed essentially as an open-faced box. A door was not included as calibration can be conducted without it, and it saves on construction time. Unlike the prototype where a frame was constructed to house the LED's and acrylic, the frame was eliminated and the acrylic and LED's were mounted directly to the sides of the box. This eliminates a step and allowed us to achieve the specification of achieving the smart-phone appearance. Eliminating the frame shortened the border and allows the surface acrylic to span edge to edge on the enclosure. This can be viewed in Figure 42. The acrylic is still removable from the box, one layer at a time. The dimensions of each piece of the box are as follows:

Ends: 40.75"x19.75"; .75" dado cut at the bottom; .34" dado cut, .2" from the top.

Front/Back: 45"x19.75"; .75" dado cut bordering the bottom, the sides; .34" dado cut, .197" from the top.

Bottom: 40.75"x44.25"; .75" dado cut bordering all sides.

Just like the prototype, the same dado design was used to piece the box together. Using a dado blade provides many benefits such as allowing an exact fit of each piece of the box; it's easy and fast to piece together as it requires no glue and minor clamping; doesn't require any angles to be set as its being glued; and provides a stronger joint which ultimately provides a stronger box. A door was not installed in the box. This was a big change from initial research. The ultimate reason for not installing the door was to save on construction time. It was determined that having a door would not help in the calibration phase. Rather than install the door to calibrate and fix constantly breaking parts, the idea was to pick parts that won't break and make the correct installation on the first try. This falls back to the requirement of everything being mounted securely. The open-phase box also allows for the acrylic to come on and off the box easily. This allowed for maintenance to the box if needed.

The enclosure was designed to accommodate a channel for mounting of the LED frame. This is much the same design as the channel cut for the prototype. The difference is that the channel is built directly into the side of the enclosure rather than on the top frame that was built in the prototype. The channel was cut with a dado blade. The channel was as thick as the EndLighten acrylic, 10mm. The channel was cut 3/8 deep into the sides of the enclosure, half thickness of the wood. It was 1/2" wide, a measurement decided upon based on the success of the prototype. There were several issues that arose in the wooden LED frame in prototype. These include extensive soldering time, large room for error, and exposed wiring leaving easy for accidental breakage. Rather than re-use this design, project Planck will incorporate PCB boards into the design of the LED frame. The PCB will be 10mm tall and surrounded all sides of the EndLighten acrylic. The PCB had holes drilled in equally distributed intervals down the entire frame. The spacing of the holes was determined by the design of Image Control System. With the frame built, drilled, and the LED arrays mounted inside, the frame will be mounted snugly inside the channel of the enclosure. From there the acrylic can begin to be stacked on top of each other. The LED channel and acrylic layering can be viewed in Figure 42.

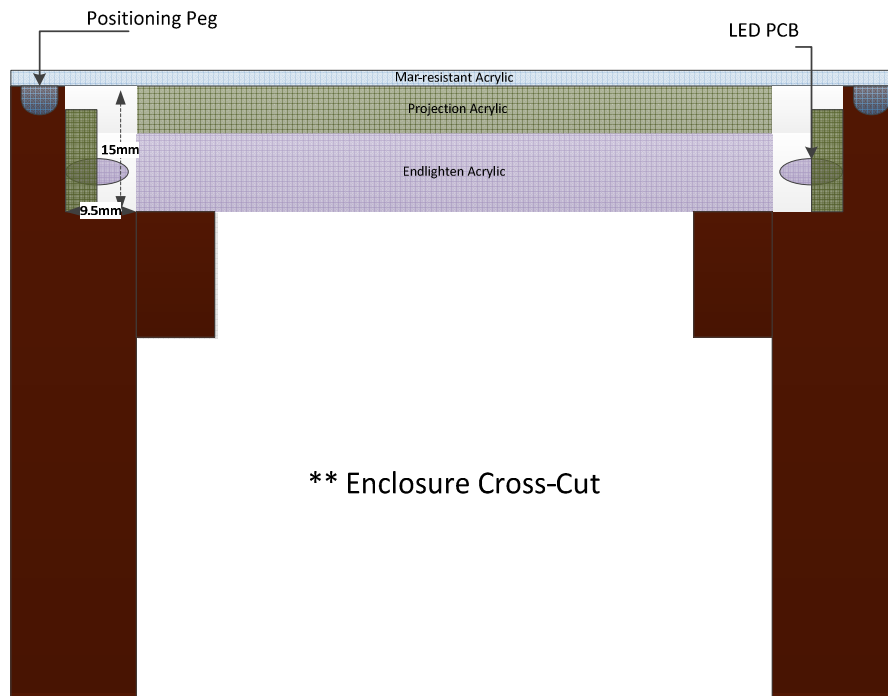


Figure 42 - Enclosure Cross Section showing LED Channel

Extra wooden blocks will have to be secured to the inside of the box for the EndLighten acrylic to rest on. The only requirement on the size of the blocks is that they cannot be wider than 2". If they are any bigger they may start blocking the projector's image. Finally short acrylic rods will be welded onto the mar-resistant acrylic to act as a method of fastening the surface acrylic to the top of the enclosure. This prevents movement of the acrylic when in use. The rods were of approximate length of 3/8" and were fastened using glue specially formulated to cause a chemical reaction that welds acrylic to acrylic. The rods were mounted inside 3/8" holes drilled into the side of the enclosure.

8.3.4 Enclosure Features

Three fans were mounted inside the enclosure to dissipate built up heat from the various hardware devices. The fans chosen are two Artic cooling AF12PWM 120mm case fan and one Panasonic Panaflo 80mm. This delivers 57 CFM at 1350rpms with 0.5 Sone of noise being produced. The fans are Fluid bearings as decided as the best choice in research. They are claimed to have 120,000 hours of operation to ensure little maintenance on the system over the years. The Artic Cooling fans also have the fourth wire for Power Width Modulation (PWM). The position of both fans can be viewed in Figure 44.

All three fans were mounted on the bottom of the box for a stealthily appearance. It was deemed unfit to have hot air blowing on the legs of our customers. One AF12PWM was

mounted directly underneath the power supply, the second underneath the outtake of the projector. The Panaflow has along PVC pipe attached to the intake that allows for the Panaflo to draw hot air that collects at the top of the box. Rather than drilling more holes for optional fans like other projects have done, the need was noted, but not included. If the enclosure does not fall under satisfactory temperature level conditions during testing, fans of the exact models as the previous will be bought and installed in locations directly next to the original locations. This is an easy task and prevents the possibility of holes being designed, drilled, and in the end, never used. The fans were connected to the MSP40 in the Control System for power and received fan control based upon the MSP40's temperature sensors. This was an accurate way of judging the enclosure temperature and controlling the heat.

The projector needs to be mounted to allow precision tilting. To accomplish this, the projector was mounted to a 15"x13" wooden board. The wooden board was glued to a second, very skinny piece of wood to effectively elevate one side of the projector. This allowed tilting of the projector in an approximate ~20 degree angle. This was chosen over metal hinges and a piano hinge due to the simplicity. Rather than go through the math to calculate exact measurements for position and angle of the projector, that will surely fail, a more estimated approach was taken. The projector can be zoomed in/out and built-in hardware can correct for the Keystone effect when the projector is mounted at an angle. We used both of these functions for exact calibration after mounting of the projector. The projector mount can be seen in Figure 44.

The computer was mounted to an old computer case motherboard tray. The tray was mounted down using screws and provided holes already on the tray. With the tray secure, the motherboard and all parts connecting to the motherboard can be mounted as any other typical setup. All relating computer hardware was mounted directly into the motherboard. The positioning of the computer was off to the side of the enclosure and directly next to the exhaust fan. The exhaust fan aided in dissipating the heat built up from the computer. The power supply that powers the computer and various other devices will be mounted on the wall of the enclosure directly above the exhaust fan. The exhaust fan will funnel all heat from the power supply directly outside the table. The location of the computer, power supply, and fans can be viewed in Figure 43.

To meet the requirement of the enclosure being 36" in total height, legs were designed. This is because of the efficiency of the projector. Rather than make the enclosure bigger than it needs to be, the enclosure was built to be as minimal height as possible. Legs were designed and built to provide the height the system requires. An extra benefit to this design is the possibility of shortening the system on a short notice. The legs were designed to be removed with minimal work. The legs were made out of the same wood as the enclosure, oak. The legs were 12" in total length. There are no requirements of visual appearance, only that it can support the weight of the enclosure and 4 grown adults leaning over the system. To meet this requirement, the legs were fashioned out of 4x4's. Each 4x4 has a wooden peg mounted on the surface. Four holes were drilled into the bottom of the table. The pegs slide into the table for a secure, accurate fit. The legs can be viewed in Figure 44.

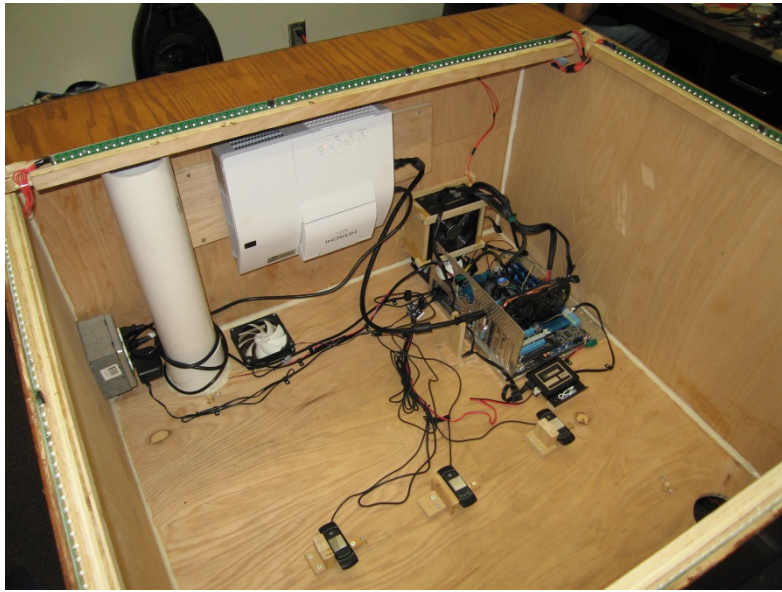


Figure 43 - Placement of Fans, power supply, and Computer

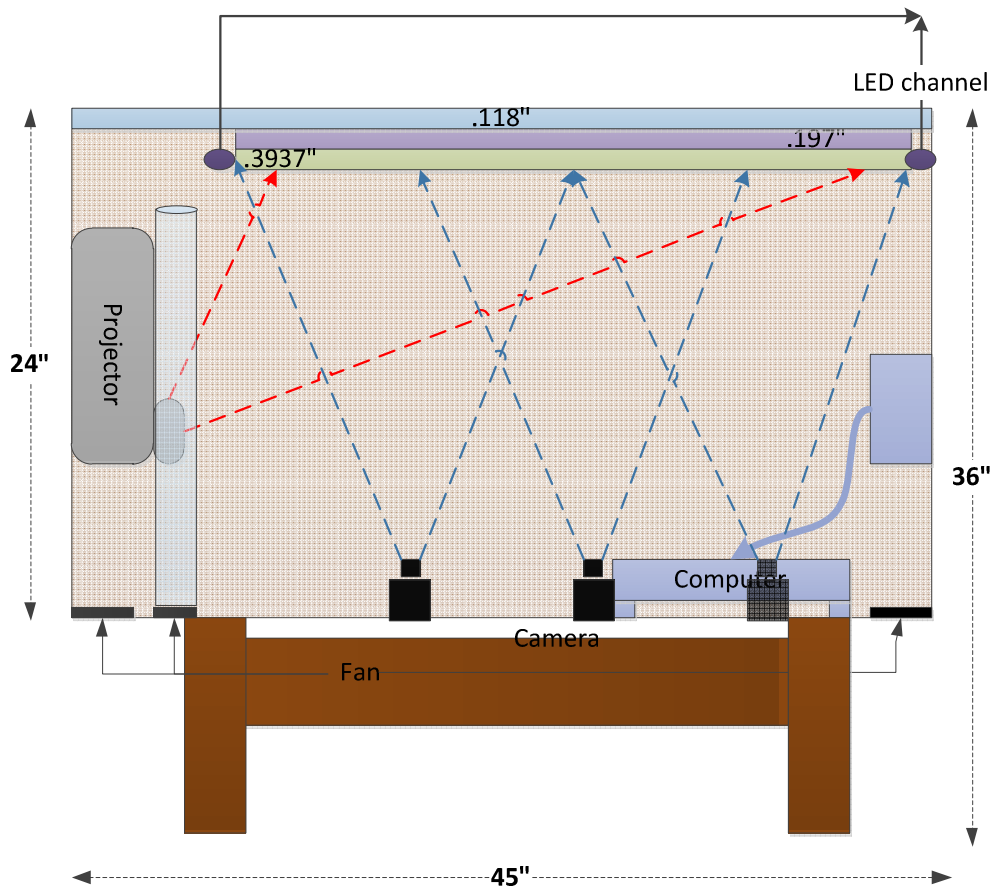


Figure 44 - Hardware Placement inside Enclosure

8.4 Control System

The entire control system was powered by the 12 volt rails on the computer's power supply. The specifications of the 700 watt power supply chosen are that it is capable of producing the entire 700 watts over its single 12 volt rail. The MSP430 microcontroller operates at a maximum of 4.1 volts, but requires that at least 2.2 volts to allow for all clocks to be active, so a basic voltage splitter will be used to bring the voltage down to its recommend voltage of 3.3 volts. This will allow it to have maximum frequency of the clocks and allow it to have maximum current out of the output pins. The potentiometer input to the MSP430 will also act on the 3.6 volts. The input current into the analog to digital converter on the MSP430 requires the current be below 1ma. Due to this restriction, the potentiometer was 50k Ω resistance. The TMP37 temperature sensors were hooked up to a 5 volt line. The temperature sensor outputs at 20mV per degree Celsius, and the produced current is well below the maximum allowable by the MSP430. The clock rate of the MSP430 was set to 8 Mhz, and ran in an approximately 3200 cycle loop, or 2.5Khz. The input values from the potentiometers connected to the analog to digital converter were converted to a digital value that modulated the output pins. These temperature sensors were connected to pins 2 and 3 of the MSP430. The modulated signals were outputted from pins 8 and 9. Pin 8 outputted to the 4th wire of the enclosure fans. These fans require that an input signal between 100 Hz and 25 KHz be used to minimize noise. The 9th pin will output to the LED array. Two bipolar transistors connected as a darlington pair were used to power the LED array.

