# Multi-Touch Table with Object Recognition
## codename Planck

Group 14

Pete Oppold

Enrique Roché

Héctor E. Rodriguez

Christopher A. Sosa

# Table of Contents

# 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 will be 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 is to offer a more intuitive input scheme that allows users to learn while doing.

Planck will be a 40 inch screen that is fitted into a table structure that can be used by the average size person. Planck will use a special mirrored particle acrylic that is illuminated by infrared light emitting diodes. An infrared camera will be used to pick up diffused infrared light which will serve as input to the touch recognition system. The application image will be displayed using a short throw projector onto a display mounted below the acrylic. The computer used to process all the recognition and application software will be fast enough to respond to a touch in less than 10 milliseconds and the power supply will be 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 Power 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 should be 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 will also be 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. Plank will utilize Diffused Surface Illumination technology. This technology is designed for large touch surfaces and has the capability of recognizing objects. Planck will use a sheet of edge diffused acrylic, which when illuminated by infrared LEDs, can be tracked by a cost efficient

webcam. Plank will also include a microcontroller, which will allow for the sensitivity and cooling within the device to be controlled. These technologies encompass the hardware that will be used to realize Planck.

The software systems in Planck 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 will require that users work together in order to accomplish military defensive objectives. The application will also show off the novel input that Planck is capable of and how traditional input is not required for complex applications.

## *1.2 Motiviation*

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 Planck is to create a novel tool set that can be integrated into several software solutions. Planck 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. Planck will be of sufficient size to accommodate multiple users simultaneously.

Planck 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*

- To remove the need for traditional input in favor of multi-touch gestures and real life objects
- To create a toolset that can be integrated into several software solutions

• To create a tool set that is intuitive and natural and allows the user to learn through experience and not require lengthy instructions.
• To facilitate collaborative work from simultaneous users
• Be of sufficient size to accommodate multiple users simultaneously
• To showcase the toolset with a battle simulator
• Provide a robust set of commands capable of being used in a variety of applications.

## *1.4 Requirements and Specifications*

**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 an 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 than 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 will be 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”, ½” 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 than 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.

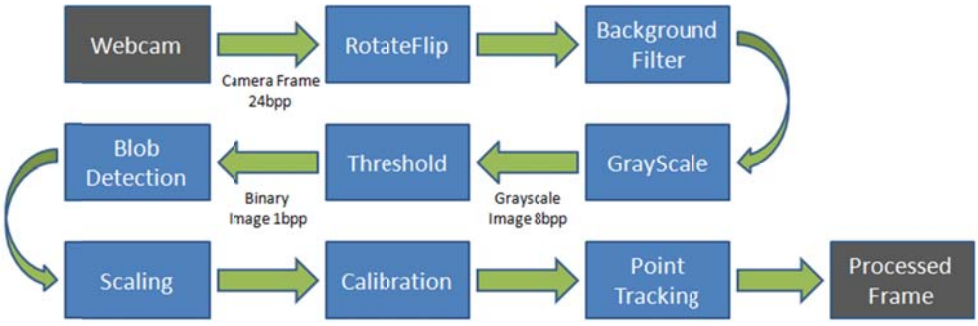



**Figure 1 - TACTUS Image Processing Process**

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.

**Table 1**

*TouchData Members*

| Type | Name | Properties |
|---|---|---|
| Int | ID | Unique identifier blob |
| float | tagID | Unique identifier fiducial |
| float | X | Horizontal position |
| float | Y | Vertical position |
| float | height | Height of fiducial |
| float | width | Width of fiducial |
| float | angle | Rotation of fiducial |
| float | area | Size of blob |
| float | dX | Delta movement in horizontal axis |
| float | dY | Delta movement in vertical axis |

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 Table 1 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 a 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 will be able 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 will use 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 will not be 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 will be 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', ¾" 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 - 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 form 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.

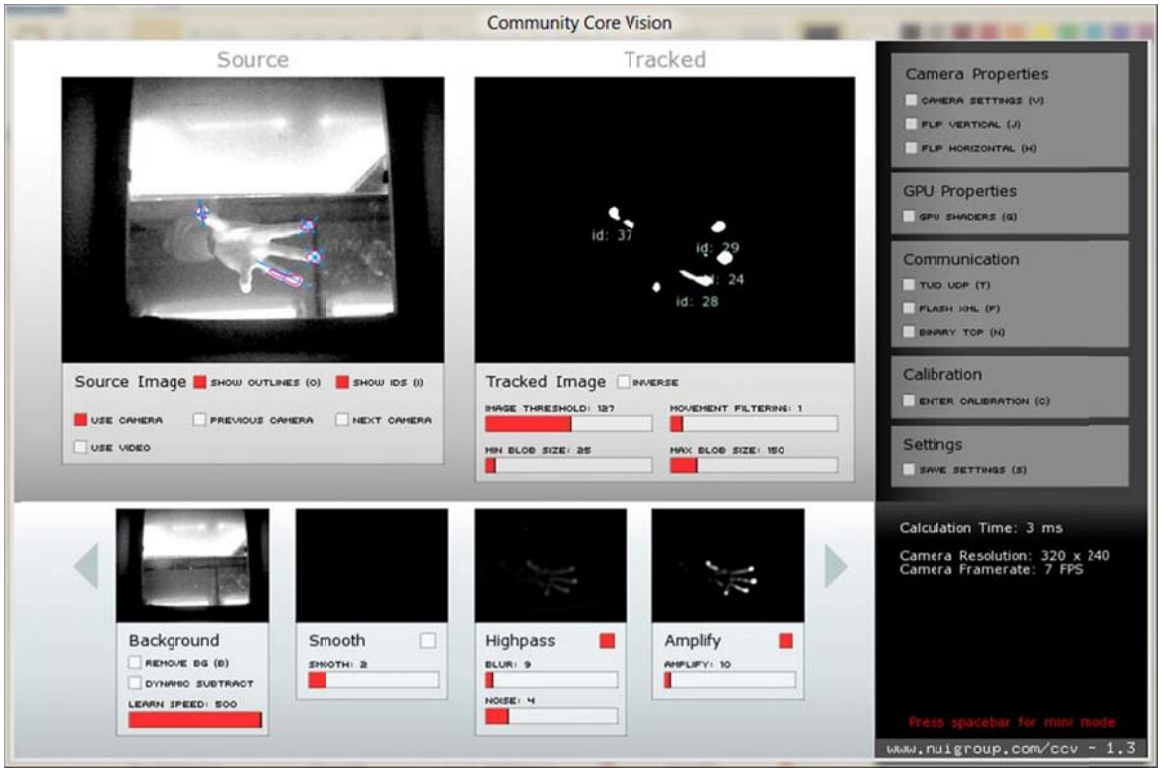



**Figure 3 - CCV Output from Prototype**

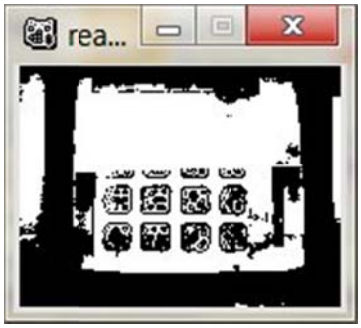



**Figure 4 - reacTIVision Output from Prototype**

# 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. The diagram below illustrates this idea.

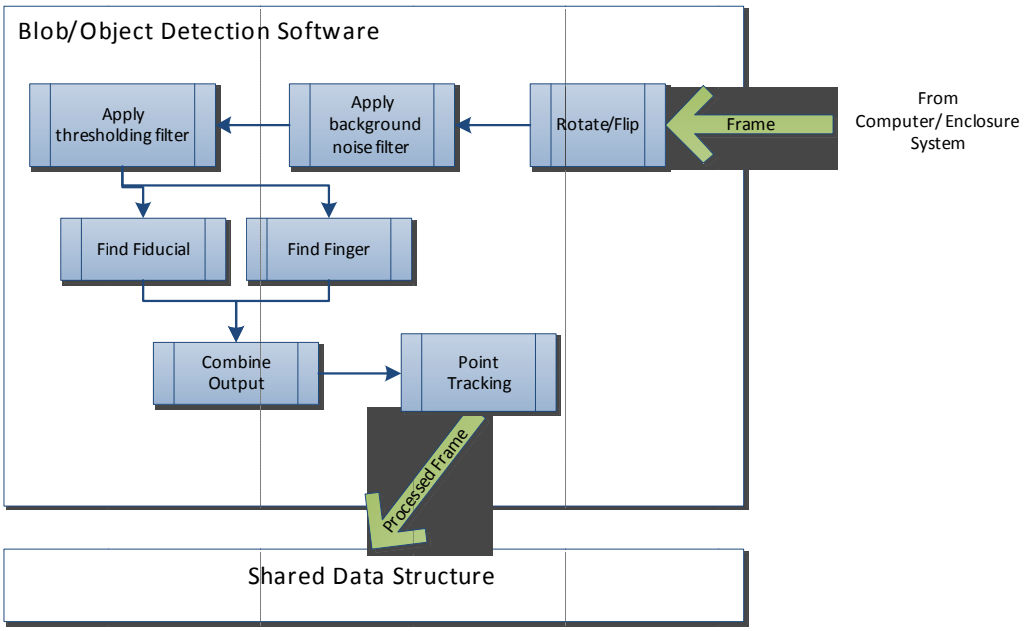



**Figure 5 - Block Diagram of Touch and Fiducial Recognition Software System**

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 will use 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 may be necessary 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 will use fiducials in the form of patterns printed on paper that can be recognized by the vision library we choose. These patterns will be adhered to an object whose shape will have a meaning to the user. When an object with a fiducial applied underneath is placed on the surface, the vision library will detect it. Each pattern has its own identifying ID number. Each unique ID number will have a specific meaning to the showcase software. An ID number will mean that a certain fiducial has been placed on screen. The showcase software will then output objects corresponding to that fiducial on the surface that interacts with the user.

The fiducials position, ID, and orientation will be acquired from the input video stream by the vision library chosen. This information will be sent to the showcase software so that it may 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 will prompt 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 blah 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. Fiducial is a Data Matrix type fiducial. B is a QR-code, as those popularized by Android OS on mobile phones. C is an example of M. Rohn's visual code. D is a ReacTIVision marker type. These are discussed later on in section 3. Type E is a TRIP type fiducial. F is an AR toolkit type fiducial and G is an early d-touch marker. These are also discussed later on.

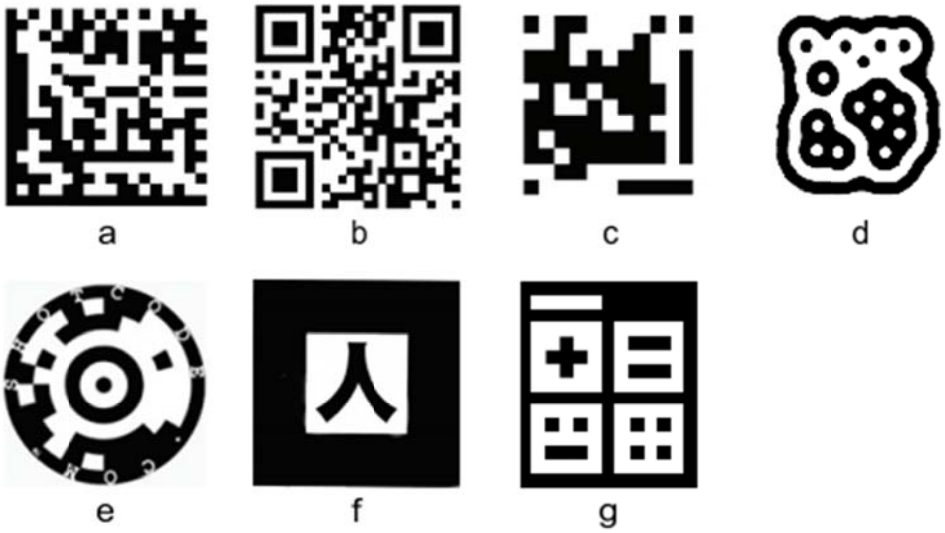



**Figure 6 - Example of Different Types of Fiducials**
**(Reprint pending approval)**

## *3.2 Vision Libraries*

Five vision libraries were considered to be implemented in Planck. The decision of which one is picked hinges on several factors. There are many requirements that must be met when choosing the vision library. First, the library needs to offer at least the minimum latency in image processing as set in the requirements of project Planck. Second, it must run on the computer that is chosen by the computer chosen, and support the camera chosen in the Image Recognition System. Third, the library must be able to work with Diffused Surface Illumination (DSI) technology. Fourth, the library needs to be reliably stable. Fifth, the library must support fiducials. Sixth, it would be helpful if the vision

library has a strong support community. Finally, the vision library needs to be versatile enough to communicate with the showcase application that will be 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 would be weighted more highly if it does offer both.

Community support of the vision library is also part of the equation in picking our vision library. Two of the group members have 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 can't 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.

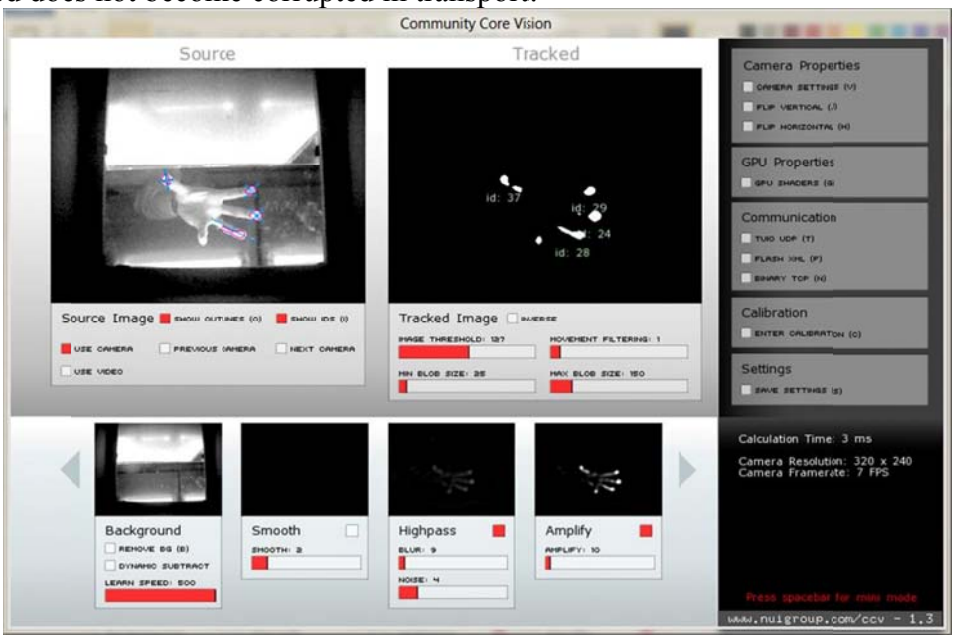



**Figure 7 - Example of CCV Configuration Screen**

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, ReacTable. It serves as the primary sensor in this table.

No specific information regarding the computer requirements could be found on reacTIVision's website. ReacTable's website contains requirements for the table. Since the performance needed out of Planck is similar to that of the ReacTable, 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, reacTable. 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, reactTIVision 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.

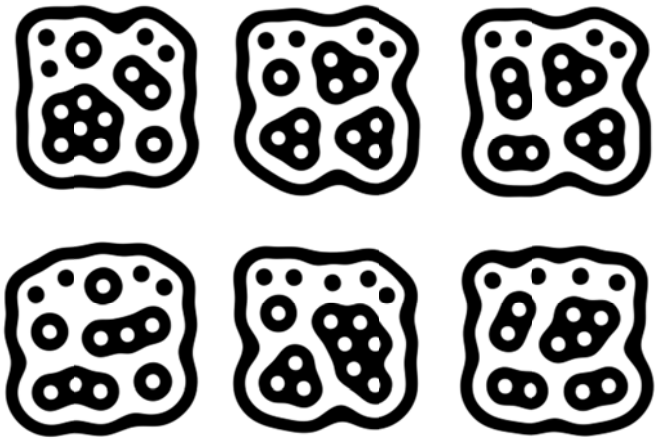


**Figure 8 - Examples of reacTIVision Fiducials (Reprint pending approval)**

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 ReacTable. 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.

DTServer 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.

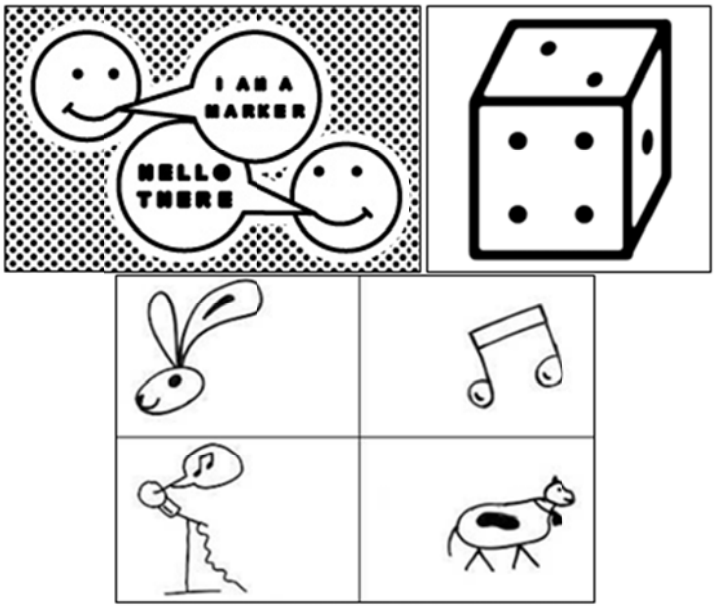



**Figure 9 - Examples of D-Touch Fiducials (Reprint pending approval)**

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×m1.5/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.

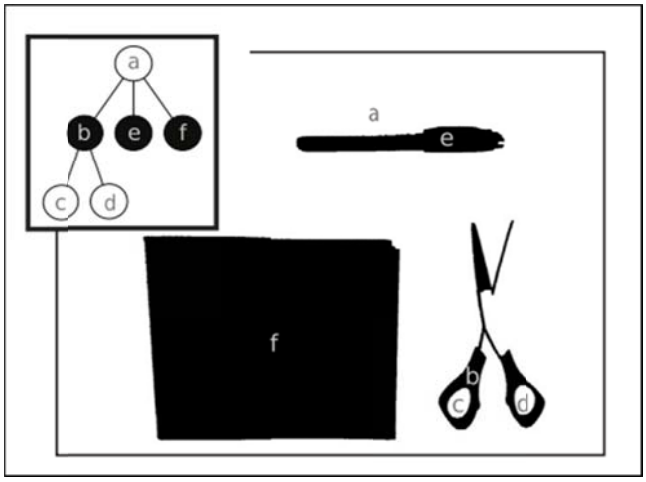



**Figure 10 - D-touch Process for Determining Fiducials (Reprint pending approval)**

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

| Osc Protocol | Value | Size (bytes) |
|---|---|---|
| Bundle Prefix | #bundle | 7 |
| Alive Message | | 32 per point |
| Point Messages | | 85 per point |

Point Method

| Osc Protocol | Value | Size (bytes) |
|---|---|---|
| Address Method | /gateway/point | 14 |
| TypeTag | ,biiiihffh | 11 |
| Payload | FtirPoint | 60 |
| | | 85 |

Alive Method — Size per point

| Osc Protocol | Value | Size (bytes) |
|---|---|---|
| Address Method | /gateway/alive | 14 |
| TypeTag | ,b | 2 |
| Payload | FtirPoint.ID | 16 |
| | | 32 |

Point Serialization

| Field | Data Type | Size (bytes) |
|---|---|---|
| ID | GUID | 16 |
| Rectangle.Location.X | Int32 | 4 |
| Rectangle.Location.Y | Int32 | 4 |
| Rectangle.Width | Int32 | 4 |
| Rectangle.Height | Int32 | 4 |
| TimeStamp | Int64 | 8 |
| Speed | Float | 4 |
| Direction.X | Float | 4 |
| Direction.Y | Float | 4 |
| Duration | Int64 | 8 |
| | | 60 |

**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 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 add unnecessary overhead to the system and would be inefficient given the substantial amount of processing Planck will be doing in order to capture touch events and gestures. The NUI user group states that the fiducial tracking algorithm of reacTIVision has 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 is the most applicable vision library to use in project Planck. It offers very versatile filters that will work with Diffused Surface Illumination

technology. It offers input from any webcam that can be used with windows. It also offers the ability to track fiducials using reacTIVision's fiducial tracking code and their physical fiducial objects. It offers quick and easy configuration and calibration on the surface of Planck. Finally, it offers 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 has 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. The following tables depict the attributes attached to each object:

| **Cursors, Blobs, and Fiducials** | | | |
|---|---|---|---|
| Session ID (s) | X and Y position on-screen (x, y) | X and Y velocity vectors (X, Y) | Motion acceleration (m) |

| **Fiducials also get** | | | |
|---|---|---|---|
| Angle measurement (a) | Class ID (i) | Rotation velocity vector (A) | Rotation acceleration (r) |

| **Blobs also get** | | | | | | |
|---|---|---|---|---|---|---|
| Angle measurement (a) | Area (f) | Dimension width (w) | Dimension height (h) | Rotation velocity vector (A) | Motion Acceleration (m) | Rotation acceleration (r) |

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:

```
x = sensor_x / sensor_width
y = sensor_y / sensor_height
```

(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:

```
X = (sensor_dx / sensor_width) / dt
Y = (sensor_dy / sensor_height) / dt
m = (speed - last_speed) / dt
```

(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.

## *3.3 Gestures*

In addition to the vision library chosen to deliver touches and fiducials, project Planck needs to have the ability to respond to gestures. For the showcase application, two distinct gestures have been defined and will be discussed later. There are several frameworks which implement the gestures needed and these will be discussed in this section. What follows is a discussion of the various software and libraries found that implement gestures. The software is all open-source and can be used as a baseline for gesture implementation. The gesture frameworks being considered are all written in various languages. Some of them also contain graphics API's which can be used to draw on-screen. This makes them less ideal to be considered for our project but since they offer open-source code that implements some of the gesture's that are needed, they may be used as references. All of these frameworks communicate with TUIO. They have been made to be used in applications such as ours. Lastly, a discussion on the creation of a tailor made gesture framework for Planck will be covered.

Research regarding gesture software libraries that work using TUIO is fairly scarce to come by. Some of the projects are still experimental. While theoretical discussions about how multiple touch gestures are less prevalent, many libraries that implement gestures were found. Consulting the NUI group site yielded a list of possible sources to be explored. Some of these sources contain research papers associated with them and those may be good sources for design ideas. For the design of this project gestures may have to be designed in house.

### *3.3.1 Grafiti*

Grafiti is a project started by Alessandro DeNardi at UPF in Barcelona where the ReacTable project came from. Alessandro DeNardi started the project as his Google Summer of Code project and wanted to publish the proceedings on the NUI group website. It was written in C++ and C# and contains a demo application to show case the software. The project is only in its alpha phase and hasn't been touched by the author in over a year. Grafiti is released under the MIT license and can be downloaded from the SVN repository where the author has placed it. It features the ability to read input from TUIO. It can work with both reacTIVision and CCV making it perfectly suited for our project on the input side. The library supports multiple gestures:

1. Pinching
2. Multitrace gesture
3. Lassoing
4. Tapping

These gestures can be manipulated to do what is needed for Planck's application. The pin-drag that we have described for our application is an instance of a pinch. Two fingers are placed on an object and the size of the object is changed based on the movement of the fingers. In the case of Planck, one finger can remain on the asset and the field of view can be changed with the other finger. Dragging of on-screen objects is also supported. A video on the website of the project can be seen showcasing the software in which dragging of objects is seen.