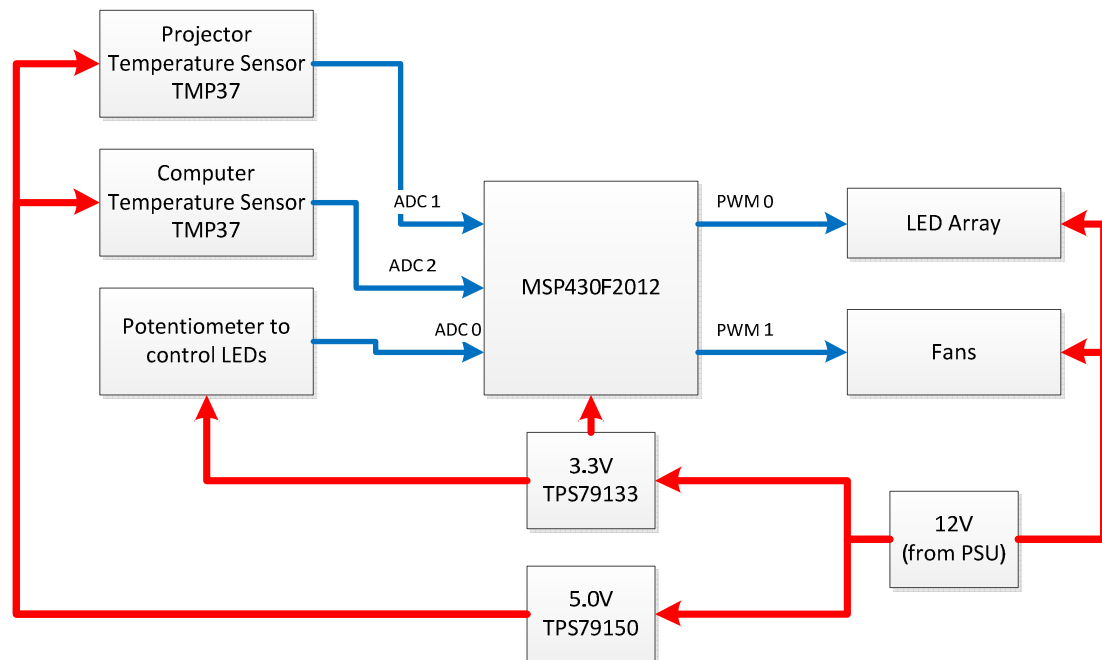


Figure 45 – Overview of Control System

The LEDs were connected as 7 LEDs and a current limiting resistor in parallel. The LEDs used were surface mount OSRAM SFH4258 LEDs. These operated at 850nm with a 15 degree half angle. These were made into PCB chains that could be daisy chained

with lengths of 7, 10.5, and 14 inches, with the LEDs spaced half an inch apart. This can be seen in Figure 46.

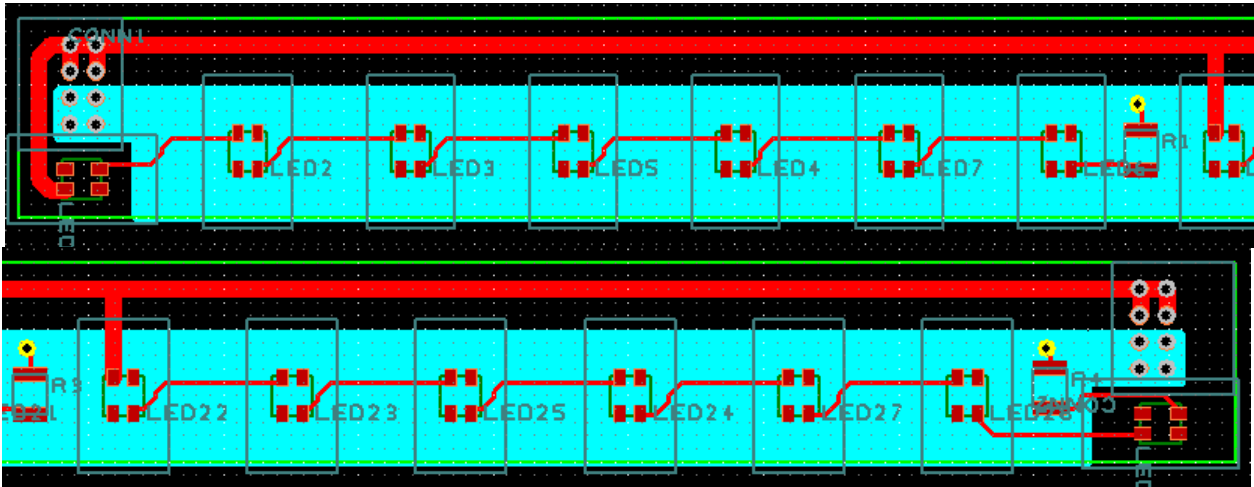


Figure 46 - LED Chain PCB Layout

9. Future Work

The showcase application and gesture recognition module can be expanded in the future to further the idea that traditional input and menus are not needed for complex applications. Expanding weDefend to incorporate more complex unit interactions and abilities can lead the way for creative new gestures and uses for fiducials. In the showcase application a lot of attention to the details of complex units and interactions with the terrain is considered. Project Planck would proceed in designing and developing a more complex application with an immersive user experience similar to what mainstream entertainment industry RTS titles strive for.

Adding many of the special abilities discussed for different units would push the gesture development team to make more intuitive gestures that users could execute to avoid menus and buttons that are typical in most applications. Some gestures could include making small circles or using multiple fingers in order for soldiers to throw grenades. Entire hands could be used to stop all patrolling units on the map. Splitting teams during the action phase could be accomplished by using speedy finger drags in conjunction with the acceleration data provided by reactIVision across a selected team. These extra gestures would also push the team into areas yet to be explored in the multi-touch domain due to the minimal amount of these devices being used commercially. Mitigation of multiple user interaction in complex applications is a glaring weakness in multi touch technologies. However this weakness can be overcome with time and clever use of scheduling algorithms common in modern operating systems.

Fiducials could also be greatly expanded upon as hi-tech tokens that carry information on a per user basis. Some developers have created multi-touch surfaces that also respond to infrared light being shined on the surface. This could be used to expand the domain of fiducials by not having them printed on paper but as a device with infrared LEDs that shine the same fiducial shape onto the surface. They can be extended to create an ownership system of touches and events on a multi touch system. Their use in applications is also apparent when looking at larger more complex units in weDefend. The ability to have air craft carriers that launch planes off the deck by using a fiducial or rotating the guns on a battle ship are the simpler and more obvious uses of fiducials in a RTS application. This work could lead to multi-touch surfaces taking the place of the personal computer in the future effectively eliminating the need for traditional mouse and keyboard input.

There are many other programs that can benefit from the collaborative nature of Planck and the fiducial inputs it's capable of. Many applications may find benefit from these features, such as education, entertainment, and simulation. The collaborative nature of Planck would be especially useful in a learning environment where many students could interact together and visualize changes. An example may be an interactive demo or simulation in a science museum. Kids of all ages could use objects to change and visualize the results or consequences of their changes over the same display they are interacting with. Similar devices are already implemented in many science museums, but are made for specific exhibits. Planck would offer an alternative that could be used for multiple exhibits, or be updated as an exhibit is updated.

Currently, Microsoft and other partners are exploring uses of the Microsoft Surface in entertainment environments. Devices similar to Planck are currently being implemented in bars, restaurants, and other entertainment venues and are used for advertising. By placing drinks or other objects that may already be at hand, advertisements and other information may be brought up. For example, a store may have items equipped with fiducials, and Planck could be modified to read their barcodes and give additional information on items that are placed on it.

Design programs also currently are limited to a single user. Mouse and keyboard input do not easily facilitate collaboration. If work is to be done collaboratively in these environments, it needs to be broken up into smaller pieces and worked on individually. Alternatively, it can be worked on through a conferencing program or over a projector, which still relies on a single person interacting with the design. Planck's feature set may offer a collaborative work environment where multiple designers can look at a design spread out and edit parts simultaneously.

A huge drawback to many multi-touch technologies on the market is their inability to be used in well-lit areas. This is true of Diffused Surface Illumination technology. While this is not an issue for many, as they just use it in dimmer areas, DSI hybrid technology would allow Planck to be used in any location and at any time of the day. NUIgroup members that have previous experience with DSI have experimented with a hybrid version of DSI. DSI Hybrid is a mix of Diffused Surface Illumination and rear Diffused

Illumination (DI). A detailed explanation of Diffused Illumination can be found in the Image Recognition System. One implementation of the DSI-hybrid is to mount the EndLighten acrylic above the rear-projection layer acrylic. Secondly, mounting directed infrared LED light sources on the bottom of the enclosure that shine onto EndLighten acrylic. This concept is specific to DI technology. This concentrates a heavy source of infrared light onto the EndLighten, making it easier for the IR camera to pick up on touches and fiducials that are placed onto the screen. DSI has been known to detect touches better than DI. DI has been known to detect fiducials better than DSI. The theory is that DSI-hybrid technology would combine the best of both worlds. This would allow excellent touch and fiducial detection. This method will not be implemented into project Planck as of now, but as the project evolves and the need arises, this method will be pursued.

10. Administrative Content

10.1 Roles and Responsibilities

The team recognized that there were roles to play other than the standard project member. Each team member will take equal part in the design and development of Planck, as well as be assigned a specialized role. The following roles were realized as being necessary in the design and build of project Planck:

1. Project Manager
2. Assistant Project Manager
3. Senior Software Engineer
4. Senior Hardware Engineer

The Project Manager's role is to set milestones for the team and to make sure the team reaches said milestone. When a team member requests a meeting, the Project Manager will set the date and time for the meeting. The Project Manager has direct contact with the team's mentor and Dr. Richie. Peter Oppold will be our Project Manager.

The Assistant Project Manager's role is to see to the requests of the Project Manager. If the Project Manager is being overloaded, then the Assistant Project Manager will assist in the workload. The Assistant Project Manager is also delegated the responsibility of overseeing the editing of each team member's documents. Hector Rodriguez will be our Assistant Project Manager.

The Senior Software Engineer's role is to manage all aspects of software as they relate to this project. This includes the showcase application, the gesture recognition software, and any other software that may comprise the system. His duty is to oversee and provide direction for the main application. He will assign development tasks to the team members and check each team member's work for accuracy. The tasks also include setting the standards that will be used for communication between the showcase application and the gesture recognition software. The Sr. Software Engineer is also in charge of distributing

and maintaining our subversion software, GIT. Chris Sosa will be our Senior Software Engineer.

The Senior Hardware Engineer's role is to manage all electronics in the system. His duty is to oversee and provide direction for the electronics used in Planck. The Sr. Hardware Engineer will be responsible for the design and production of the Modulation timer. He will have ultimate decision making powers in the Image Recognition System. He will be managing the production of all electronics in Planck. Enrique Roche will be serving as the Senior Hardware Engineer.