The site also states that this project is built on a C# layer which is independent of the client's software language. This bodes favorably for this framework since the language the showcase software will be written in will most likely not be C#. The input is also abstracted since it interfaces with TUIO.

One of the most favorable aspects of Grafiti is that it focuses on the use of tangible objects. Planck is specifically designed to capitalize on the implementation of fiducials. Grafiti has gesture recognition abilities to unite touches with objects on the screen as well as to unite two objects on screen together. Gestures can interact and affect the on-screen fiducials and behaviors involving fiducials and multiple touches can be programmed using this framework.

One of the most difficult aspects of the gesture detection system is the ability to deal with conflicts. Since multiple touches are being experienced by the surface involving different objects on-screen in a layered atmosphere, the same touch on-screen could mean several different things for different objects. Also, touches close in proximity could be misinterpreted. Fiducials near certain objects involving touches could also have conflicts. Grafiti does a great job of addressing these concerns by providing a priority mechanism and by allowing more flexible gestures to be designed.

One of the major downsides to Grafiti is that the author has stopped development on the project to focus on an iPhone app. As of the date of this writing, an e-mail has been sent to the author asking for information on its applicability to the Planck project. No response has been received but as stated earlier, the software can be downloaded from the repository and explored.

### *3.3.2 MT4j*

MT4j is a cross-platform multi-touch gesture framework that is similar graphically to the Swing library of Java. It can receive many different types of input and supports TUIO so it can work with CCV and reacTIVision. MT4j allows the creation of gestures but packages the 'most commonly used' gestures in the framework out of the box. Included are:

1. Tap gesture
2. Dragging
3. Rotating

4. Resizing by moving two fingers
5. Zooming gesture
6. Camera movement gesture
7. Lasso clustering

The dragging can be used to implement the dragging we need in our application. Gesture number four in the library can be used by the software showcase to implement the changing of a field of view. Lassoing is also implemented in this library which makes it a strong possibility to be used in Planck. This library supports all of the gestures we need to use with touches. There is also a handy tutorial for how to create gestures. This library allows the use of mice for testing processes and also handles the drawing of objects on the screen through OpenGL. It is distributed through the GPL License.

The biggest downside to this architecture is that there is no way to move these gestures from the framework to our showcase application. If this library were to be used, the showcase program would need to be programmed within it. This brings the positive aspects of speed and closeness of the two programs, but it may reduce the control we had by writing our own showcase software.

### *3.3.3 Kivy*

Kivy is an open-source toolkit to aid in the creation of hardware-accelerated user interfaces. It is actively being developed by much of the same team that created Python Multi-Touch (PYMT). The developer's goal is to create graphical interfaces faster and simpler than previous frameworks. One way of doing this is by letting the GPU do as much of the work as possible. Much of the under layers are also written in the programming language C. Kivy is written In Python and is cross-platform supported, including Android. Kivy focuses heavily on the support of the mobile Android platform and the use of widgets. It uses its own description language, The Kivy Language, for creating interfaces. Kivy does support the TUIO protocol for transfer of touch objects. Kivy provides its own graphical API in the bundle, using its own abstraction of OpenGL. The developer's did this as they believe that most beginners find openGL code to be confusing. Their API allows the user to create graphics through a simpler means that openGL does not provide, such as Canvas and Rectangles.

The main purpose for the use of a framework is the use of pre-written gestures. Kivy does various kinds of gesture stroke recognition. Examples of a few of these gestures are circles and rectangles. They break down touches into three categories; Down, Move, and Up. A touch is only Down when it first is recognized on the screen. The touch continues to Move as long as the same touch (specified by ID) stays on the screen. The touch moves to Up when it leaves the screen and is no longer recognized. Using these classifications, gestures can begin to be created by tracking the position in the Move state.

Implementing Kivy into the design of project Planck would require the team to develop the showcase and Vision System in Python and The Kivy Language. The implementation of other graphical libraries would not be necessary.

### *3.3.4 OpenExhibit*

Open Exhibit is a multi-touch framework that prides itself in its extensive gesture recognition library. It is written primarily in Flash which uses ActionScript. Open Exhibit is multi-platform supported, including Mac OSX, Windows, and Linux. It is widely supported by its own community and readily used by the users at the NUIGroup community. Open Exhibit is free to museums, non-profit organizations, students, and educational purposes. All others must use the licensed version, GestureWorks. Open Exhibit was developed by Ideum in collaboration with National Center for Interactive Learning (NCIL). Its partners are: the Don Harrington Discovery Center; the Maxwell Museum of Anthropology; and the New Mexico Museum of Natural History and Science.

One of the reasons Open Exhibit is so widely used is because of its extensive gesture library that allows developers to easily build multi-touch applications over the TUIO protocol. Open Exhibit contains all the gestures project Planck would require. Just a few of the supported gestures are zoom, rotate, flick, or scrolling of objects. A second reason it's so widely used is because of the implementation of Modules. A developer can choose to write their application that implements these pre-written modules to quicken the development process and add needed functionality to their program. A few of the modules that exist are Google Maps Viewer, flickr Viewer, Magnifier Viewer, and keyViewer. Google Maps Viewer is a module, based off Google maps API, which creates an interactive map for the user. Flickr Viewer uses the Flickr API to create multi-touch media that can be manipulated. MagnifierViewer magnifies multimedia objects to increase the user's experience. Finally, KeyViewer displays a functional keyboard that can be rotated, flipped and manipulated in several ways across the screen.

It is strongly suggested that users use Adobe Flash CS5 with Open Exhibits. It can communicate in TUIO, so it's compatible with any of the Vision frameworks that are under review in regards to project Planck. Open Exhibit has a graphics API in Flash so no other graphical libraries, such as XNA, are required. It is meant to be an all-in-one suite. The multi-touch hardware, vision framework, and Open Exhibit are all that's needed to create a novel application. Open Exhibit may not be the best choice, as Adobe just announced that they will no longer continue development on mobile Flash. Instead, HTML5 will be continued instead.

### *3.3.5 Writing and Original Framework*

Instead of implementing any of the libraries above it is possible to create one specifically for the use of Planck. This option is attractive since there are many example projects that implement TUIO listeners. Finding references and examples for creating TUIO clients would not be a problem. These examples cover communication with TUIO and the code implementation of interpreting the standardized TUIO input as gestures. The next piece of the puzzle is the algorithms involved in creating these gestures from the raw data given to us by the vision library through TUIO. All of the libraries discussed above are open-source. Their code implementations of two-finger gestures, lassoes, and drags are all downloadable and readable. Some of the downsides to using prebuilt gesture libraries is

that they would need to be implemented in the language they were written. This may limit the extent to which certain gestures can be edited. If any editing of the gesture needs to happen, it would need to be edited in the language it was written. None of the members have experience in Flash or Python so some of these libraries have inherent drawbacks. Another good reason to write a gesture library for Planck is because all of the gesture libraries discussed above contain their own graphics API's. Planck needs a specialized application and the research we've conducted has concluded that it would be more effective to implement proprietary graphics in XNA. If the software above were to be used, it would be very difficult if not impossible to design a system to allow the gesture library to easily communicate with any outside sources such as Planck's graphics software.

## *3.4 Touch and Fiducial Recognition Software System*

The gesture recognizer is 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 gesture 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, fiducials, and gestures.

There are two ways of implementing the gesture recognizer. The first way involves using a network port as a communications tunnel to feed the information to the showcase software. The other approach is to use a shared data structure and the gesture recognition module embedded inside the showcase software as a thread. It was chosen to use a shared data structure, with the gesture recognition module embedded. If problems arise with this method, then it will be programmed as a separate piece of software.

The first way of implementing the gesture recognizer application would be to have the gesture recognizer application running separately and then sharing its gesture data through a network socket. Some issues arise from the network socket configuration. A protocol would need to be used in order for the showcase application and the gesture recognizer to communicate via a network. In this type of system the gesture recognizer would have a socket listener that would be receiving input information from the vision library via TUIO. It would then interpret this information with a gesture recognizing algorithm and create gestures. Once the gestures are recognized and packaged, the gesture recognizer would need to use a socket on a port different than the one used to receive the input data to package the gesture 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 gesture recognizer would need to act as a client to the vision library, run it's 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 gesture recognizer wouldn't need to be built in the same language as the showcase software. Since the gesture recognizer uses a standardized protocol to send the information, both applications speak the same language. The main disadvantage with this approach is that the gesture 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 generates an unnecessary middle man in our infrastructure that can 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 gesture library in this way. The gesture library would be in charge of three things: 1-receive gestures via a network socket, 2-recognize and package the gestures 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 needs to happen within a latency of at most 0.5 seconds. The smaller the latency the more enjoyable the user experience will be. 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 is to create the gesture recognizer 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.

### *3.4.1 Top Level Program Flow*

The gesture recognition software will receive TUIO messages. Each object on the screen will be represented with its own unique ID number. Each fiducial will also have a unique ID number. This list of on-screen objects will be received through a socket and read first by the gesture recognition system. The list of received objects will be stored in the program temporarily. When an object first comes into the gesture recognition system, it will be checked against all the current elements in the shared structure to see if they already exist. If it does, the object will be updated with whatever the newly received information is. If the object isn't in the list, a new container for it will be created and the object will enter a function that will determine if it is a touch or a fiducial. If the function determines that the object is a fiducial the fiducial ID's are determined and the object and its corresponding information are packaged to be placed in the shared data structure. If the object is not a fiducial, it must be a touch. Touches are sent to a function that determines which type of touch it is. We're looking for two different types of touches: two-finger pinches and one-finger drags. In this function, every touch is considered a one-finger drag until another touch is found within some radius of the first. This design is based on the idea that two users won't touch areas close to another user. Also, since a two-finger pinch is designed to operate under on the object under the user's finger, it's unlikely that one user will try to modify the object under another's control. If the finger touch is found to be a one-finger drag it is packaged by the gesture packager to be placed in the shared data structure. If the finger touch is a two-finger pinch, a function will look through the other one-finger drags to see if any of those are close to this one. Closeness will be determined by seeing which of the other one-finger drags are within a certain distance threshold of the touch currently being assessed. If they are close together, the two touches are linked together and added to the shared data structure as a two-finger drag. Once all of the new touches have been saved into the gesture data structure, the socket is checked once more for new data from the vision library and the process begins

again. When an object is removed from the surface it will be searched for in the list of gestures and will be removed.

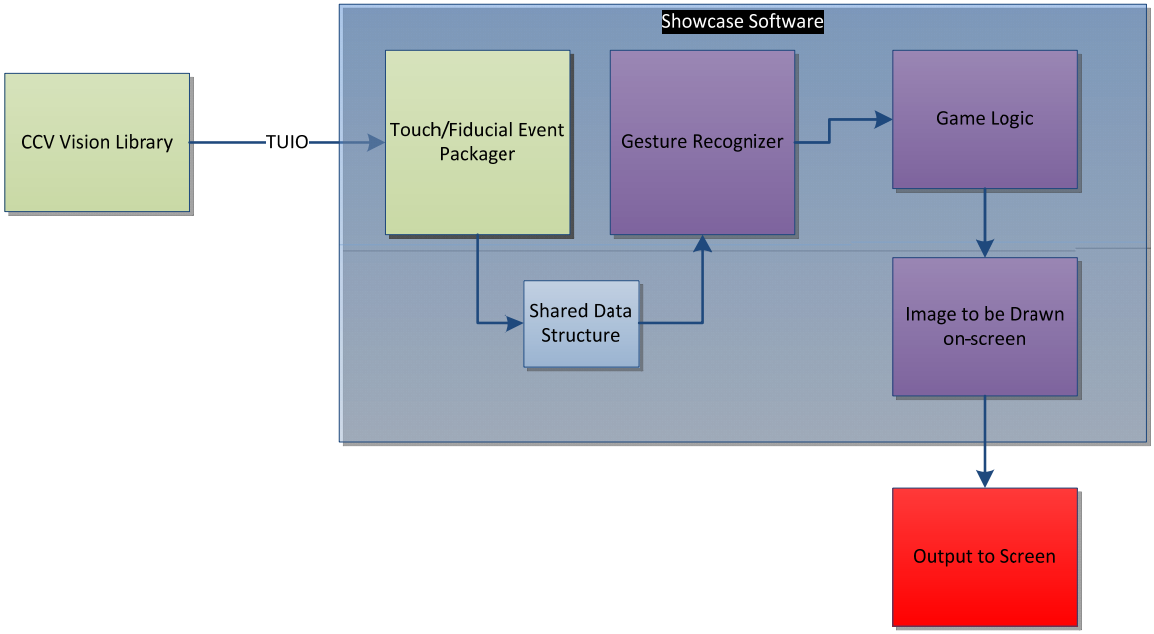



**Figure 12 - Top Level Program Flow**

### *3.4.2 Reading from TUIO*

Messages are received from CCV 1.5 vision library over TUIO. These are read in by the gesture recognizer program using a TUIOListener class. Messages are received by the gesture recognition program whenever objects on the surface are added, removed, or moved. Messages are sent to any program that is registered with the TuioListener interface. This interface is 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 Figure 5Table 2.

**Table 2**

| 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 sends a message entitled “refresh”. The purpose of this message is to indicate the end of a packet of information from TUIO. It can be used to repaint the screen of a graphical program. In our application it will serve 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 is seen on the screen messages are received into the gesture recognition application and these are objects can be queried based on their type for the information we need about the gesture. The two types of objects that TUIO sends inside the touch events are objects and cursors. Objects can have the following fields within them:

Cursors can have the following table of fields associated with them:

**Table 3**

| 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 |

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.

**Table 4**

| Class TuioPoint | | |
|---|---|---|
| **Variable Name** | **Variable Type** | **Variable Description** |
| currentTime | Protected TuioTime | The time stamp of the last update represented as TuioTime (time since session start) |
| startTime | Protected TuioTime | The creation time of this TuioPoint represented as TuioTime (time since session start) |
| Xpos | Protected float | X coordinate, representated as a floating point value in a range of 0..1 |
| Ypos | Protected float | Y coordinate, representated as a floating point value in a range of 0..1 |

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.

As each touch is received from TUIO it will be added to a linked list array that contains all of the touch objects. In the event a fiducial is received on the surface, the object added to the linked list will be the object type defined earlier. Since each fiducial object has a unique symbol ID, each fiducial will be able to be differentiated from another. Additionally, each cursor and fiducial object has a unique session ID. In the event more than one instance of the same fiducial is found on-screen, the session ID can be used to distinguish the two. The fiducial, once added to the linked list, can be updated as needed based on the information received from the vision library through TUIO. Touch objects will not be automatically added to the linked list of touch events. They will be stored in a temporary cursor and will be checked against every other touch cursor in the list to see if they reside within a certain radius of another touch cursor. The radius will be determined by the size of an on-screen object in the showcase software. If a touch cursor is found to be within the threshold defined for the size of an on-screen object, the two cursors will be linked together in an object that comprises them both. This new object will constitute a pinch-drag gesture. There will be a Boolean value set to true in the container object that defines it as a pinch-drag gesture. If there is no other touch found near this cursor, the object will be packaged as a drag and will be added to the linked list. In this case, the Boolean value that defines a pinch-drag will not be switched to true.

The gesture container object will have the following elements:

**Table 5**

| Object Name | Object Type | Object Function |
|---|---|---|
| multi | Public Boolean | Defines whether this gesture is a pinch-drag or cursor. |
| | | T=pinch-drag |
| | | F=cursor |
| CursorOrig | TuioCursor | First touch detected on-screen with no adjacent touches found within threashold |
| CursorDest | TuioCursor | TuioCursor object that is the second touch on-screen linked to the first. This object originally resided within the threshold of CursorOrig TuioCursor |
| Partner | Int | Defines the ID number of the touch that is partners with this touch |

A few other methods that come with the TUIO standard involve the ability to write the gesture recognizer so that it can poll the vision library for new touches or fiducials. There are four methods provided by the creators of TUIO for these purposes. They are getTuioObjects(), getTuioCursors, getTuioObject(long s_id), and getTuioCursor(long s_id). A table summarizing their attributes can be found below in TABLE. 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. At this point the GestureTracker will be implemented using an event driven design, awaiting the output of the vision library and proceeding from there.

**Table 6**

| 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 gesture tracking software will be implemented as a thread of the main showcase software. This class will be called "GestureTracker". In order to it to listen to the vision library and TUIO, it must implement an instance of TuioClient. It must also implement the TuioListener class. A code snippet, found in TABLE 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.

```
GestureTracker application = new GestureTracker();
TuioClient client = new TuioClient(port);
client.addTuioListener(application);
client.connect();
```

A class diagram of the GestureTracker is shown in the diagram that follows. There are several methods. They are:

1) addTuioObject: accepts a touch cursor and adds it to the list of touches currently found on-screen.
2) updateTuioObject: accepts a currently existing touch cursor and updates the properties of this cursor with the properties of the new object.
3) removeTuioObject: accepts an object that currently exists in the list but is not on the surface anymore and removes it from the list of gestures.
4) addTuioCursor: accepts a new cursor object and adds it to the list of touches on-screen.
5) updateTuioCursor: accepts an existing cursor object and updates the information of this object with the newly received object.
6) removeTuioCursor: accepts a cursor object that currently exists in the gesture list but no on-screen anymore and removes it from the list.
7) refresh: accepts a TuioTime object and can be used to determine when a bundle of messages has ended. A 'refresh' is sent at the end of every TUIO bundle of messages.
8) packager: This is an overloaded method that accepts either a TuioCursor object or a TuioObject (fiducial). If the object received as input is a TuioObject (fiducial) it is added to the list. If the object received is a touch its corresponding touch is looked for to determine if it is a pinch-drag gesture and then it is added to the list. If no partner is found, the object is added to the gesture list as a single drag.
9) findPartner: accepts a TuioCursor object, gest1, and finds if any other TuioCursor object in the linked list is close enough (within a threshold) to this object. If there are several objects within the threshold it picks the one that is closest. If a close object is found, it returns true and links both objects together. Otherwise, it returns false and links nothing.
10) isFid: returns a Boolean value stating that the object passed to it is a fiducial. If this returns false, the object must be a touch.
11) getList: returns a linked list that is populated with cursors and objects.

Gesturelist is a linked list data structure that defines all of the current gestures and objects that have been found on-screen. It will contain GestureObjects (pinch-drags), TuioCursors (one-finger drags), and TuioObjects (fiducials). Gesturelist will be a global linked list that will be peeked at by the showcase application using a public getter method, getList(). This list forms the basis for what is to be drawn on-screen as well as linked to the objects inside the showcase program for interaction.

As stated in earlier sections, the findPartner() method takes in one TuioCursor object, lacking an attached partner, and searches through the linked list looking for other TuioCursor objects that also don't have partners. A Boolean value stipulating if these two are within range is returned indicating that these two should be made partners. The two TuioCursors are then pointed to each other using their ID numbers. This is accomplished by making the int variable, partner, of each TuioCursor equal to the ID

number of its corresponding partner. Partner will be the ID number of the partner with which this TuioCursor is associated.

The logic behind this gesture is that a touch within a certain distance of another should be related since the user will only be able to modify single units using multi-touch gestures. For example, if user1 wishes to modify the properties of an object displayed on screen and touches that object, within fairly short order, an additional touch should happen near this first touch. Any touch within user1's first touch will be assumed to correspond to this first touch. One downside to this is that if another user (user2) touches the screen near where user1 did, those two touches will be bound as one gesture. The case where this occurs will be mitigated by two things: 1-a visual indicator will be displayed on-screen as soon as the user touches the screen to assist them in defining the area they can touch to affect this object onscreen, 2-when user2 sees the indicator popup, they will understand the area is restricted from outside use until user1 is done with their touch. They are still free to touch locations outside the restricted area, however.

The function, findPartner(), will take the first object,gest1, and procure its x and y position. The second (gest2) object's x and y position will also be acquired inside the function. The x and y position from each gest object will be subtracted to calculate the distance between the two objects. If the absolute value of either of these subtractions is within a certain threshold, the two will be linked together. Below in table is a pseudo code illustration that shows how the decision on pairing gest objects is achieved.

```
If ( abs ( gest1.getX – gest2.getX) > threshold )
        link gest1 and gest2
else if ( abs ( gest1.getY – gest2.getY) > threshold )
        link gest1 and gest2
```

# 4. Showcase Program

The showcase program is needed to demonstrate the user interface that the multi-touch table is capable of. To show off this innovative interface a human controlled battle simulator can offer an interesting environment to implement multi-touch inputs. This battlefield simulator will be 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 can demonstrate that multi-touch surfaces can be used in time sensitive applications. In addition to simply using touch instead of a traditional mouse input the keyboard commands and shortcuts can be replaced by real life objects with fiducials imprinted on the underside. In this way real life objects can be selected based on their relevance to what is 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 will create a new paradigm for RTS input. The inputs will still accomplish the same tasks as they always have in a RTS scenario; however the physical input from the user will be dramatically different and arguably easier to learn and master. All traditional input will be accomplished using two different methods; physical touching of the screen and using fiducial imprinted objects. All the inputs that previously required a mouse will be handled by single or multiple finger touches. Any kind of selection of a unit or building as well as giving units commands can be handled through single finger touches. Unit abilities and special functions can be handled by using multiple fingers, double tapping the screen, or creating finger gestures that are 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.

The build menus in RTS software are usually mapped to a key on the keyboard and the items within the menu are mapped to other different keys sometimes the same key that opened the menu in the first place. There are usually two to three of these menus; however some entries in the genre have been known to use more. Real life objects can take the place of all these keyboard shortcuts by imprinting fiducials on the bottom of the object. Military structures in RTS scenarios are built by using a conjunction of the keyboard key to select a certain structure and then using the mouse to place it on the terrain in the battlefield. With fiducial imprinted blocks, the user could simply place a block directly onto the exact screen coordinates where the structure should be built. With these capabilities the blocks could be constructed to look like the structures, removing any ambiguity in the link between the fiducial and its function. In this way the user is freed from memorizing key combinations that could be difficult to remember and simply has to know what the structures look like.

Fiducials can also be used in creative ways that don’t involve structures. Since the camera can also detect when fiducials rotate and translate, many other functions can be implemented on non-static battlefield objects. Using these two input options, Planck

offers near limitless types of input that come naturally to the user and is theoretically bound by the designer's imagination.

## *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 is needed. The RTS battle simulator needs 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 will require a novel piece of software that exhibits the unique interface. For this to be accomplished, a graphics API will be 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 Application*

In order to show off the user interface provided, Planck must run a showcase program. As explained above, an RTS model of a modern battlefield will be created to show the power that multi-touch and fiducials have. In conjunction with the hardware, this program will propose what a futuristic battle command center might be like. The system as a whole has many benefits such as: real time communications link to higher chains of command, visibility of the operation area, and a broad real-time understanding of the battle.

The model could be representative of a modern day battle scenario. In order to be successful, a set of mission objectives must be completed using the units and support structures offered at the beginning of the mission. The application will approximate a real life scenario in which preliminary objectives put forth might change as the mission unfolds. It will show that having a tool like Planck can remove the burden of having to make decisions that require awareness of the entire battlefield based solely on information provided by the soldier in charge on the ground. Instead a higher ranked officer who has total awareness of the battle can make a more informed decision about what a certain unit should do next. The application also has the ability to be used as a simulator to train personnel in working together with other officers to accomplish a common goal. Because multi-touch can handle a nearly infinite amount of touch based interaction it allows multiple commanders to use a divide and conquer methodology to achieve all the mission objectives. If friendly forces are split into two groups, both commanders will be able to command different platoons and work together to meet their objectives.