Two more roles were created after project Planck began. The first is the Project Status Manager, or PSM. These responsibilities were assumed by the Assistant Project manager, Hector Rodriguez. The Project Status Manager is in charge of the status of the each group member. Status includes the wellness, measurable work count, member's issues/concerns, and deadline preparedness. Wellness includes moral, physical health, and motivation. A few examples of the role of the PSM includes: The PSM stepping in to address any group member's failure to make meetings on time; the PSM coordinating with the group to inform and re-allocate work because of a group member getting sick or injured; and the PSM checking the status of measurable page count of each group member and addressing any relating issues. This is a task important under the development model Scrum which will be addressed later. Second, the Project Tester role was created to address the issue of testing the System. Testing is a huge component of project Planck. The Project Tester responsibilities will be delegated to Enrique Roche. The Project Tester will manage the test cases for all systems of project Planck. The Tester will manage all developer's that are in the testing phase. The final approval of the Test case passing can only be made by the Project Tester. Finally, he will address any issues that are found from testing. If the test case fails, the Project Tester will create a Spar (see Software Development Mode section for definition) and forward it onto the Senior Software Engineer. The Senior Software Engineer will allocate resources to address the Spar.

10.2 Division of Labor

Project Planck as a whole was split up into five systems. These five systems are the Blob/Object Detection Software, Showcase Software, Enclosure/Computer, and Power/Control and Image Recognition System. Each System was allocated to one member of project Planck. This member is given the responsibility of defining, researching, designing, building, and testing the System. The allocations of the Systems are as follows: Hector Rodriguez is delegated the Touch and Fiducial Recognition Software System; Chris Sosa is delegated the Showcase Software System; Pete Oppold assumes responsibility for the Computational Container System; Enrique Roche is delegated the Power/Control System and the Image Recognition System. These five systems are directly dependent on each other and thus, communications between the systems are important during the design phase. Each member is responsible for communicating with members of another System when needed.

The team meets at least once a week in-person. This is typically after Senior Design class on Tuesday or Thursday evenings. Besides these two nights, the team meets every week either in-person or using a web-streaming application, Skype. Every meeting follows a defined structure. The defined structure includes an opening, body, and closing. The opening is where members will bring attention to any questions, issues, problems, or concerns. This could be concerns about designing their system or concerns about meeting a deadline. The group will be updated with each member's progress since the previous meeting. The body includes the main activity for the meeting, such as editing a paper or working on the prototype: Phloe. This is the majority of the meeting. The closing includes setting a future meeting date and assigning action items to group members.

10.3 Milestones and Timelines

A Gantt chart was created at the start of project Planck to measure the team's performance and gauge their work output. The Gantt chart includes the dates of all formal deliverables as well as milestones leading up to those deliverables. The milestones will be used to track performance during the course of the next two semesters. The ongoing Gantt chart can be found in the Appendix. The first deadline on the Gantt chart is the Initial Proposal document. This is the document used to first propose the multi-touch display, Planck. Next, The Workforce Central Florida presentation on Nov. 18 can be found. The measured work up to that date was presented. The Gantt chart also shows the milestones leading up to the PDR on December 5th. The PDR is the initial submittal of the design of project Planck. First the research was completed, then the design. After the PDR, the Critical Design Review (CDR) date can be found on the Gantt chart. The CDR is the review of the design of the project by the client and mentors. If the CDR meets approval, the construction of project Planck can begin. There are many milestones leading up to the completion of project Planck such as Touchscreen assembly, Touchscreen Calibration, Optical Recognition Development, Device being touch capable, Fiducials Recognized on the Device, Functional Demo of Software, Test Program Development, and Testing on the system. These milestones lead up to the final deadline on April 5th of our Final Presentation.

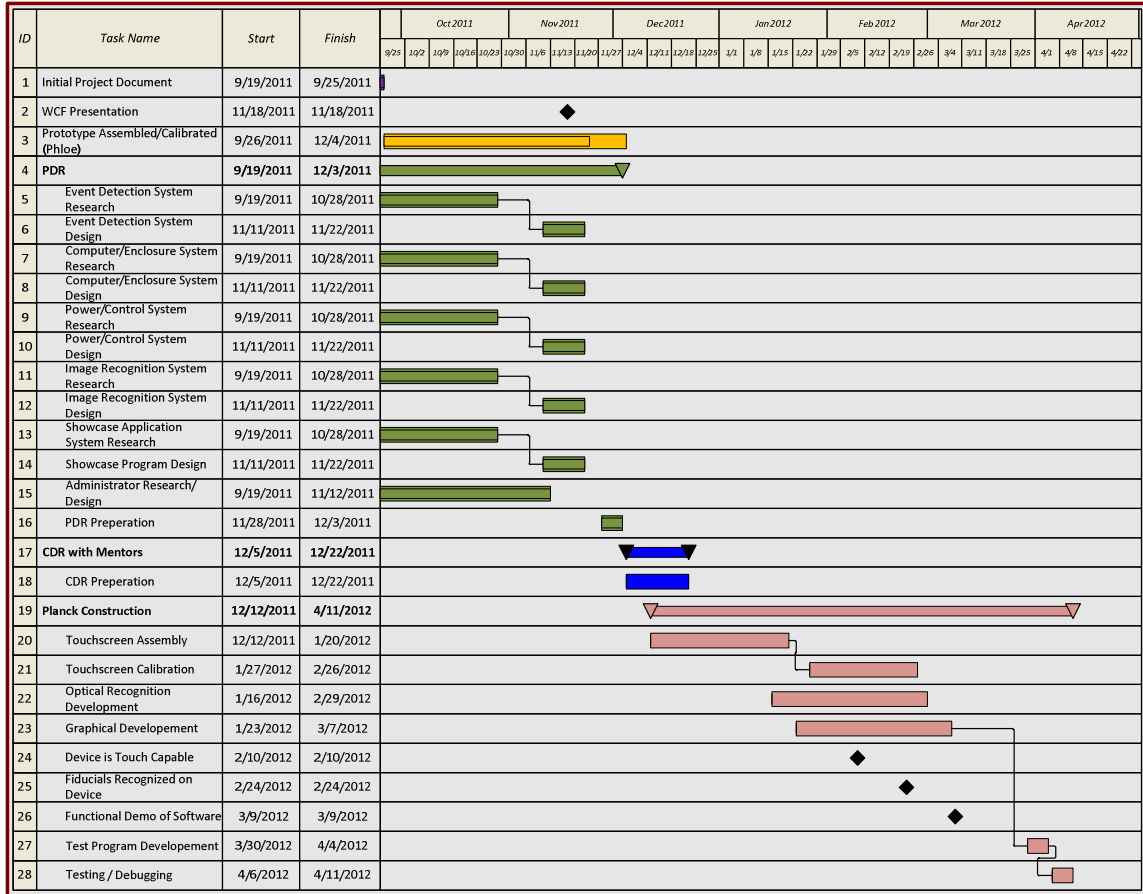


Figure 47 - Gantt Chart

10.4 Software Development Model

The Software Development Model that will be used in project Planck is Scrum. Scrum is an implementation of Agile Methods. The first traces of Agile Methods dates back to 1957. It became popular in the 1990's after heavily-regulated and documented models, such as the infamous Waterfall model, began to face heavy criticism. These heavy-weight models faced opposition after the realization that software is not a product that can be predicted or controlled. One of the key founders of Agile Methods is Martin Fowler. Agile Methods development methodology promotes development over documentation. Face-to-face communication over written documents is promoted, both with the client and the development team. Teamwork and collaboration are heavily stressed as well. Agile Methods also realizes the need for adaptability in software development. Things do not always go according to plan; issues may come up throughout the life cycle and the development model should account for this. Tasks are broken up into smaller increments and the work-output is based upon working software only. The task is composed of the whole life cycle model. Each task should be defined, designed, written, and tested before the next task begins.

Scrum, Extreme Programming (XP), Crystal, and Dynamic Systems Development method (DSDM) are all implementations of Agile Methods. Two members in project

Planck have used Extreme Programming in previous work. However, Scrum was chosen for a few reasons. First, the project Planck team has found that most engineering firms are using Scrum for their software development model. Secondly, two members of project Planck are using Scrum at their co-op workplace. Finally, the most important reason for using Scrum is that it incorporates all of the important aspects of a Software Development Model that project Planck supports. Scrum divides the development staff into teams. Each team is assigned a Sprint for an interval period. A Sprint is a compilation of tasks. The team must stay focused on the Sprint and not deviate from its course. A Sprint can last one week to a month, but typically doesn't last much more than a week. Each team meets daily to go over the daily activities in the current Sprint. A Scrum Master is the representative for each team. The Scrum Master meets with the Scrum Master of another team when team coordination is required. The Scrum Master also allocates resources to fix Spars as they are created. Spars are simply bugs in the system. Spars can be of different severities. As Spars are discovered, the Scrum Master will implement a fix. The Product Owner keeps the development team's plan in alignment with the client's desires. This is done by weekly meetings to assess the new Software from the previous Sprints. The Scrum process can be viewed in Figure X below. In the first stage, the product has been defined and each definition is sorted by priority. From the definition of the product, backlogs of sprints are created to define the tasks that will be sent to the development teams. A Sprint will be given to each Scrum team and the interval to the deadline will range from 24hours to 30 days (typically one week). When the Sprint is completed, the new software will be tested. Spars are created for any defects found in the software. When the Spars are fixed, the team is assigned another Sprint and the cycle continues.

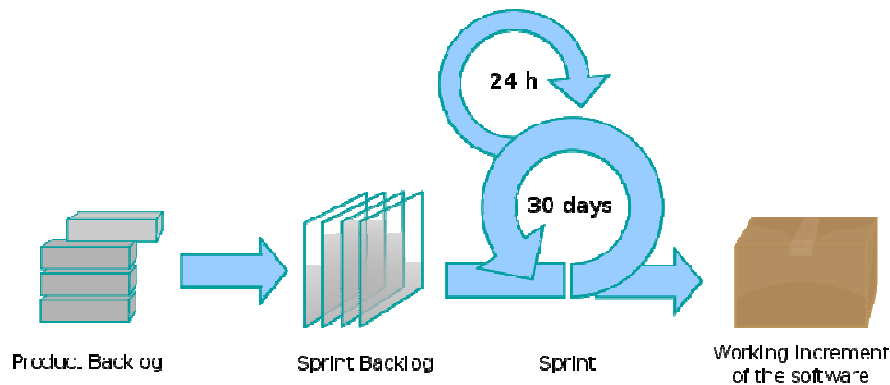


Figure 48 - SCRUM Model

10.5 Software Version Control

Planck is a large system that incorporates two massive pieces of software. There are four members in the group and each has experience in working with software. Since the two software systems, the gesture recognition software system and the showcase application, are so large, they will be split up into separate modules. This will allow all of the group

members to work on the software as a whole. Large-scale software projects are difficult to undertake without some sort of version control management software. To this end, the use of Tortoise version control software is being employed. Enrique Roche has set up a server at their home where the Tortoise project repository will reside. Each member of the group will be able to work on the software and upload their changes to the central server. The Senior Software Engineer will be responsible for maintaining consistency and accuracy in the project repository. He will also be responsible for addressing any errors or quality issues in a group members design. All new versions of the project will be managed by Tortoise.

10.6 Budget

Workforce Central Florida is a government organization which runs the unemployment offices in Orange, Osceola, Seminole, and Sumter counties. They also provide no cost recruitment retention and training programs. Project Plank is being funded by Workforce Central Florida contingent on the team being mentored by professional engineers. WCF is providing five-thousand dollars to each senior design team that meets their funding requirements.

Project Planck has two mentors Ronald Wolff and David Kotick from NAVAIR who are assisting with the administrative and project development aspects of project Plank in order to provide realistic industry input to the project. Below is a table of project Plank's budget. This includes what was budgeted versus the actual cost of the production of the table, Planck. Planck came in barely under-budget. Items that ran over cost include the cameras, electronic PCB', and acrylic. Money was shifted from other allocations to compensate for over expenditures to compensate for the loss.

Table 7

Major Electronics	Actual Cost	Projected Cost
Computer	1,195.54	\$1,150.00
Short Throw Projector	1,379.00	\$1,450.00
Cameras	368.98	\$300.00
Enclosure Materials		
Endlighten Acrylic	750.50	\$525.00
Wood	157.54	\$300.00
Misc(screws, glue, etc)	46.86	\$50.00
Electronics		
LED's, electronics, PCB's	568.92	\$300.00
Development Board		
Acrylic	94.27	\$150.00
Electronics	73.74	\$50.00
Enclosure Related	46.68	\$0.00
Other Costs	56.43	\$500.00
Total	\$4,738.46	\$4,775.00

11. Owner's Manual

The owner's manual details how to transport, setup, calibrate, and run the application weDefend on the multi-user, multi-touch table Planck. Overall, the process is fairly simple and can be mastered in only a few tries. It is recommended to not proceed with any of these tests without the assistance of at least one other person. Ideally 2-3 people are recommended. The owner's manual is split up into two sections, hardware and software. First hardware setup will be discussed, which entails transportation and setup of Planck.