All scenarios will take place in a dessert environment in order to model the most recent conflicts. Most objectives will be centered on eliminating the insurgent presence around a certain town or city without incurring too many civilian casualties. Some missions will require the user to use only infantry and heavy weapons, while other scenarios will require joint strikes involving all types of units including infantry, air, and naval units. All missions can be accomplished without the help of others; however, teamwork is encouraged whenever a scenario involves more than just a few infantry units.

### *4.3.1 Basic Commands*

The basic mechanics of a traditional RTS allows for multiple units to be rendered onto a screen on top of a terrain which covers the entirety of the map. Units will be ordered around by selecting them and pointing to a location on the map. In order to select multiple units the user will drag their finger across an area containing at least one unit. The location of the first finger touch will be in the upper left or upper right coordinates of a square depending on the direction the finger is moving. This action will create a rendered translucent square which will expand and stretch along the diagonal of the square in the direction of the moving finger. When the user releases their finger all the units which lie inside the translucent square will be grouped together and available to accept commands.

All units usually have a green health bar that hovers above that unit. When any unit, friendly or enemy, is attacked, the unit's health bar will begin to decrease, visible by the lack of color in portions of the bar. The health will also be represented as a percentage which is based on a number of maximum hit points that a unit starts with. Every unit will have a different amount of hit points, but if a unit's hit points reach zero that unit will be been eliminated. Units that are at twenty percent health or lower will be immobile or have very limited mobility and will have to be rescued in order for them to survive.

Each different type of attack deals a static amount of damage, which is subtracted from a unit's total hit points when they are hit by that attack. Every unit has a basic attack that they can execute against an enemy unit. However, the damage each unit deals is proportional to the weaponry and training possessed. For example, a tank usually deals orders of magnitude greater damage than an infantry unit. Also, the insurgents do not deal as much damage as the organized military units due to their lack of proper training as soldiers. Some units are also able to carry out special attacks such as throwing grenades or clearing houses and buildings. The special abilities can be activated by using either a specific multi-touch input or by using a fiducial.

In most cases the basic objectives can be achieved by finding and reducing all insurgent units' hit points to zero. In order to accomplish the objectives a commander will have to use the units and resources available to him or her. At the beginning of each scenario a certain amount of money will be allotted in order for the commander to purchase units and reinforcements. A commander can use the command post to purchase reinforcements and units if he or she feels they are needed. Some scenarios will require a decision to be made when presented with an event such as an unknown vehicle approaching a

checkpoint or a group of civilians being held hostage. These types of events simulate how an officer would have to give a set of rules for how his troops should handle situations which will arise on a day to day basis in a war time environment.

There could be a total of nineteen units, thirteen coalition units, and six insurgent units, ranging from infantry ground soldiers and marines to air force fighters and bombers who can work in conjunction with naval air craft carriers. It could also include enemy forces such as local insurgent militia and improvised explosive devices to pickup trucks that have been modified to carry a heavy machine gun.

### *4.3.2 Coalition Forces*

Coalition forces are military assets that the user can command to carry out the objectives put forth by the scenario. Coalition forces are made up of highly trained soldiers as well as battle hardened experienced officers. They utilize highly specialized units in order to complete objectives while minimizing collateral damage. If all the coalition forces in a scenario are defeated then the mission is a failure.

The first and simplest of all the units is the different infantry or ground units available. Most missions will be close to impossible to accomplish if all ground troops are dead. The first and least expensive unit is the soldier. The soldier consists of a platoon of new recruits along with low ranking officers who have been trained to fight and are offered standard issue military equipment. At least one of these units will always be available at the start of a mission. The soldier's normal attack is to fire their machine guns upon the targeted enemy unit; their special ability is to throw a grenade which deals four times the damage it normally would. When the soldier unit is selected this ability can be activated by making a relatively small circle with a finger around the area the grenade should land, the soldier unit will then move towards that area until it is in range to throw the grenade. The medic unit is a single man who can be added to the basic soldier platoon in order to give medical attention. The unit's ability can be used to recover some of the soldier's lost hit points. The engineer unit will have two abilities, first to create bridges and roads which allow units to pass rough terrain and rivers, and second offering intellectual conversation to stimulate the minds of all other units which has absolutely no effect whatsoever on their performance or hit points because no one ever listens to engineers. The special forces unit is usually extremely expensive and is necessary to diffuse highly volatile events such as hostage situations and snipers. The last ground unit is the marine; they are identical to soldier units except that they can be deployed on any coastal area and have slightly more hit points.

The next set of units is the heavy weapons units which consist of tanks, artillery, and Humvees. Tanks are hard hitting units with lots of hit points capable of destroying buildings and automobiles easily. Tanks are specialized units usually used for destroying other tanks and providing cover for infantry units. The user must use them with caution however because of their destructive capabilities. It is very easy to harm too many civilians with the use of tanks in populated areas. The artillery unit is a stationary unit that stays close to the command post. This unit can fire upon enemies who are very far

away. In order to use artillery, the enemy unit must be in sight of a unit such as infantry or tanks. The last heavy weapons unit is the Humvee. The Humvee's main purpose is ground troop transport; however it can still be used as a basic attack unit. The Humvee's normal attack uses a machine gun that is mounted to the vehicle. The Humvee is also capable of loading ground units into it. This can be done by touching a ground unit and then touching the Humvee. A Humvee with a platoon of soldiers in it benefits by having increased firepower.

There are only four other units: two of them naval and the other two are air assets. The two naval units are destroyers and aircraft carriers. Destroyers can defend coastlines and carriers, as well as being able to bombard coastal areas. In addition to these abilities, destroyers can also deploy marines onto coastal areas. To do this, a fiducial could be placed on the destroyer enabling the user to purchase marine units. After they are purchased, the user could drag marine units out of the destroyer and onto land. Air craft carriers hold both fighter and bomber type air units. When a fiducial is placed on an air craft carrier it enables the user to select fighters or bombers, and in the same way as marines, they can be touched and dragged to be sent on bombing missions. The last and most expensive units are air units. Fighters and bombers can be used to bombard areas with heavy enemy presence to make it easier on ground troops and tanks to take over an area. The fighter however has the ability to also engage other air units in air to air combat, should the need arise.

### *4.3.3 Insurgent Forces*

The insurgent forces are the opposing forces which stand in the users' way of accomplishing their objectives. They are comprised of a handful of different ground and heavy weapons units. Their goal is to hinder coalition forces, convince local leaders to fight against the coalition, and terrorize anyone who is affiliated with coalition forces. They rely on guerilla and terror tactics such as ambushes, explosive traps, hostage taking, murdering civilians, and using religious martyrs to kill unsuspecting soldiers. They do not wear any kind of military uniform identifying themselves as insurgents and therefore are difficult to identify and eliminate.

The most basic ground unit for the insurgents is the militia unit. These are local people who have been convinced by the insurgents that it is not in their countries interest to have coalition forces fighting the insurgents. This unit has a very weak normal attack due to their outdated weaponry and lack of traditional combat training. Their normal attack is to fire their weapons when they are in range of dealing small amounts of damage relative to coalition soldiers. Their range is not very good due to the fact that they lack traditional firearms training and must get very close in order to hit the target. Militia possess no special abilities, however they occur in large numbers and know the terrain and villages even better than the insurgent soldiers allowing them to ambush coalition forces. Another insurgent ground troop is the insurgent soldier. This unit has been given extensive firearms training and as such is about as capable as coalition new recruits. These units carry more modern machine guns which they use effectively as their normal attack. Insurgents have also been trained in making anti-personnel explosives and occasionally

carry grenades and Molotov cocktails. They also have been trained to create difficult situations for standing armies like hostage situations. Insurgent soldiers do relatively high amounts of damage as compared to coalition soldiers.

There are two ground units that are specialized in the insurgent force. One is the RPG, or rocket propelled grenade team, they specialize in destroying tanks and Humvees. Although RPG teams should be taken seriously and eliminated quickly, it is known that their equipment is extremely dated and lacks any kind of accuracy. Hidden or unknown RPG teams can be deadly to a convoy of Humvees or tanks and can cause a large amount of damage to ground troops. The other type of specialized ground troop is the martyr. The martyr is usually a religious fanatic picked from the militia's ranks in order to carry out a terror mission. These units are always used to ambush coalition forces and use a bomb strapped to their bodies to cause enormous amounts of damage to both coalition forces and civilians alike. They are difficult to eliminate due to the fact that they usually appear to be normal citizens. The martyr can choose to detonate the bomb on him at any time but usually waits till he has many units in the blast radius to do so. If any infantry unit is within the blast radius when the bomb goes off they will instantly lose all hit points and die. The martyr never survives once the bomb has been detonated, hence why they are called martyrs.

The insurgents only have very limited amounts of heavy weapons and highly destructive military hardware. The technical, another insurgent unit, is a civilian vehicle that has been outfitted with a machine gun, usually a pickup truck. The technical's normal attack is a machine gun burst that can be quite deadly to infantry. However due to bad engineering, the accuracy of this weapon is nowhere near what coalition forces have mounted on Humvees. Because of this, technicals do slightly less damage than Humvees because of their lack of accuracy. The technical can also transport militia and insurgent soldiers increasing its attack power while the units are in the truck. Mortar teams are one of the greatest assets the insurgents have. Mortar teams are similar to coalition artillery. However, they take less time to prepare and are mobile. They also do much less damage due to the shells being much smaller and older. Mortar teams are usually situated in mountainous areas where they can get better range, line of sight targeting, and camouflage in order to hit their targets. The last weapon the insurgents have is the IED, or improvised explosive device. These devices are homemade bombs that are usually filled with nails, glass, or metal shards, or sometimes all these things. These bombs are usually hidden along roads or inside inconspicuous objects. The IED unit cannot move but when any coalition unit enters its detonation radius the trap will deploy and explode. Any unit that is within the damage explosion radius will be removed from the scenario and can no longer participate. The only unit that can destroy an IED is the engineer unit. Engineers can detect and disable IEDs. The detonation radius for an IED will be about the width of a road in the application. The explosion radius however, will be about two to three times that radius.

### *4.3.3 Implementation of Units*

Each unit belongs to one of four sets. These are the ground troops, heavy weapons, naval units, and air units. Each of these sets is implemented as a class. Each class has certain parameters that define basic attributes of that set such as cost, x and y coordinates on the screen, movement speed, texture size, and many others. Each unit is usually defined as a subclass of that class which contains specifics to that unit such as hit points, damage it is capable of inflicting, the different abilities it possesses, as well as the texture image associated with that unit. Each unit will also have its own identification number so that interaction between units is possible. This is how the application will determine which unit to apply a special ability or damage received to.

Each unit will be graphically displayed by using a three-dimensional model or as a two-dimensional texture. There are many model repositories that have been published as open media for use in graphics applications such as Google Warehouse. Models for RTS type units are not uncommon among graphics forums and game development communities. Using these models is more desirable than using two-dimensional textures for many reasons. Textures are image files which must be loaded into specialized GPU memory that allows texture access. The process of mapping these textures to precise locations in a rendered environment can become costly if the texture must be accessed many times. The graphics programmer must also decide on object data such as normals, colors, and material properties that are not inherent in the texture image data. This process must be done for every specialized unit texture and if animation is desired, all the different animated poses of the unit must exist as texture images and be accessible. Once all these parameters have been set the specific texture is referred to as a sprite. Game developers traditionally used sprites in the early development of video games because the systems that they were developing on were not as powerful as those found today. Sprite texture images were limited to sixty-four by sixty-four 8-bit color images in order to reduce the expense of loading textures. This option is feasible for any graphics application. However, the more modern solution of using model meshes creates more desirable, true three-dimensional results. In this scenario, a model exists inside of a file as a formatted list of data that is created by using computer aided design software. This software creates complex models such as a car or tank by specifying the material properties, normals, colors, and vertices that define the model. The vertices of the model are the direct floating point data needed by a graphics API to draw the object. These models are broken down into smaller meshes which are also made up of the same data that makes a model. In this way a graphics API can use a model loading API that is capable of reading the file and displaying the model in true three-dimensional space. The benefits of using models over sprites are: models are truly three-dimensional where sprites are two-dimensional, they contain all material, color, normal, roughness, and vertex data in the file, they do not require the GPU to do many texture access, and some file formats support animation data. Therefore all units will not only require their class attributes and status methods used to update their logic data, but will also require graphics attributes and a rendering function for each type of model. The reason a separate rendering function is needed is because models are not native to graphics APIs, such as OpenGL and DirectX/XNA. Instead they come from outside files such as collada, 3ds Max, lightwave, etc. In order to read these files a model loading API is needed. OpenGL has many options such as ASSIMP and

DevIL. XNA supports only .x and .fbx asset types, which are loaded by Microsoft's asset importer.

Each object in the application which can be interacted with will also require an event identification object which ties it to a certain touch or fiducial event that exists within the gesture recognition thread's linked list. The object will not only have the identification number but will also have the unit's screen coordinates so that a kind of locality can be established and determine whether the input event is related to that application element. Special Commands and interaction will require two touch events both of which must be in the spatial range of the object being interacted with. This range can be determined by finding the difference between a unit's screen coordinates and the screen coordinate of the touch event and seeing if that result exceeds a certain threshold. If it does exceed that threshold than the unit in question will ignore the input that was associated with that specific touch event because it is too far away to be related to it. If it does not exceed the threshold, then as long as the touch event's identification integers stay the same, the existing gesture can be directly linked to an application object. That unit or object will only be able to be controlled again once the current bound touch event has ceased to exist in the gesture list.

### *4.3.4 Terrain*

The terrain can be rendered in a variety of different ways depending on what is needed. The simplest terrain consists of rendering a large flat plane in world space and then overlaying a texture to make it appear as though it is indeed a terrain. This method has the benefit of being extremely simple and straightforward in both implementation and computation needs. This method can be expanded by applying more than one texture image to different locations on the plane as well as blending these different textures together allowing for something which looks more realistic. These methods can be used to make flat textured terrains feasible. However, a more believable terrain should have some varying heights in the yz-plane. This more aesthetically pleasing terrain can be made by using a height map. A terrain height map is an image which can be referenced by a graphics system in order to use the variation in color as the variation in terrain height. By using this height map it is possible to assign the height position for each vertex that makes up the terrain. Once the mesh for the terrain has been created it is possible to overlay textures of grass, rocks, and dirt to make the terrain look believable. In the three figures below is an example of the rendered terrain mesh, the height map image, and a multi-textured terrain created in XNA Game Studio (Grootjans, 2003-2011). Riemer Grootjans has tutorials online which are open source for using XNA to create complex terrains and even a flight simulator game. All of his tutorials can be accomplished with a very basic knowledge of programming and little to no experience in graphics or game programming. A team of programmers with very little knowledge of graphics and game programming can make basic, yet publishable, media with little overhead and difficulty.

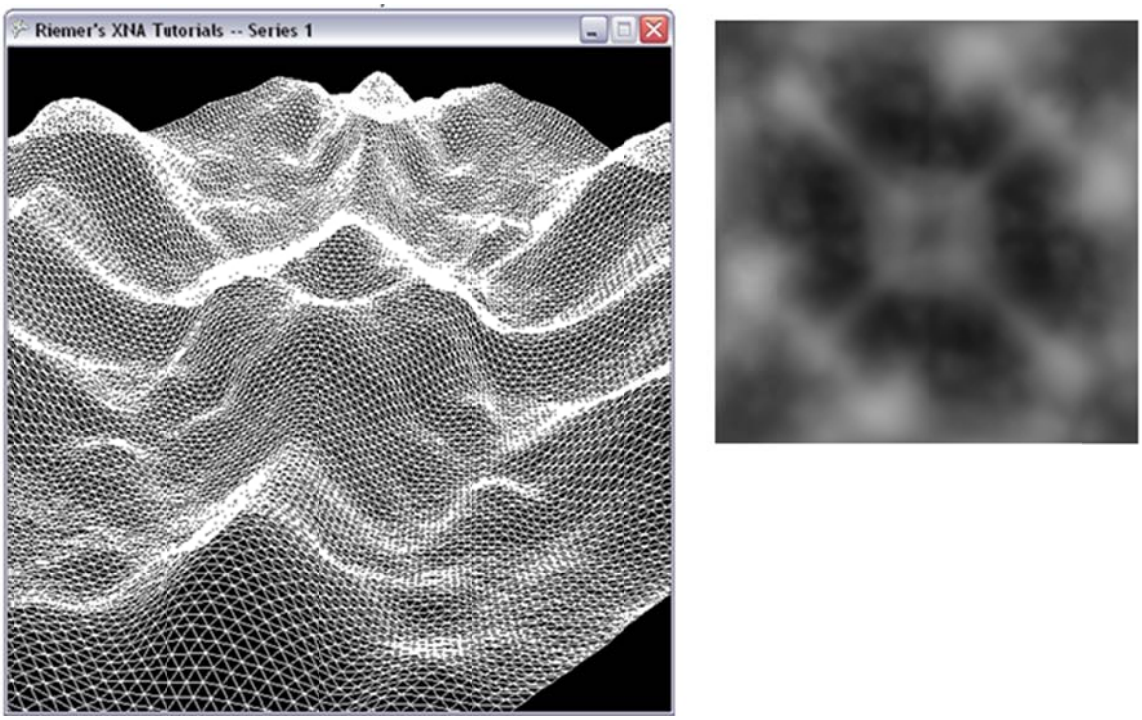



**Figure 13 - Example of Depth Map and Resulting Picture from Reimer (Reprint pending approval)**

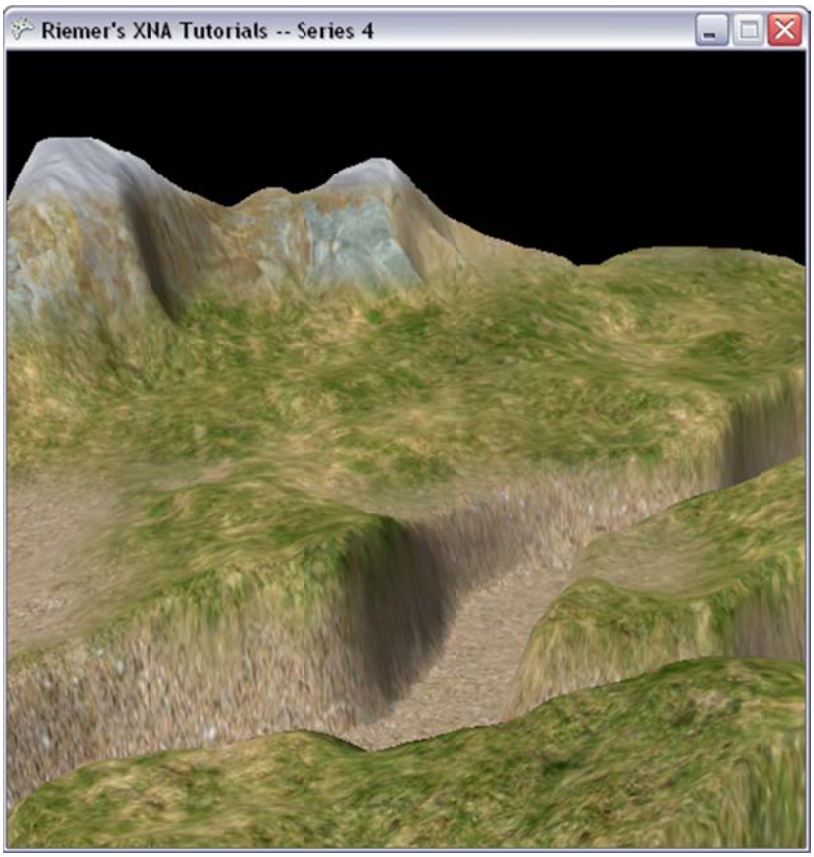



**Figure 14 - Example of Textured Terrain**

## *4.4 Design*

WeDefend will 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 is the best choice. However weDefend will be 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 will not be 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.

### *4.4.1 Scenarios*

WeDefend will present two common defense scenarios that arise in both war-time and peace. The first 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 will boot up into a scenario selection screen where the user will then have to choose to complete an asset defense or VIP defense scenario. Once the scenario type has been chosen, the user will be in the preparation phase. In the preparation phase of a scenario 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 will be able to place soldiers in any configuration they see fit to defend the highlighted area. The user may also 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 can 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 will be fired at and hopefully destroyed. By using this feature the user sets up zones which his or her units will defend. Each unit whether friend or foe, will have a certain number of hit points. These points represent the unit's life and will be visually represented as a green bar above each units head. Additionally each unit will also have an attack power attribute representing the amount of damage they are capable of inflicting on an enemy; this quantity will never be 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 will no longer be visible on the screen and will no longer be able to inflict any damage. In order to ensure that enemies do not reach the area being protected, it is allowed to have two units fields' of view overlap so that they will both fire upon a single enemy, granted that enemy is in the overlapping field of view area. Below is a figure demonstrating what a field of view looks like and what overlapping fields of view look like.

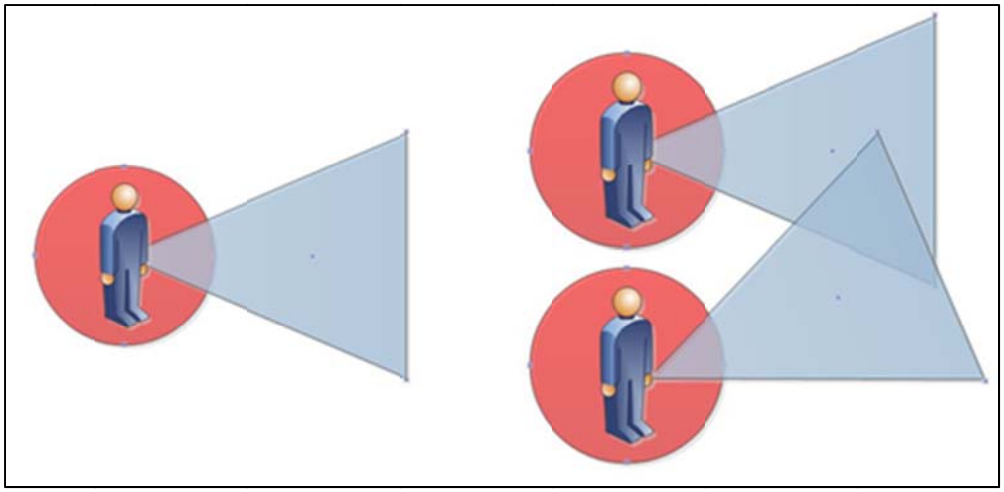


**Figure 15 – Example of Fields of View (left) and Overlapping Fields of View (right)**

The user is allowed to augment the field of view of any soldier both in the preparation and action phases. The following two commands can only be issued in the preparation phase. The user will be 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 will always be perpendicular to the line they are walking along. Units will follow their designated patrol routes until they reach the end of their route. At this time the unit will perform an about face and continue patrolling back along the route. In the case of a circular or closed patrol route the unit will never about face and instead infinitely walk the closed loop unless commanded to stop. Units can 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 is a crude way of showing that they avoid collisions in real life.

The other option available to the user during the preparation phase is squad creation. Squads allow units to have slightly larger fields of view simulating the effect of a large team being able to see and fire upon more due to their being more soldiers in the group. The drawback of grouping soldiers is that they must all face in the same direction and they cannot be put on patrol routes. However, when a squad is first created the soldiers will continue to look in whatever direction they were originally looking based on their set fields of view. After the squad has been created, if the user adjusts one squad member's field of view, the rest of the squad members will follow suit. Squads are usually used to increase firepower and cover an area where there will be a large number of enemies. To create a squad the user will highlight the area that contains all the soldiers that will be put into a particular squad. Any unit outside the selection area will not be put in the squad. The last thing the user is able to do in the preparation phase is setup blockades. A blockade is an obstruction like a concrete wall which will prevent enemies from getting

past for a certain amount of time. After a certain amount of time however the enemies will destroy the blockade and be free to move through that area again.