11.1 Parts List

There are 9 separate components that make up Planck. These components are as follows:

1. Box

2. Legs
3. Cotter Pins
4. Mar-resistant Acrylic
5. Rear projection acrylic
6. Endlighten XXL acrylic
7. Projector controller
8. Keyboard and mouse
9. IR filters

11.2 Transportation of Planck

The multi-touch device, Planck, can be divided into smaller sections to allow for easy travel. The legs detach from the box, and the acrylic can be lifted off the box to be transported separately. The legs, box, and acrylic can all be transported separately and all are capable of fitting through a standard doorway. The only complication is that the box needs to be tilted on its side. It is recommended that the side with all of the hardware components (computer and projector) travel on the side closest to the ground. The following sections detail the exact procedures for the transportation and setup of the multi-touch device Planck.

11.3 Hardware Setup of Planck

Once all components of Planck have reached the final destination, setup can begin. A large floor footprint is needed to accommodate Planck. This should be at least a 6x6 ft area. The first step is to place the legs of Planck in the middle of the areas set aside for Planck. The pegs are keyed and should be facing up. The pegs slide into four holes on the bottom of the box. The smaller peg is designed to fit inside a hidden hole underneath the motherboard. See Figure 49 which illustrates the pegs located on the legs.

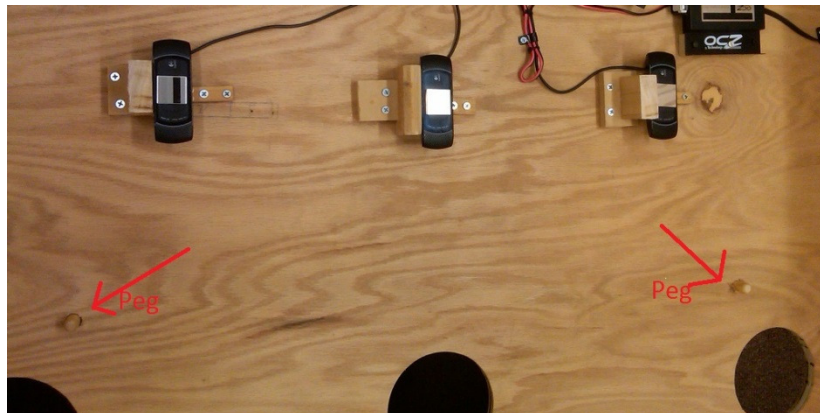


Figure 49 - Positioning of Pegs

With the legs positioned accordingly, the box can be lifted and placed on-top of the legs. At least two people are required for this task. The pegs of the legs slide easily into the holes of the box. Care should be practiced to prevent the pins from sliding into a fan. Once the table is secure with the legs attached, the three cotter pins slide into very small holes in pegs. This should be done inside the box, as seen in Figure 50.

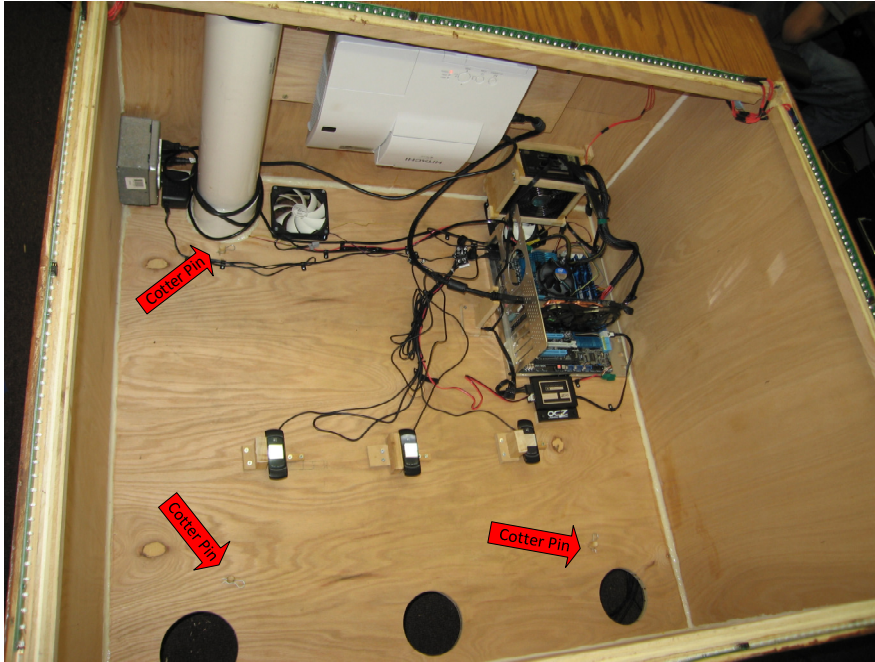


Figure 50 - Cotter Pin holes located on leg pegs

With the table securely mounted on the legs and fastened with the cotter pins, the next step is to check cable security. Every cable inside the box of Planck should be checked for snugness before the startup of the system. First, check all cables attached to the LED pcb's. These are the green boards that encompass the upper perimeter of the box. All cables should be snug inside the black connector. This is illustrated in Figure 51.

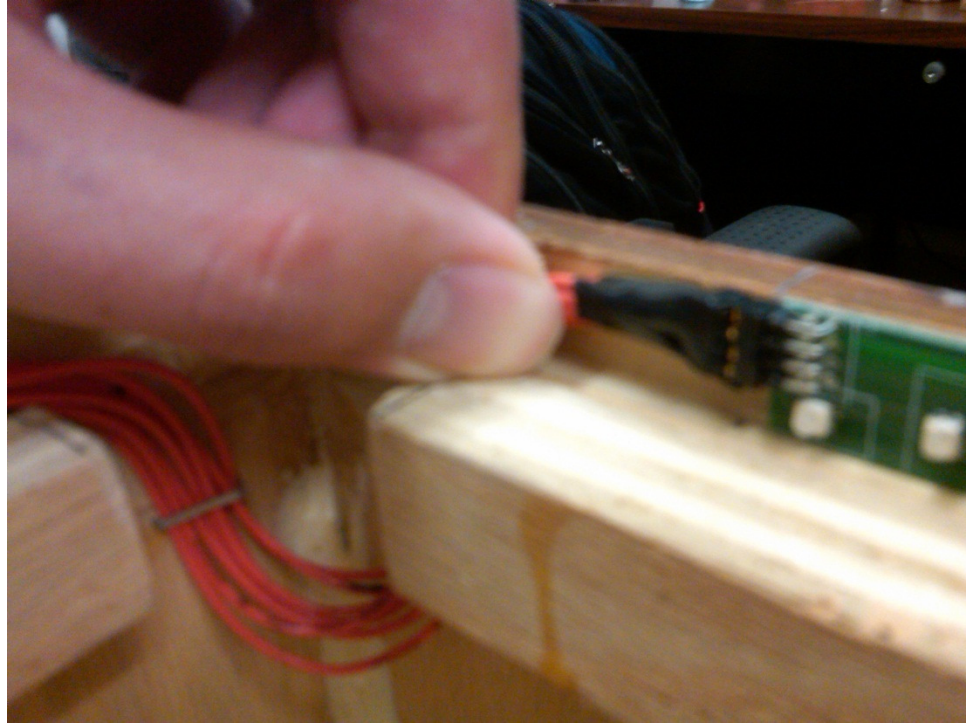


Figure 51 - Checking LED PCB Snugness

Second, confirm all other cables are snug. This includes all power plugs that attach to the power module located inside the box. This includes the USB cables that attach to the cameras and USB hub. Finally, confirm that the HDMI cable that attach to both the projector and video card are snug. With all cables confirmed, the cameras should be checked for proper positioning.

There are three cameras mounted inside the box of Planck. All three cameras are used to capture touches. It's possible that the cameras were tilted incorrectly during Planck in travel mode. To aid the user in this process, block stops were mounted. These block stops confirm the correct position and tilt of each camera. Carefully, tilt each camera towards the block stop. Stop moving the camera when the camera can move no further. This is the correct position for the camera. After all three cameras are positioned, the IR filters can be placed on-top of each camera. These filters resemble that of a mirror. These should be placed directly over the lens of the camera. This can be viewed in Figure 52.



Figure 52 - Placement of IR Filters over camera

11.3.1 Cleaning of Acrylic

At this point, the table should be mounted on the legs and secured with cotter pins. All cables inside the box should be secure. Finally, the cameras should be positioned correctly and the IR filters placed over each camera. It is assumed that during the transportation of the acrylic that fingerprints and dirt has reached each layer of acrylic. Cleaning the acrylic before installation is a crucial step. Acrylic is a very soft material. Foreign particles left on the acrylic can lead to permanent scratching. Fingerprints and other smudges can lead to a distortion of the image the camera to capture for touch recognition. Ideally, only a clean soft cotton towel and water should be used to clean the acrylic. This process can be long and slow. It is recommended to follow rigorous practices to prevent dirt and smudges from ever reaching the acrylic. The acrylic should be dried with another clean, soft, dry cotton towel.

11.3.2 Mounting of the Acrylic

After the acrylic has been cleaned, it's ready to be mounted to the box. It is recommended to have at least two people for this job. With only touching the corners, two people should pick up the Endlighten XXL acrylic. This is the thickest of the three pieces. There is a black dot on both the table and the acrylic. The black dot on the acrylic should go in the corner with the black dot on the table. This is illustrated in Figure 53. There is a designed for the acrylic to sit on in the table, it is a snug fit.

Carefully place one side into the slot, and then the next. Care should be taken place not to move wiring or scratch the acrylic. After the Endlighten has been placed, the rear projection acrylic can be placed directly over the top of the Endlighten. There is no specific orientation for this. The rear projection acrylic is grey. Finally, the mar-resistant acrylic should be placed on top. There are four acrylic rods attached to the acrylic. These four rods secure into four holes in each corner of the table. The acrylic has been painted black to hide hardware appearances. The thicker black band overlays the thicker wooden band as seen on-top of the box.

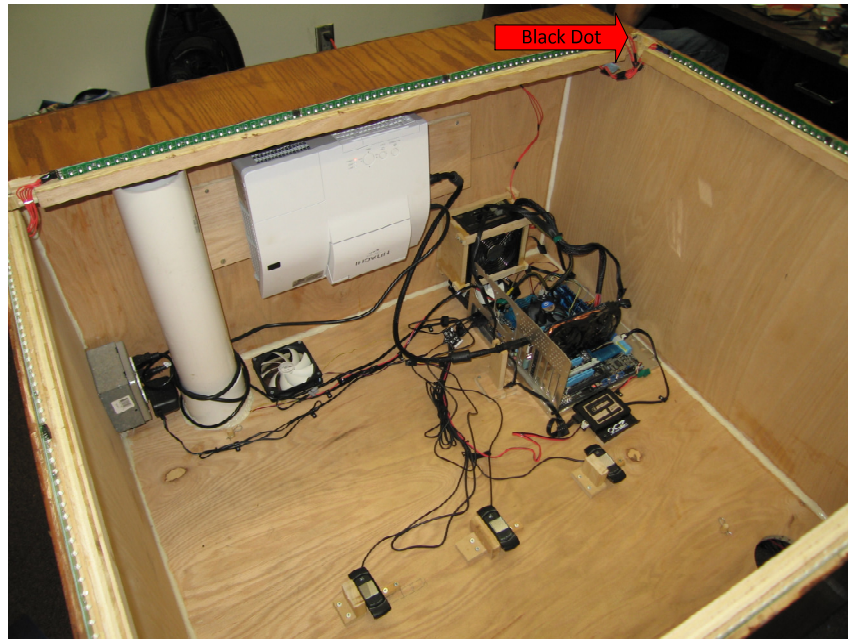


Figure 53 - Location of Black Dot

This concludes the installation and travel process for the multi-touch table Planck. This process can be reversed for tear down instructions. While the box is fully capable of being travel ready, it is recommended to reduce frequency of travel due to calibration and the time involved in tear-down and setup. Please contact the manufacturer if any issues arise. The next section will demonstrate how to start Planck once the table is fully setup.

11.4 Powering-up Planck

The starting of Planck is a fairly simple chore. There is a USB hub and a power switch located underneath the table near the leg by the computer. This is can be seen in Figure 54. The USB hub is provided to connect input devices such as a keyboard or mouse. A USB flash drive can also be connected to transfer files. First, flip the power switch to the right, and then back to the middle. It is important not to leave the switch flipped or the computer will immediately shutoff. The computer and cameras will receive power and startup at this point.