Once the user is finished adjusting his or her units and placing blockades, they will move into the action phase by placing the action fiducial anywhere on the screen. Once a user enters the action phase they cannot return to the preparation phase. In the action phase enemies will begin to appear from many places and attempt to get to the highlighted zone. If an enemy reaches the zone, an enemy entered counter will be incremented. Also if an enemy comes in range of a soldier, they will probably be in the soldier's field of view but regardless of whether they are inside the field of view or not they will stop and attack the soldier attempting to reduce that soldiers hit points to zero. In the scenario of defending a certain location the objective is to allow less than one-hundred enemies to reach the defense zone and survive until all enemies flee or are destroyed. Should the enemy entered counter reach one-hundred, the user will have failed the scenario. In order to not fail the scenario, the user will be able to issue only a few commands to his or her defense team. All actions that users could take in the preparation phase are not allowed save the two described below. The user will still be able to adjust the field of view of all his or her units except the ones who are patrolling. The only other thing the user can do in this phase is stop a patrolling unit. If a patrolling unit is stopped in the action phase, they will remain in the location where they were stopped. However, they're field of view will become adjustable again.

The last scenario that weDefend will incorporate is the VIP defense scenario. In this scenario the objective is to protect a certain unit called the VIP from being killed. If the VIP is killed by the enemy the user fails the scenario. This scenario is similar to defend a location scenario in that it has both a preparation and action phase. The difference mainly lies in the objective and capabilities during the preparation phase. In the preparation phase the user is still capable of doing the same things he or she could in the preparation phase of the other scenario with only one addition. During the preparation phase the user has the option of adjusting the preset path the VIP will take in order to reach the exit. The path the VIP takes will be highlighted in green and will be adjusted in the same way that a regular soldier is given a patrol route. This capability allows the user to steer the VIP towards his defense setup instead of taking a preset path. Assigning soldier's patrol routes that follows a route very similar to the VIP's will be limited to two soldiers. In the action phase all the same actions as in the last scenario can be taken. The only difference is the objective. In the action phase the VIP will only take damage from enemies and will follow the path set until he reaches the exit or is killed.

All of these actions and automated systems will be implemented by using a C# program with supporting classes. The automated logic in the action phase will be handled by a while loop that: continually checks for new user input, instantiates soldier objects in the preparation phase, instantiates VIP and enemy objects during the action phase, destroys unit objects during both phases, and updates the fields of all objects. The visual rendering will be accomplished by using XNA Game studio 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 last two functions of the

rendering will be called at the end of the application logic while loop. The game loop will end when either the victory or failure Boolean condition has been met. This will be explained in more detail in the implementation section of the design.

### *4.4.2 Gestures*

In order to carry out the actions described in the above scenarios a set of touch and fiducial gestures will be required to provide user input to the application. There are only three types of basic inputs that make up the many gestures that will control 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 vision 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, create waypoints for a patrol route, place soldiers, place barricades, load a previously saved scenario, and save a scenario currently in progress. 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 have finished in the preparation phase it will be necessary to activate the action phase to initiate the defense scenario. To do this the officer or trainee with the highest rank will possess the action fiducial. When the officer has approved the defensive set-up created by the team, he or she will place the fiducial with the pattern facing down on the table. Once the recognition system has recognized the action fiducial, it will set the preparation mode Boolean to false which will allow the main game loop to begin running. Once the action phase is in progress the action fiducial will be totally inert and will be ignored by weDefend. The place soldier and place barricade fiducials perform similar actions. However, they have their own unique patterns and thus their own unique fiducial. These two fiducials will only work during the preparation phase and will be used to "stamp" units onto the battlefield. When either of the fiducials for unit creation is placed on, Plank the gesture recognition unit will report a screen coordinate relating where that fiducial was placed as well as what type of unit that fiducial represents. With that information, weDefend will instantiate 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 will be able to easily place units onto the battlefield without having to access any kind of menu. Multiple copies of the soldier and barricade fiducial will exist 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 will not be a problem because each fiducial will have a unique identification number to differentiate it from the others. This allows weDefend to only be concerned with the data the gesture recognition system reports to it about a fiducial. These fiducials will also become inert once the action phase has begun because placement of units is not allowed in the action phase.

The waypoint fiducial will be used to create a patrol route in the preparation phase for a soldier. Creating waypoints via fiducials is one of the only gestures that require both a finger touch and a fiducial. In order to create a patrol using fiducials 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 fiducial to stamp out waypoints similar to how he or she would stamp out units. In order to make clear to the user that waypoints are being created, yellow dots connected by yellow lines that lead back to the unit will be drawn to give a visual cue of the path the soldier will patrol. Creating patrol routes using fiducials is a quick and easy way for a user to create a non-loop patrol. Similar to the other actions discussed thus far, the waypoint fiducial will become useless once the action phase has started. Additionally, the waypoint fiducial will not work if a soldier is in a squad because squads cannot be put on patrol routes.

The last two fiducial types, the save and load fiducials, are not integral to the function of weDefend but would be a valuable gesture to any application. The save fiducial can be used at any time by a user to save the state of a defense setup in the preparation phase or the state of a current scenario that is in the action phase. In order to accomplish this, the fiducial will be placed on the surface and then weDefend will record in a text file all the fields associated with each object as well as the identification number of the fiducial being used as a save function. Having this information will allow weDefend to restart from where it was by instantiating the same objects with their respective data. Once a fiducial has been used as a save fiducial, it can be used again later as a load fiducial. When a load fiducial is placed on the surface, the fiducial's identification number will be checked against the identification numbers in the save files to see if the specific fiducial corresponds to a load fiducial. If the fiducial's identification number matches that of any of the ones stored, weDefend will load all the object states and scenario states in that save file, effectively loading a previous scenario or defensive preparation.

The remaining gestures use touch events to both setup and command units in both phases. The two gestures are a drag, which will be used for moving units in the preparation phase and creating squad lassos, and a pin-drag used to adjust the field of view and create patrol routes. A drag is 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 is moving an existing unit to a new location. In order to do this, the user must place a finger on a unit. The unit will then be highlighted in red as a cue that the unit is active. Once a unit is active, the unit can be dragged across the battlefield and then dropped into place by removing the finger. This command is only valid in the preparation phase and will be 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 can also be used to group soldiers into squads. To group soldiers into a squad 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 circle around them. Once the finger is lifted off the surface, all the units that were within the "lasso" will be

grouped and they will be highlighted with a squad color to show that they are all now part of the same squad. The figure below illustrates how units can be grouped into squads.

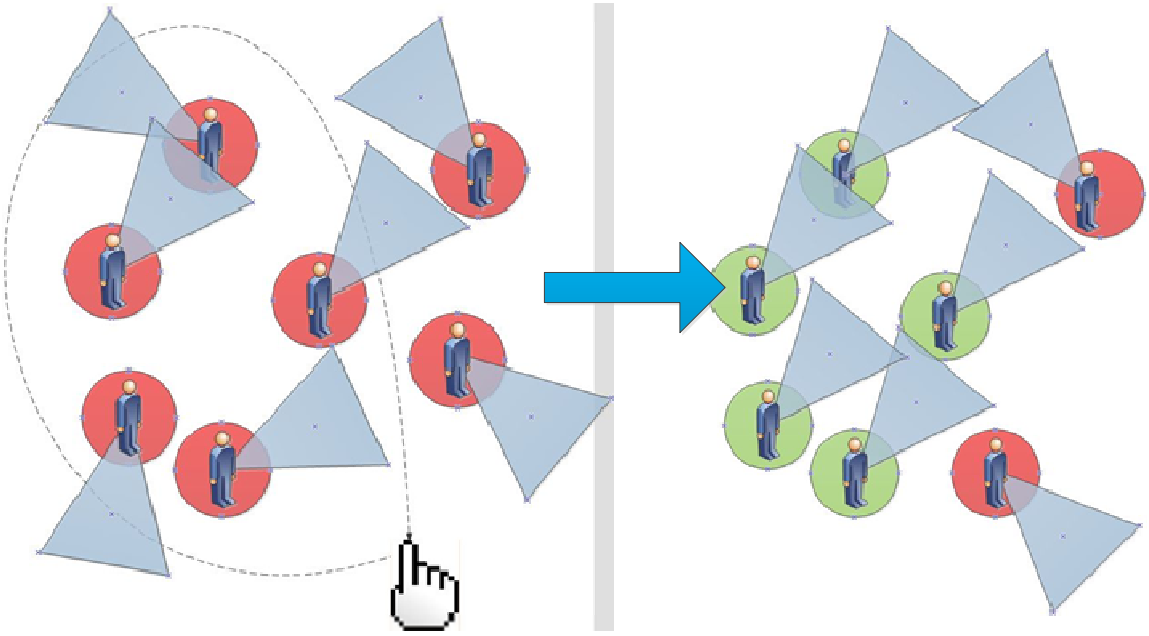


**Figure 16 - Example of Lasso Select Gesture**

The commands that involve 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 is 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 is 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 will be established using the distance in pixels from the unit as the denominator of the ratio which will affect how the angle will be calculated. The field of view angle is inversely proportional to the field of view distance. If the distance of the field of view is 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 can engage an enemy.

The last command, creating a patrol, also uses a pin-drag and is similar to adjusting field of view but is 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 within the active radius of a unit. The active radius of a unit will be denoted by a red circle that surrounds the unit when a finger has been placed. Once both fingers are inside the active radius the second finger should be dragged away from the unit in

order to specify the route which the soldier should patrol along. The route will be denoted by a yellow line that is drawn as the user drags their finger. Once the fingers are lifted off of Planck the patrol route will be created and the unit will begin patrolling. The last and final command that a user can give is to stop a unit from patrolling. To do this the user need only tap on a unit that is patrolling and the unit will 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.