Figure 54 - USB Hub and Power Switch

The next step is to use the provided projector remote to turn on the projector. Pressing the power button once will accomplish this. It will take a few minutes for the bulb to warm up. By this point, Windows will be waiting to be logged in. There is no password needed to enter the home screen. Once all applications have loaded, start CCV software. Next, start Microsoft Visual Studios and load the latest version of weDefend. From there, a build without debugging is selected. The application, weDefend, will boot and will be ready for use with multi-touch and object recognition capabilities. This concludes the hardware setup for the multi-touch table Planck. The next sections to follow are that of software calibration and game-play.

11.5 CCV

The settings of CCV are saved in .xml in the following directory: “C:\Program Files (x86)\NUI Group\Community Core Vision\data\xml”. They are also backed up on the desktop in a folder called “xml”. These .xml files stipulate the resolution of the cameras, various settings in CCV that work with our table, and the calibration required in order to stitch all three cameras together into one cohesive image. If they ever become corrupted or a change is made in CCV that needs to be corrected, overwriting these files in the installation directory listed above will solve the problem.

The camera settings used by CCV are not saved whenever Planck is rebooted. These settings must be set every time CCV is booted. In order to adjust the camera settings open CCV (shortcut located on the desktop) and click the “setup/track” button on the left control panel column as seen in the figure below. Alternatively, pressing TAB on the keyboard achieves the same function.

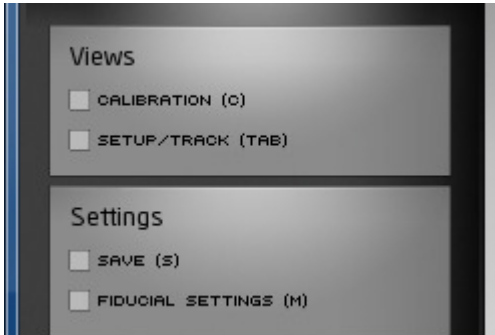


Figure 55 - Calibration and Setup tabs

Once in the setup screen you will see the three cameras that are being used by CCV. From left to right these are cameras 1, 2, and 3. Left clicking in on one of the camera windows shows two options, “config” and “clear”. An image of this operation is shown below for each camera.

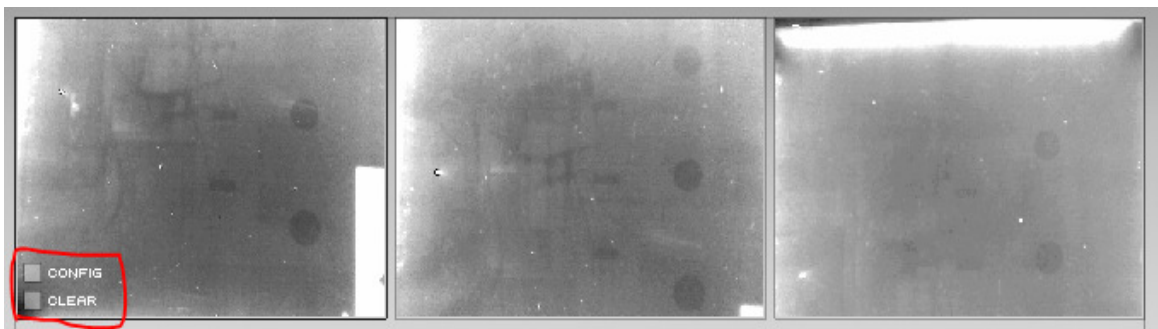


Figure 56 - Location of config for Camera 1



Figure 57 - Location of config for Camera 2

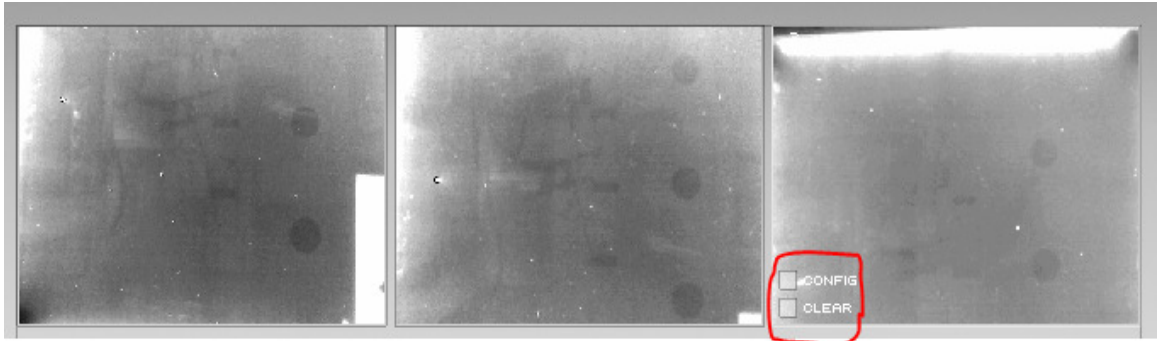


Figure 58 - Location of config for Camera 3

Each camera must be configured individually by clicking “config” in the respective camera that you wish to configure. When you press config a window with all the camera settings is shown. It is very important to disable any automatic settings the camera has before applying settings. Unchecking the ‘auto’ checkbox disables these automatic settings and allows manual manipulation of the camera settings. Copy the settings in the two screen shots below into the two areas of the camera configuration:

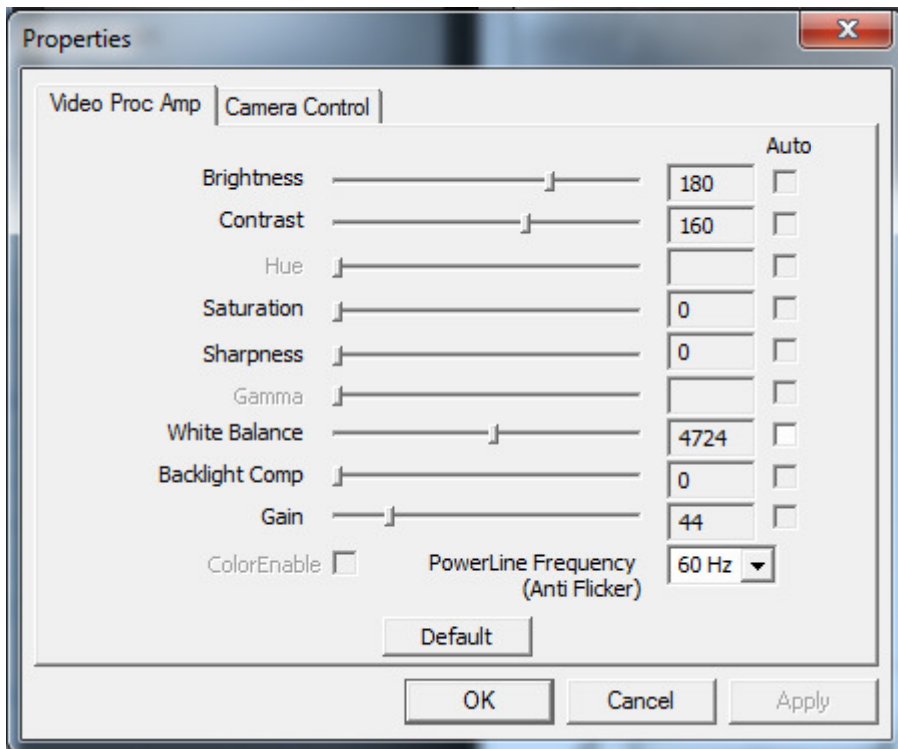
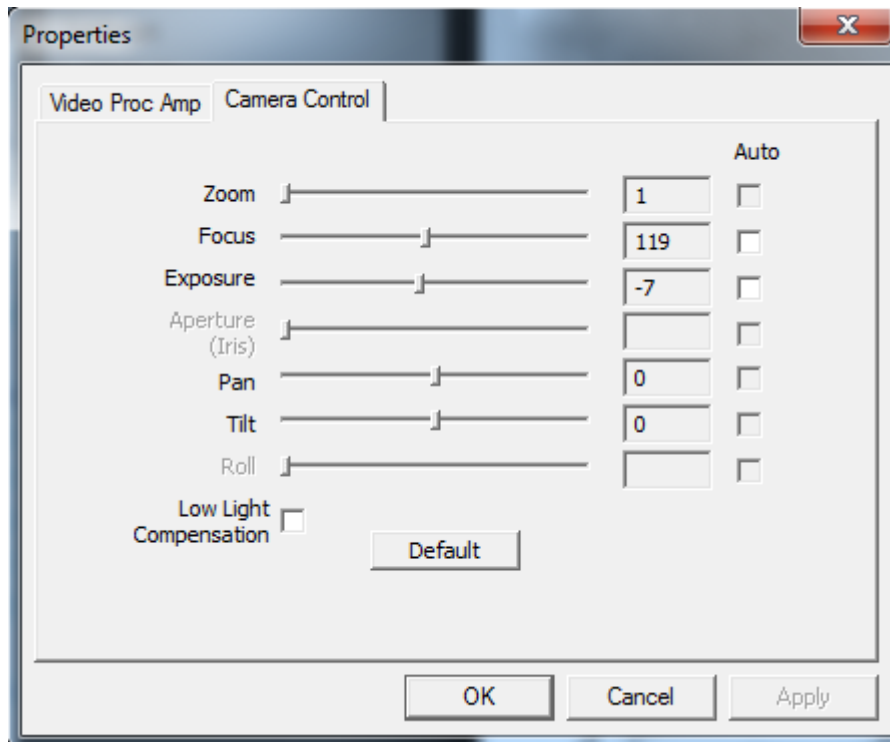


Figure 59 - Example Calibration Settings (Part A)



These settings must be applied to each camera separately by moving the slider bars.

Once all the settings on the cameras have been adjusted, go back to the beginning screen by hitting TAN or clicking the setup/track button on the left hand control column again. Once at the home screen the images captured by the cameras should look like the Figure 61.

Fine-tuning of the settings based on external lighting conditions may be necessary to have images such as what is shown above. Figure 60 shows of what the fiducial settings look like. In order to see the fiducial settings, the "fiducial settings" button must be hit in the left hand control column

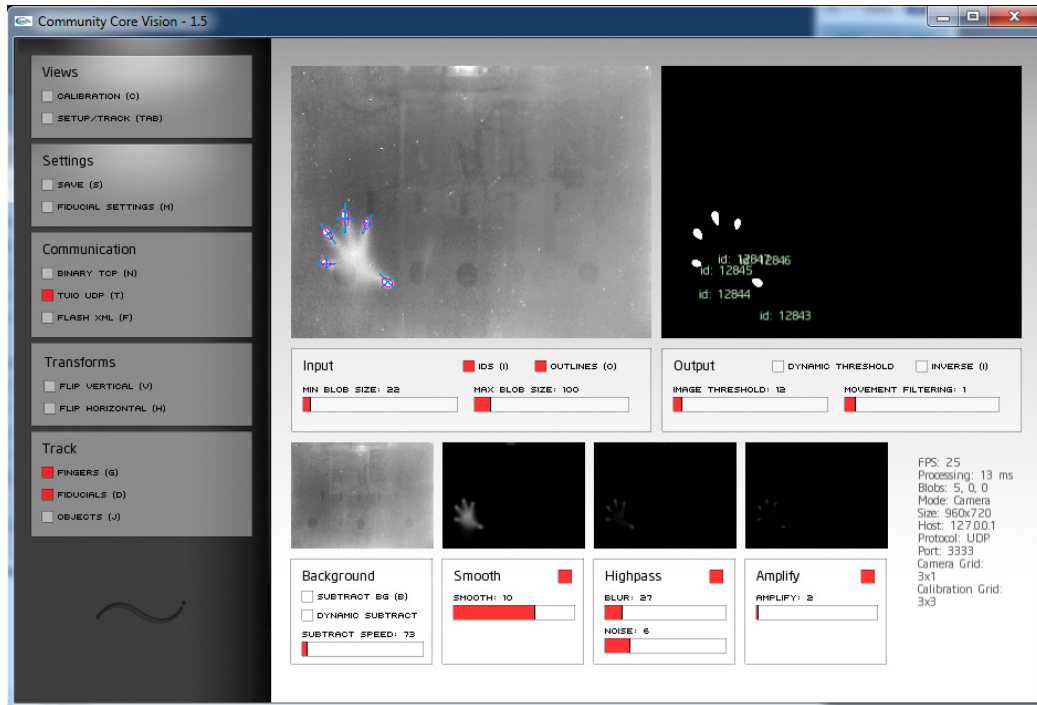
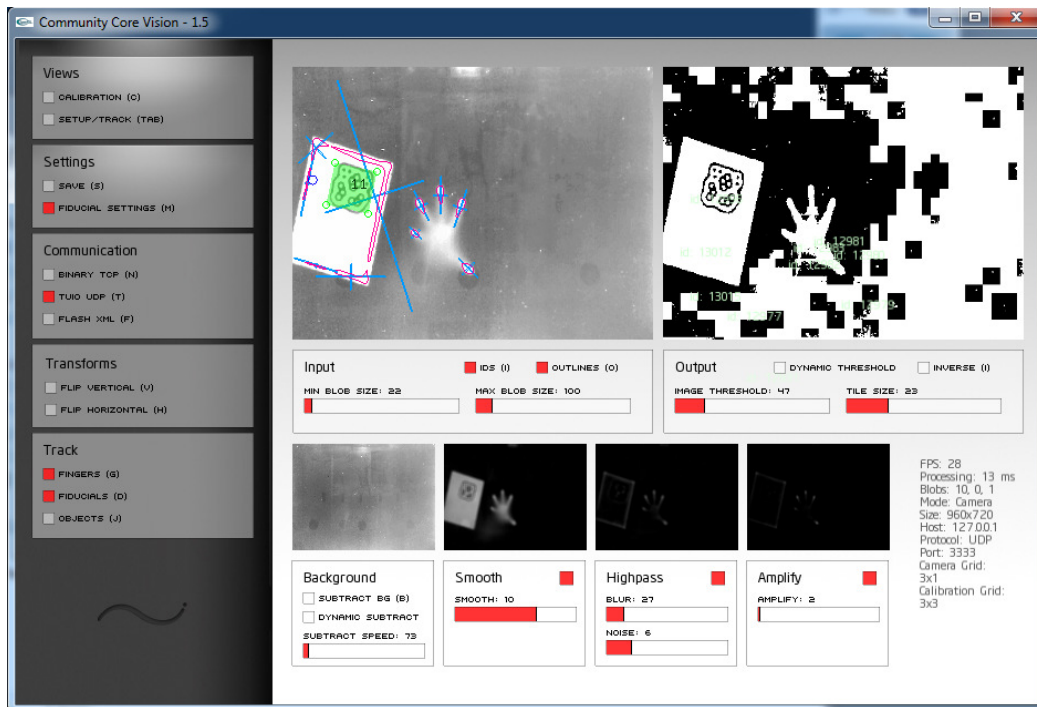


Figure 61 - Touch input settings screenshot



For more information on specific settings within CCV, consult their website and forums located at <http://ccv.nuigroup.com/>. There are many tutorials, screenshots, and a discussion forum for topics related to CCV and its settings with different IR technologies.

11.6 weDefend Simulation

The weDefend simulation is comprised of a Microsoft Visual Studio solution project. Below are instruction for running the file and using the software.

11.6.1 Running weDefend

To run weDefend first start up CCV and configure it as discussed in the instructions above. Once it is configured and running, minimize it and double-click the “weDefend Simulator” shortcut icon located on the desktop. The shortcut points to a Visual Studio .sln file that will open the source code of the project. Once visual studio is up and running hit the “F5” key on the keyboard to begin execution of the program. This will open up the title window of the simulator as shown below. WeDefend is now running.



weDefend

- Insert Fiducial -

Figure 63 - Screenshot of weDefend start screen

11.6.1 Using weDefend

There are four main modes/states of operation in weDefend:

1. Opening Welcome Screen
2. Preparation Mode
3. Action Mode
4. Game Over

Changing between modes is primarily handled by using three fiducials to switch states. Each fiducial is uniquely numbered with an ID number. The fiducials are shown in the Figure 64.

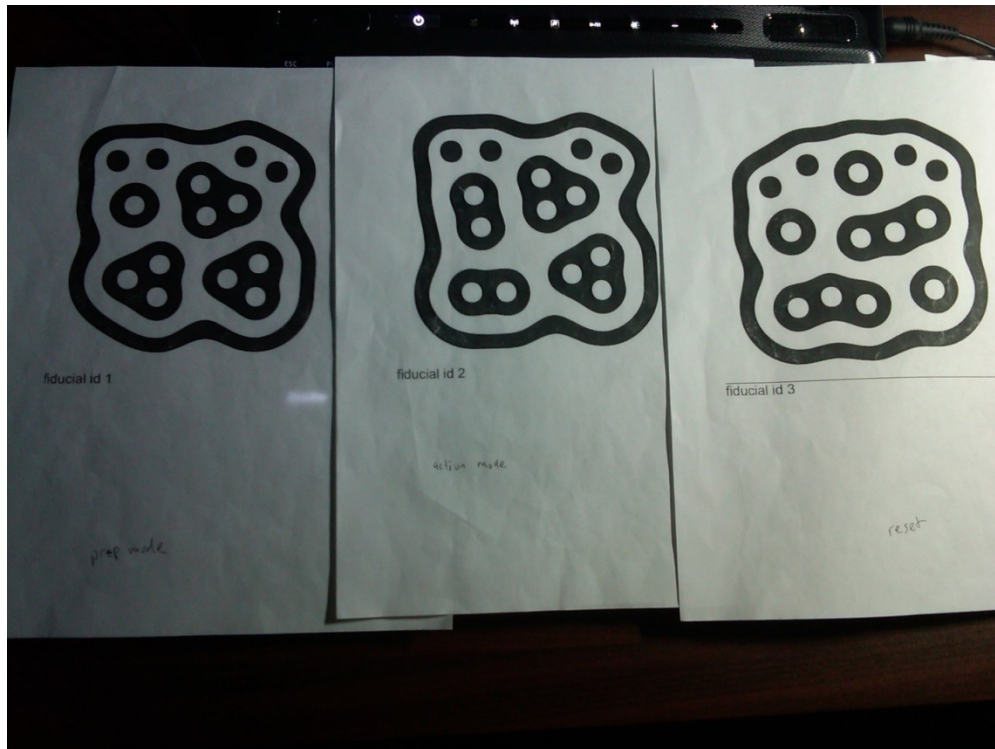


Figure 64 - Fiducials for switching states. 1 for Prep mode, 2 for Action mode, 3 for Reset.

Fiducial 1 will switch from the opening welcome screen to prep mode. Fiducial 2 will switch from prep mode to action mode. When the simulation is lost (too many insurgents in the defended area) the Game Over screen is displayed. To reset the game fiducial 3 must be placed on the screen. Fiducial 3 resets the game from any mode/state and takes the user back to the opening welcome screen.

Prep Mode

By placing fiducial 1 on the opening welcome screen the user is taken to prep mode as shown in the screenshot below.

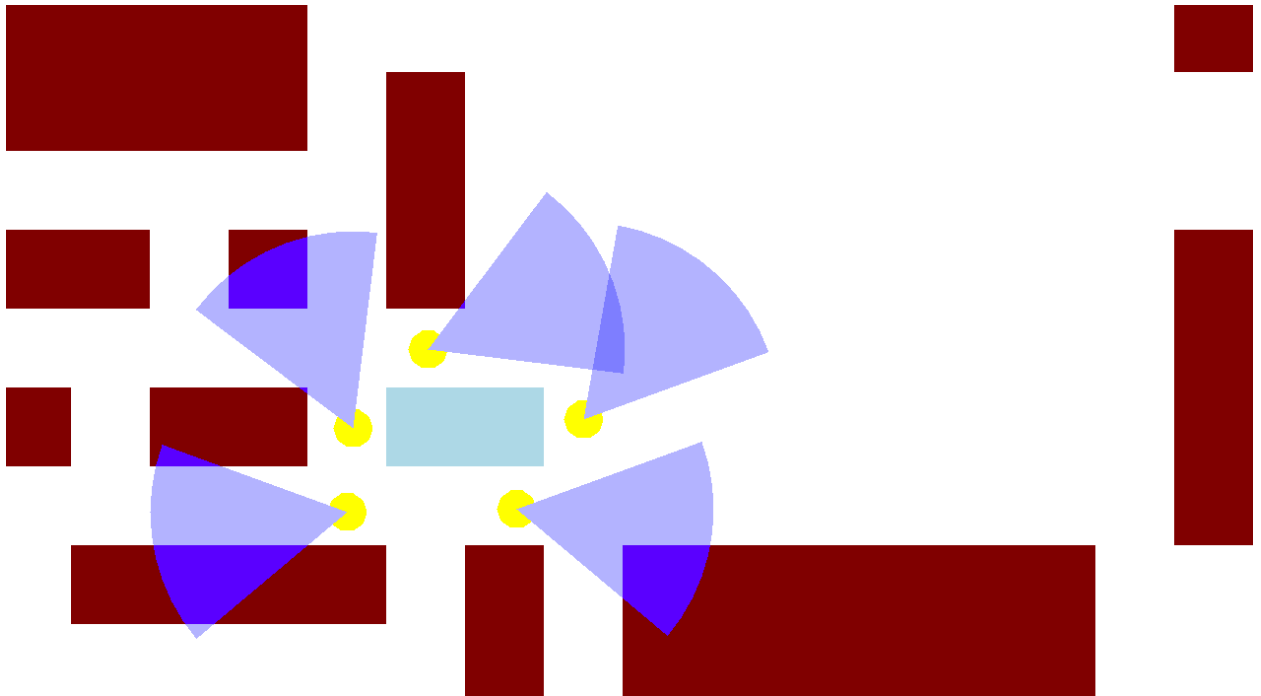


Figure 65 - Screenshot of Prep Mode

In prep mode the user is faced with a top down view of the map they will be running the simulation in. Each one of the yellow dots shown above is a soldier and each has an associated field of view. The field of view corresponds to the area a soldier can see within. If an enemy comes inside the blue area field of view area, the soldier fires at the enemy.

If a touch is placed on any of the soldiers, the soldier will turn green to indicate it has been selected. This is shown below. Selecting a soldier allows the user to do one of three things: 1. Move the soldier, 2. Change the soldiers field of view size, 3. Create a patrol route for the soldier.

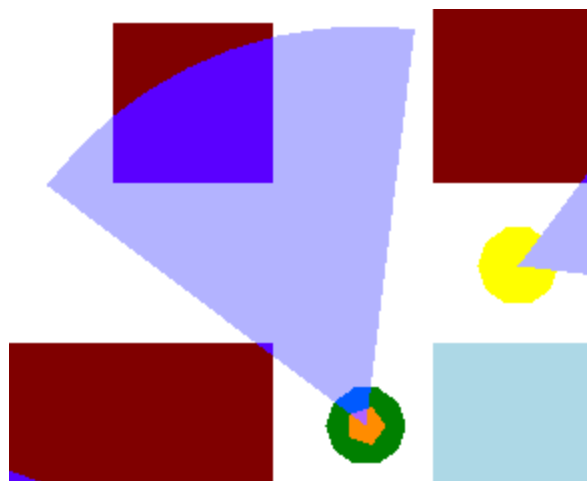


Figure 66 - Example of soldier being selected

To move a soldier, select it with a finger. The soldier turns green to confirm that it has been selected. Move your finger on the surface, dragging the soldier around.

To modify the field of view of a soldier select it with one finger. The soldier turns green to confirm that it has been selected. Place another finger in any area of the field of view and drag it to the selected radius away from the soldier. Rotating the finger that is holding the field of view around the soldier shifts the location of the field of view around the soldier. To deselect the field of view and apply those changes to the FOV, lift the finger on the field of view. An illustration of these actions is shown below.

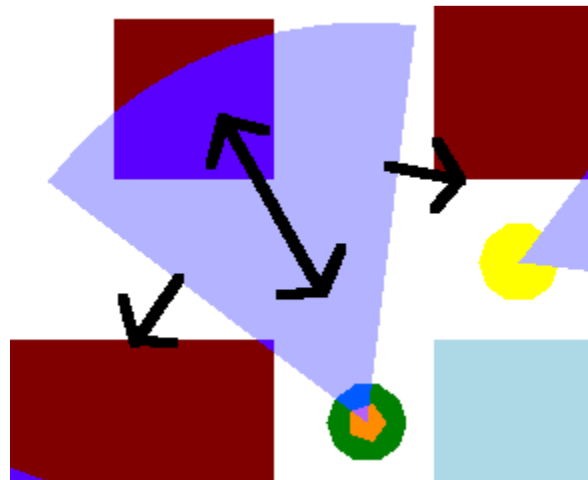


Figure 67 - Field of View Movement

\

To create a patrol route for a soldier, select the soldier with one finger. The soldier turns green to confirm that it has been selected. Use a second finger to select the orange puck located at the center of the soldier. Drag this puck around the area you would like the soldier to patrol, image below.



Figure 68 - Example of patrol route puck being moved

When finished drawing the route, let go of the puck. It will snap back to the soldier and the soldier will begin on the route just created.

Another control allowed in prep mode is the lasso object. The lasso object allows for a selection area to be created in the map to group several units together. A lasso object can

be begun by placing your finger on an area of the map that has no objects in it, then dragging this finger over the soldiers you would like to group together. A screenshot of this is shown below.

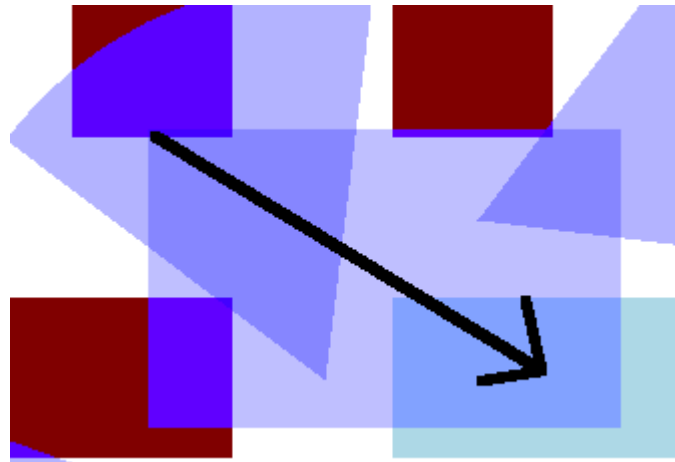


Figure 69 - Example of Lasso being drawn

With the soldiers you would like grouped under the lasso, remove your finger. The soldiers under the lasso should all get their colors changed, indicating that they now belong to the same group. A screenshot of this is shown below.

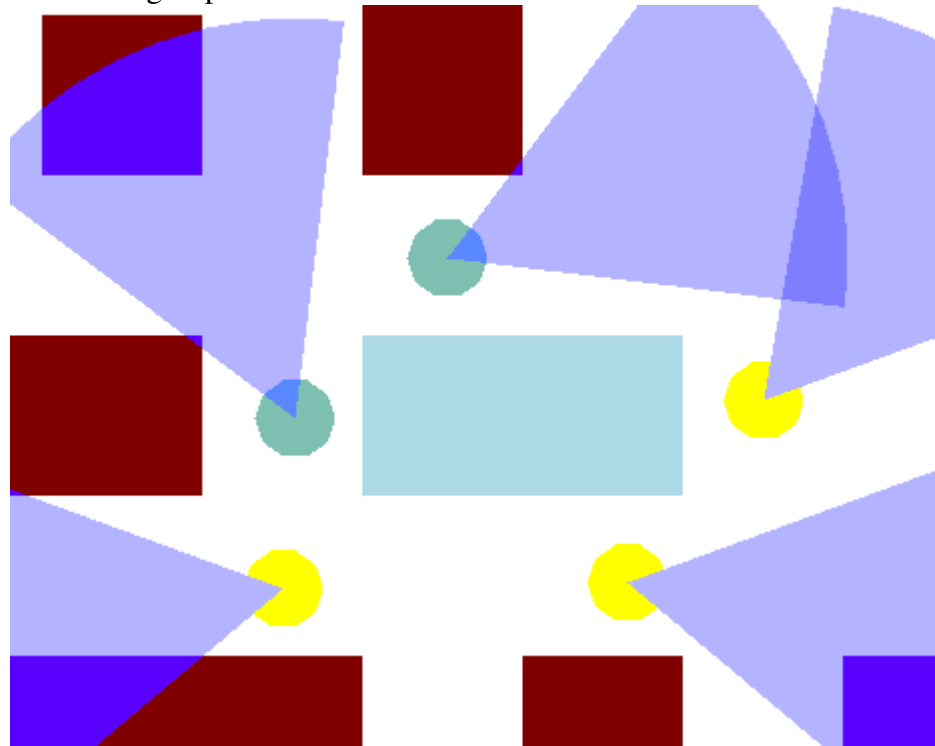


Figure 70 - Example of Units being Grouped

Action Mode

To access action mode, place fiducial number 2 on the surface. In action mode insurgents/enemies will begin to fill the screen from all four corners. There's a screenshot below of enemies coming into the map and soldiers firing at them.

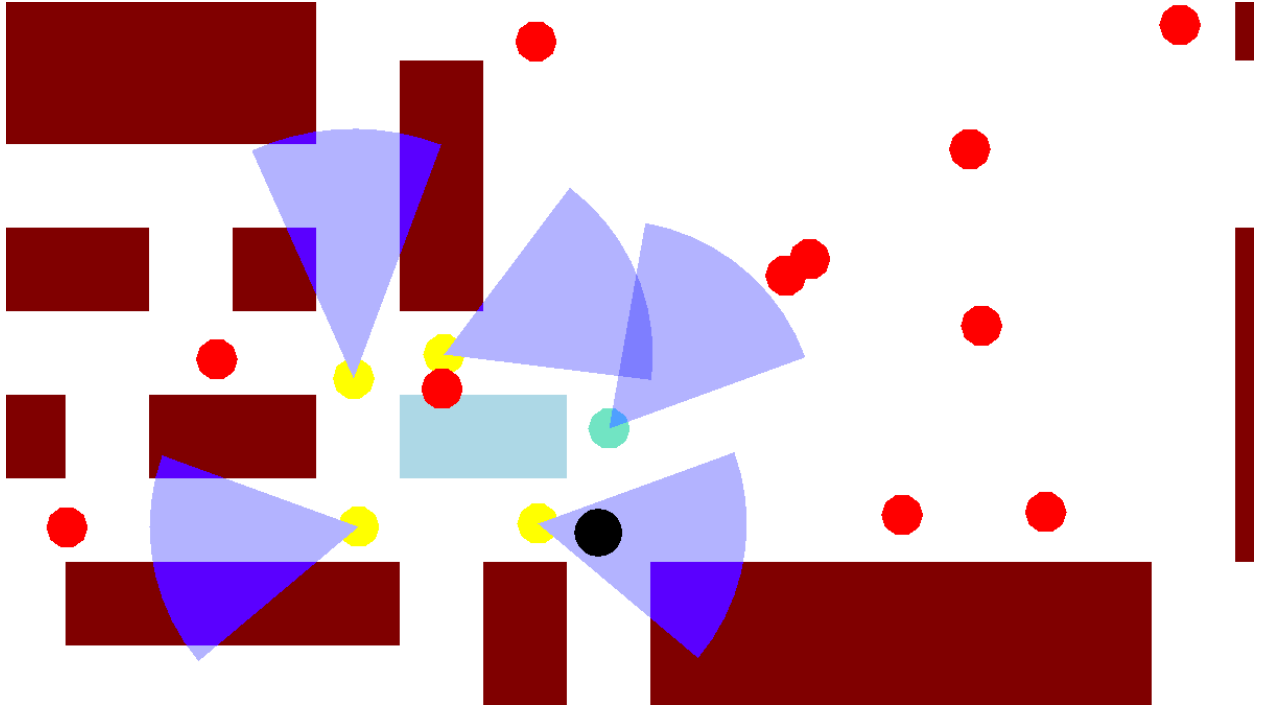


Figure 71 - Screenshot of Action mode

When 8 enemies reach the area being protected on-screen (the blue square in the center), the simulator will automatically switch to the Game Over state, as can be seen below.

GAME OVER

Figure 72 - Game over screen

Fiducial 3 can then be used reset the game.

Debug Mode

Three debug modes have been built into the game: 1. Debug Boxes, 2. Finger debug, 3. Full debug. These modes can be enabled by placing certain fiducials on the surface:

4. Debug Boxes
5. Finger Debug
6. Full Debug

An image of the fiducials used to control the debugging options is shown below.

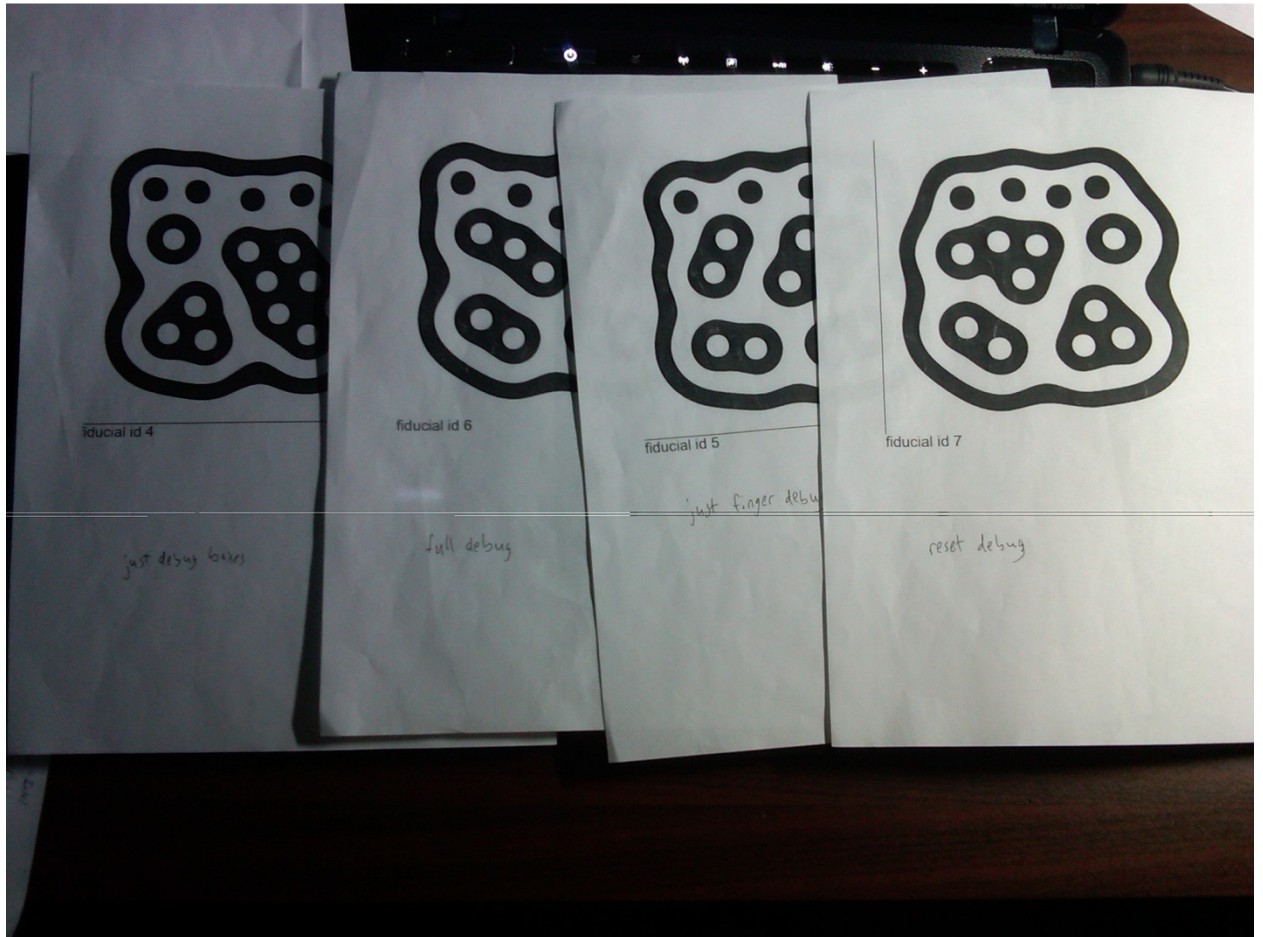


Figure 73 - Fiducials used for Debugging

When fiducial 4 is placed on Planck, Debug Boxes is enabled. Debug Boxes gives information about all objects that can be interacted with on-screen. A screenshot of it is shown in Figure 74. As you can see, one soldier has been selected and their debug information contains a parameter that states where the original of its touch is located. In addition to this information, the debug boxes also contain the coordinates of the soldier, the angle its FOV, the FOV position, the FOV's width, and if the FOV has also been selected.

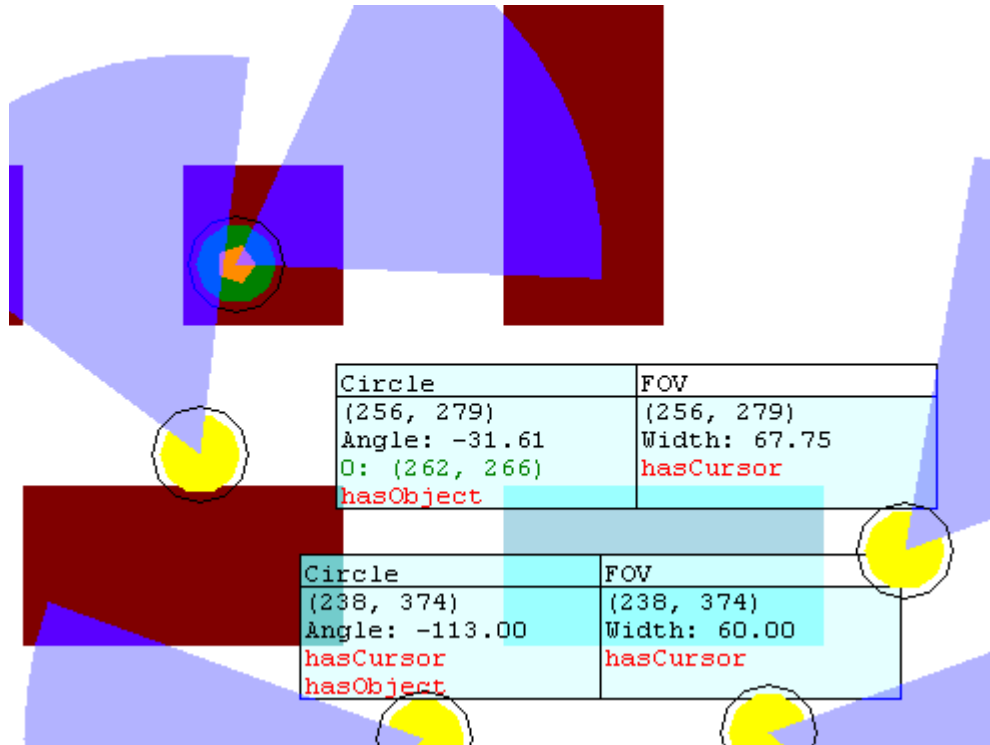


Figure 74 - Debug information being shown

Fiducial 5 begins Finger Debugging. The Finger Debug is option is designed to show the user where touches are being recognized. A small diamond shape is drawn beneath each touch indicating where the touch is being recognized on the surface. This allows for more exact selections of objects and is especially useful when trying to use the patrol puck feature of the game. Fiducial 6 begins full debugging. This option turns on the two debugging options listed above: Debug Boxes and Finger Debugging. A screenshot of this option is shown in Figure 75.

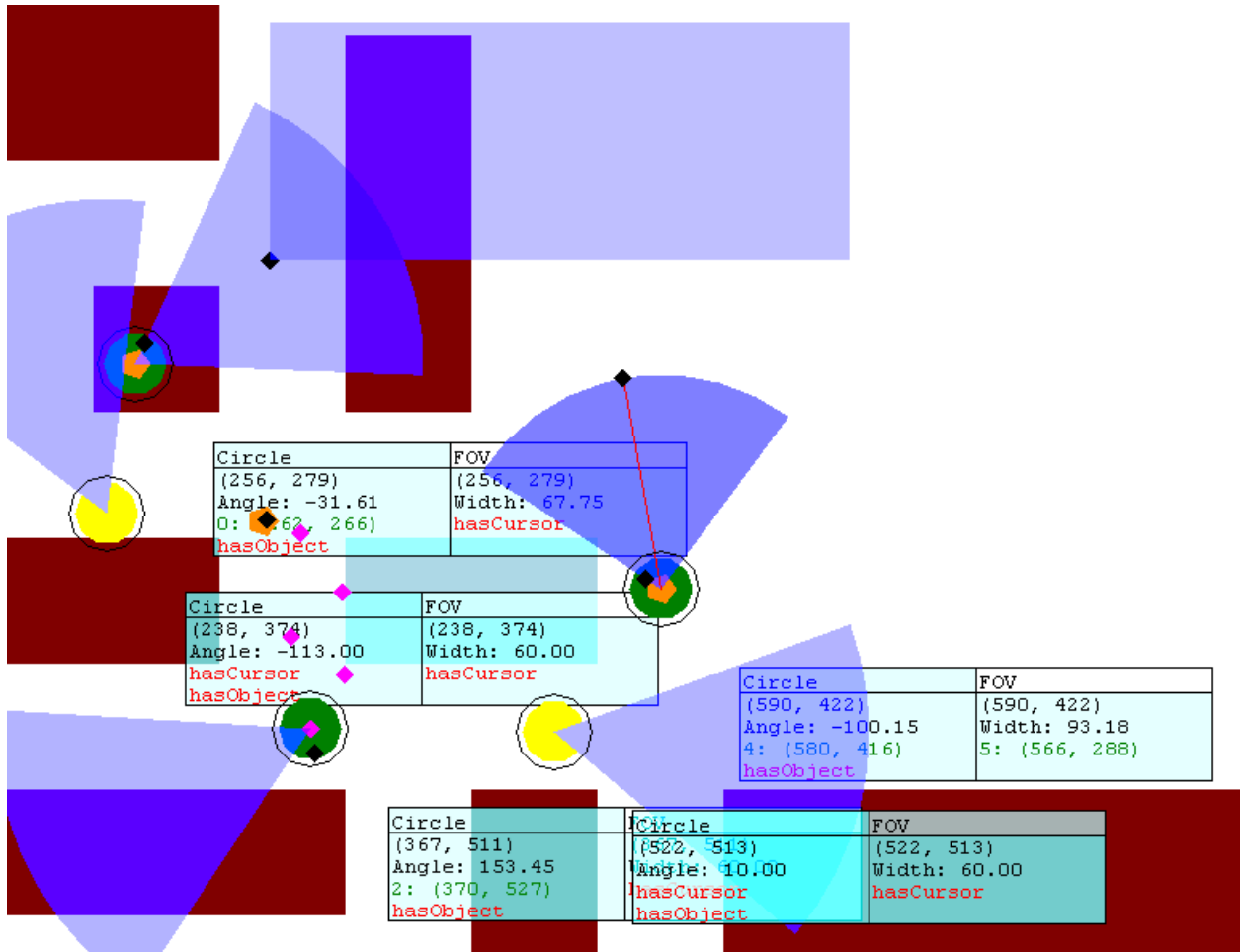


Figure 75 - Full debug information being shown

Works Cited

- 10 things you need to know about 1080p/50. (n.d.).
Understanding SAR ADCs: Their Architecture and Comparison with Other ADCs. (2001, 10 02). Retrieved from Maxim: <http://www.maxim-ic.com/app-notes/index.mvp/id/1080>
- 4-Wire PWM Controlled fans Specification. (2005-2009).
- Answering the "Where is the proof That Agile Methods Work" Question".* (2007, 01 19). Retrieved 11 2, 2011, from Agilemodeling.com: Agilemodeling.com
- ATSC Standard: Video System Characteristics of AVC in the ATSC Digital Television System. (2008-07-29).
- 720p. (2010, 10 08). Retrieved 10 22, 2011, from CNET.com: http://reviews.cnet.com/4520-6029_7-6301006-1.html
- Windows 7 system requirements.* (2011). Retrieved 11 2, 2011, from Microsoft.com: <http://windows.microsoft.com/en-US/windows7/products/system-requirements>
- 55" Class LED 8000 Series Smart TV.* (n.d.). Retrieved 10 23, 2011, from Saumsung.com: <http://www.samsung.com/us/video/tvs/UN55D8000YFXZA-features>
- Ambler, S. (2010). Survey Says: Agile Works in Practice. *Dr. Dobb's.*
- Ambler, S. W. (2005). *The Elements of UML 2.0 Style.* Cambridge, New York: Cambridge University Press .
- Analog Devices. (2010). Low Voltage Temperature Sensors.
- Baldwin, T. (n.d.). *Combating Keystoning - Equation Derivation.* Retrieved 2011, from http://freespace.virgin.net/tom.baldwin/keyst_deriv.html
- Bradshaw, B. (2010, 10 08). B.V. Technology. *HDTV: What is 1080p?*
- Burke, M. (2004, 02). *Why and How to Control Fan Speed for Cooling Electronic Equipment.* Retrieved from Analog Dialogue: http://www.analog.com/library/analogDialogue/archives/38-02/fan_speed.html
- Cha, B.-R. P.-Y. (n.d.). <http://www.jstage.jst.go.jp/article/elex/7/1/40/> pdv. In *Thermal consideration in LED array design for LCD backlight unit applications* (pp. pp. 40-46). IEICE Electron Express.
- Costanza, E. (2011). *Home Page.* Retrieved 2011, from d-touch.org: <http://d-touch.org/>
- Costanza, E., & Huang, J. (2009). Designing Visual Markers. *CHI.*
- Douxchamps, D. (2011, 08 11). *The IEEE1394 Digital Camera List.* Retrieved from <http://damien.douxchamps.net/ieee1394/cameras/>
- Drewry, T. (n.d.). *Multi-Touch Gaming.* Retrieved 11 4, 2011, from multitouchgaming.blogspot.com: <http://multitouchgaming.blogspot.com/>
- Edwards, L. (2010, 01 12). *New multi-touch screen technology developed.* Retrieved from Physorg: <http://www.physorg.com/news182502758.html>
- Erickson, B. J., & Jack Jr, C. R. (n.d.). *Correlation of single photon emission CT with MR image data using fiducial markers: Abstract.* Retrieved 2011, from American Journal of Neuroradiology: <http://www.ajnr.org/content/14/3/713.abstract>
- Grassle, P., Baumann, H., & Baumann, P. (2005). *UML 2.0 in Action.* Birmingham, UK: Packt Publishing.
- Grootjans, R. (2003-2011). *Riemer's XNA Tutorials.* Retrieved December 4, 2011, from file:///C:/Users/Chris/Dropbox/Senior%20Design/Systems/Showcase%20Applicat

- ion%20System/References/Web%20Resources/Riemers%20XNA%20Tutorials.htm
- Hickey, E. (January 2005). A look back at the NC200.
- Kaltenbrunner, M. (n.d.). *Main Page*. Retrieved 2011, from TUIO.org.
- LED vs. LCD TV Comparison*. (n.d.). Retrieved 10 23, 2011, from LED TELE: <http://www.ledtele.co.uk/ledvslcd.html>
- Macedonia, M. (2002). Games, Simulation, and the Military Education Dilemma. *Forum for the Future of Higher Education*.
- Martin, K. (n.d.). *Reactivation User Forum*. Retrieved 2011, from Sourceforge: <http://sourceforge.net/apps/phpbb/reactivision/viewtopic.php?f=2&t=134&p=449&hilit=CPU#p449>
- Microsoft. (2011). *XNA Game Studio 4.0 Refresh*. Retrieved Decemeber 4, 2011, from file:///C:/Users/Chris/Dropbox/Senior%20Design/Systems/Showcase%20Application%20System/References/Web%20Resources/MSDN%20XNA%20resource.htm
- Natural User Interface Group ~ X1. (n.d.). *Getting Started with CCV*. Retrieved 2011, from NUIGroup Wiki: http://wiki.nuigroup.com/Getting_Started_with_CCV
- NUI Group. (n.d.). *CCV - About*. Retrieved 2011, from CCV: <http://ccv.nuigroup.com/>
- NUI Group Community. (n.d.). *NUI group about*. Retrieved 2011, from NUIgroup.com: <http://nuigroup.com/go/lite/about/>
- NUIGroup. (n.d.). *NUI Group Forum*. Retrieved 2011, from NUI Group: <http://nuigroup.com/forums/>
- Powell, E. (2009, 7 28). *Projector Central*. Retrieved 12 4, 2011, from projectorcentral.com: http://www.projectorcentral.com/lcd_dlp_comparison.htm
- projectorcentral authors. (n.d.). *Project Central*. Retrieved 11 1, 2011, from projectorcentral.com: <http://www.projectorcentral.com/>
- Ramseyer, N. (n.d.). *Diffusers and Projection Screens*. Retrieved from Peau Productions: <http://peauproductions.com/diffusers.html>
- Salmon, T. O. (n.d.). *Lecture 8 - Lambertian Surfaces, Trolands*. Retrieved from http://arapaho.nsuok.edu/~salmonto/vs2_lectures/Lecture8.pdf
- Schulmeister, K. (n.d.). *Wavelength Considerations*. Retrieved 10 28, 2007, from http://info.tuwien.ac.at/iflt/safety/section1/1_1_1.htm
- Screen gain*. (n.d.). Retrieved from Dnp: <http://www.dnp-screens.com/DNP08/Technology/Basic-Visual/Screens/Screen-gain.aspx>
- Segal, M., & Akeley, K. (August 8, 2011). *The OpenGL Graphics System: A specification*. The Khronos Group Inc.
- Sutcliffe, G. C. (2011). *Microcontroller Interfacing*. Retrieved from http://www.w9xt.com/page_microdesign_toc.html
- Texas Instruments. (2011, 07). MIXED SIGNAL MICROCONTROLLER.
- Tomshardware Authors. (n.d.). *Tom's Hardware The Authority on Tech*. Retrieved 12 1, 2011, from Tomshardware.com: <http://www.tomshardware.com/reviews/Components,1/Cooling,7/>
- Varcholik, P. (n.d.). *BespokeSoftware.org*. Retrieved 11 1, 2011, from Bespoke Software: http://www.bespokesoftware.org/wordpress/?page_id=41
- Varcholik, P. (n.d.). TACTUS: A Hardware and Software Testbed Research in Multi-Touch Interaction. 10.

- Whitaker, R. (n.d.). *XNA Tutorials*. Retrieved December 4, 2011, from <http://rbwhitaker.wikidot.com/xna-tutorials>
- Wilkinson, S. (2009, May 29). *Ultimate Vizio*. Retrieved 10 13, 2011, from UltimateAVmag.com.
- Williams, M. (2007). Ball vs Sleeve: A Comparison in Bearing Performance.