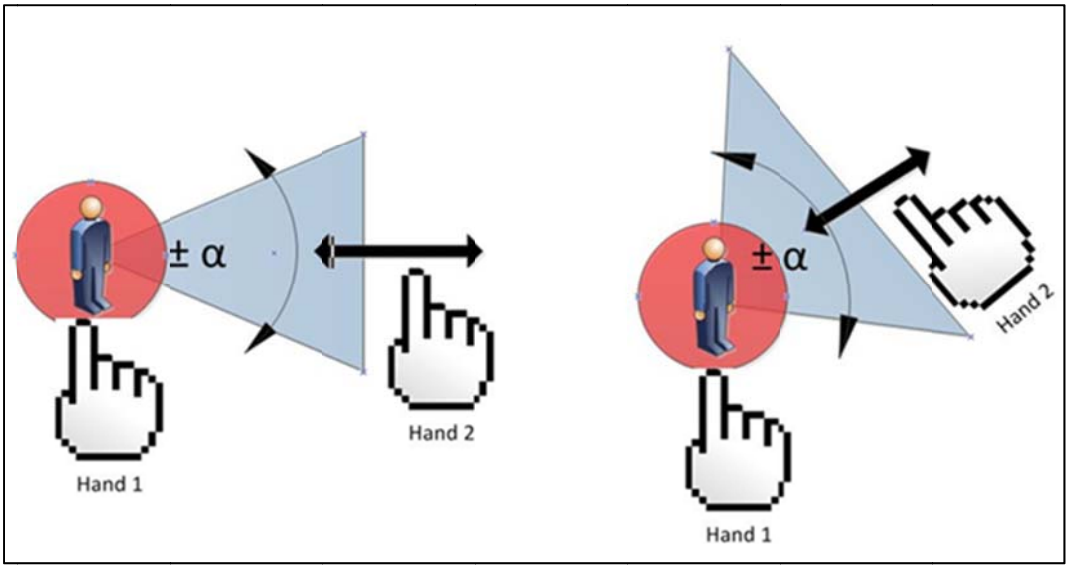



**Figure 17 - Example of Pinch Gesture for adjusting Field of View**

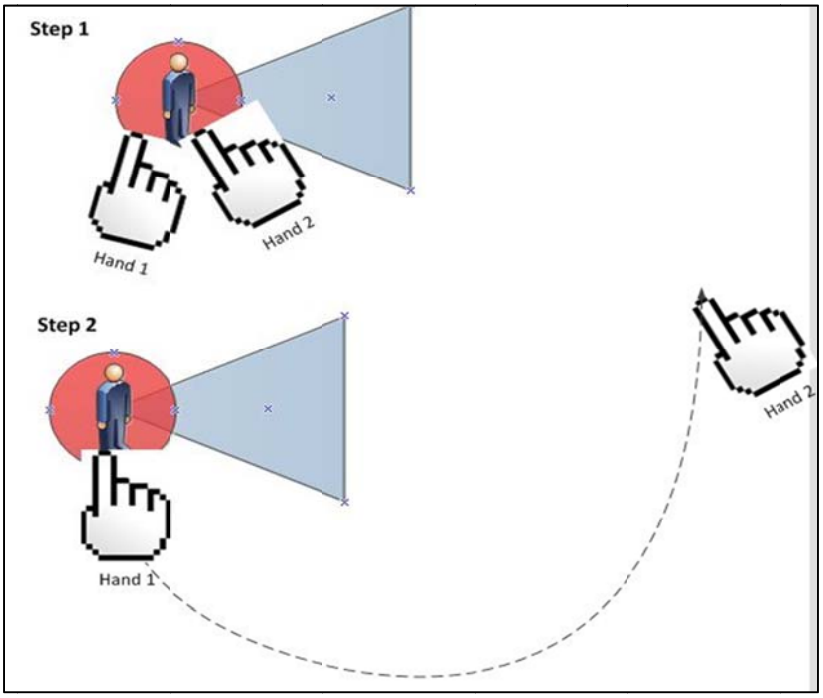



**Figure 18 - Example of Patrol Gesture**

### *4.4.3 Graphics Structure*

XNA Game Studio will be responsible for rendering the required graphics, executing and processing scenario logic, providing sound effects, 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 pogram.cs and game1.cs. The program.cs handles creating the entry point or window that the application will use. The game1.cs file which will be renamed weDefend.cs 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 weDefend.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 Blob 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 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 recognition class. The update method should first access the gesture linked list and associate gestures in the list with the proper soldier objects. A separate method called Decode Gestures 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 check the victory and failure conditions to see if the scenario is over. These conditions are VIP dead Boolean, asset overrun Boolean, and a check to see if one-hundred or more enemies have slipped through the defensive line. Update will then check the soldier and insurgent arrays to see if the game is over based on whether or not one side has been routed. 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, VIP, blockade, and insurgent objects. In order to do this each object type will have a render method built into to 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 is made up of five classes. Four of these classes are objects which have a role in the scenario, while the other is the main class. The main class which will be renamed from game1 to weDefend has eight fields and ten methods. This class is responsible for holding the majority of the logic and running the two possible scenarios 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 will be severely overhauled to incorporate Plank's novel input. The update method will run in a two parameter switch case which decides whether weDefend is in the preparation phase or the action phase. It will then use the decode gestures method to link gestures to soldiers. Then the update units state method will adjust all the fields of the existing objects given they have an input gesture linked to them. Once these two methods are done the create and destroy unit methods will be called to delete or create any new soldiers or insurgents. At the end of the update method the victory and failure parameters will be checked. 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 Plank. The decode gestures method will access a linked list that is part of the gesture recognition class. This linked list will contain many gesture objects which contain the identification number, screen coordinates, and gesture type. With this data an algorithm will pair gestures with objects and assign the gesture identification number to the gesture link ID field which each usable object has. 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. The update unit state method will go through each instance of insurgents and soldiers and update their fields based on input, for example if a field of view was changed, and based on whether or not enemies came into contact with soldiers or the VIP. Create and destroy units methods will instantiate and or remove objects that need to be created or have died. The last method is actually the constructor of the weDefend class which initializes the content pipeline and identifies the graphics device being used.

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 preparation mode Boolean which tells the update function whether weDefend is in the preparation or action phase. The VIP dead Boolean which is a failure condition of the VIP scenario and lets the update function knows if the VIP has been

killed. The asset overrun Boolean again lets the update function tell if a defense scenario has ended in failure. And lastly the enemies infiltrated integer will be checked in the update function to see if one-hundred or more enemies have infiltrated the area to be defended. The other two fields are linked lists called soldiers and insurgents. These two linked lists will contain each instance of soldiers and insurgents. It will be 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 four classes are the units that take part in the different weDefend scenarios. The four units are soldiers, insurgents, blockades, and the VIP. The VIP and blockade classes are sub classes of the soldier class with a few differences. Both the VIP and blockade only differ from their parent class in that they have a different sprite image and amount of hit points associated with them. The VIP will have one-hundred and twenty hit points, while the blockade will have four-hundred hit points.

The soldier class will contain many important methods and fields which the VIP and blockade will inherit. The soldier class has thirteen fields which allow all game logic and graphics to be updated. The soldier has a hit point integer which represents his health. Every soldier has two-hundred hit points. Another integer the attack power is a constant that never changes for any unit. The soldier will reduce an enemy's hit points by sixty every time he attacks an enemy. The field of view distance defines how far from the unit the field of view must be drawn. The field of view angle specifies how large the field of view is. 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 will be a linked list of point objects that contain the waypoints of where a patrolling soldier needs to go to next. This data structure will have a start point and endpoint to denote the beginning and end of a patrol route. The gesture link ID is a linked list of gesture IDs that belong to a specific soldier. The unit model is a XNA texture2D object or .x file object that contains the image or model for the soldier. The enemy target ID is the enemy that the soldier is firing at. A soldier will fire at the first enemy that enters his field of view and will 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. If two enemies are in his field of view when he switches than the unit IDs of the soldier will be compared and the smaller ID will be the one that gets attacked. The gesture exists 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 consist of set and get functions for the setting of fields explained above. The get methods will allow the weDefend methods to look into the state that each soldier is in so that they can be updated by using the soldier set methods. The only unique method is the render method. The render method uses all of the field data that a soldier contains in order to properly set up the drawing parameters for an object such as the location of the unit, the image that makes up its sprite and so on. This method

will make it so that the draw method does not have to look into all the data in each object but rather call the render function of every object that still exists.

The last class in weDefend is the Insurgent class. The insurgent class has six fields and eight methods. Every Instance of insurgent created corresponds to an enemy that is drawn on the screen. Insurgents have some basic data that pertain to them. They all have one-hundred and eighty hit points. The hit point field holds this number which can be decremented as the scenario progresses; an integer will hold this value. The hit points can be updated by using the get and set hitpoints methods. The update unit state method in the weDefend class will have access to the hitpoints of all units via the get and set hit points methods. The attack power integer is a constant which cannot be changed for any insurgent. The insurgent is capable of inflicting forty hit points worth of damage every time they attack an enemy. Any time a unit is attacked by an insurgent the update unit state method will be able to access the attack power of the unit that is attacking by using the get attack power method. In this way hit points can be subtracted from the unit being attacked. In order for an insurgent to attack a soldier or VIP the distance from soldier field must fall below a certain threshold. This variable will be constantly updated to see if an enemy is close enough to a soldier to attack. Enemies will always attack the closest opposing unit to them which will be calculated using the screen coordinate field of the enemy and the screen coordinate field of the soldier that is closest to them. If the closest soldier's distance falls under the threshold then the insurgent will begin inflicting forty hit points of damage for as long as they are in range. The screen coordinate and unit ID fields will also be used to update the positions of the enemies as they move through time about their set paths. This movement route parameter will define how each insurgent that comes into the scenario will move through the map. Each map will have a set number of areas that insurgents can spawn from and each of these areas will have their own route which all insurgents that spawn in that area will follow. This means that the route each insurgent takes will be based on their starting location and only a set number of routes will exist that the insurgent can take. One of these routes will be assigned to each instance of insurgent's movement route field. The set screen coordinate will be used to continually update the insurgent's position based on the movement route. The last method an insurgent instance uses is its render function. The render function will use the attributes such as screen coordinates and hit points to update the insurgent's appearance on the screen. The insurgent's unit model will be used to draw a visual representation of an enemy on the screen and all the draw method in the weDefend class will do is loop through all the insurgents that are present in the existing Insurgents linked list and call each of their respective render methods which contains all the updated attributes of each instance of insurgent.

# 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.

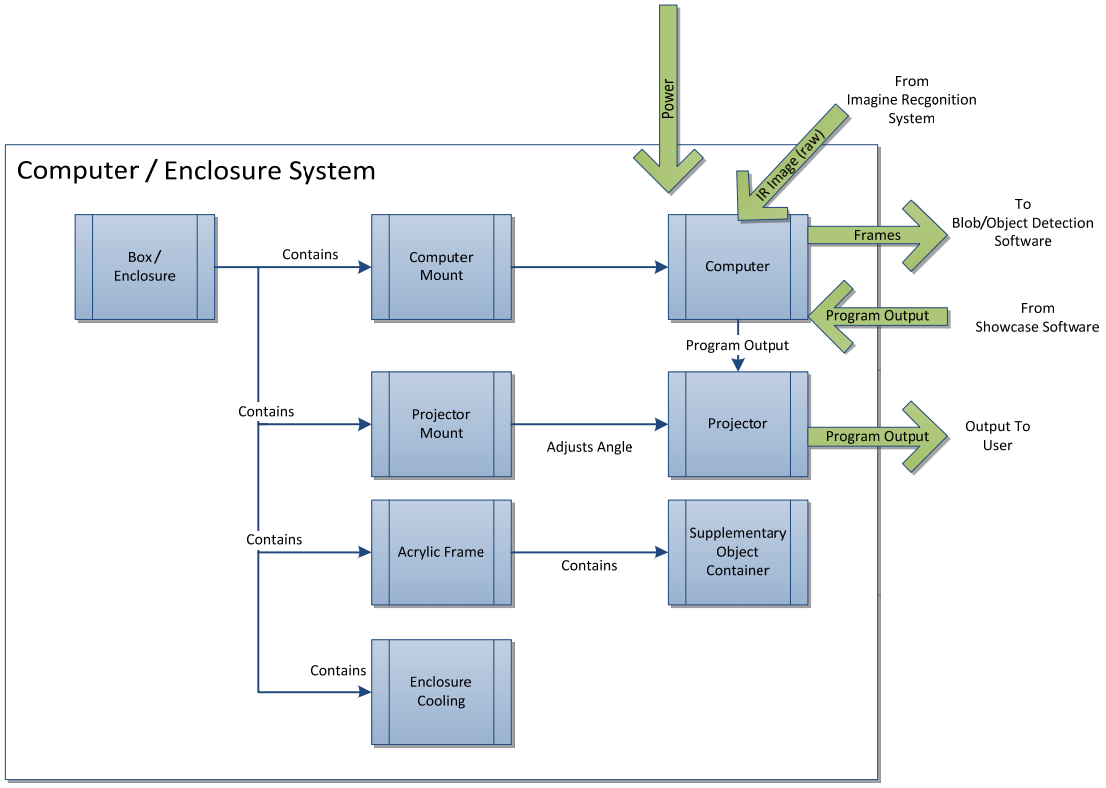



**Figure 19 – Top Level Diagram of Computational Container System**

## *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 will be 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 should be implemented into the design of Planck. A high grade veneer hardwood would be acceptable. 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. Hardwoods that would be visually appealing for our enclosure could be oak, walnut, maple, or cherry.

The enclosure should be tall enough for our tallest user and short enough for our shortest user. Current estimates is that the enclosure should sit about three feet tall. The footprint will be as big as needed to accommodate the display screen. The enclosure should feature a mounting system for the mirrors and projectors so they can rotate on one axis for easy configuration of the system.

The enclosure should have a door for access to the hardware mounted inside. This includes the computer, fans, camera, and possibly a projector. The door should be big enough so a grown adult can easily stick both his arms and part of his shoulder through the doorway. The top of the enclosure should also feature a designated spot for the fiducial objects to rest. This can simply be a walled-off open area bordering the viewing screen.

## *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.

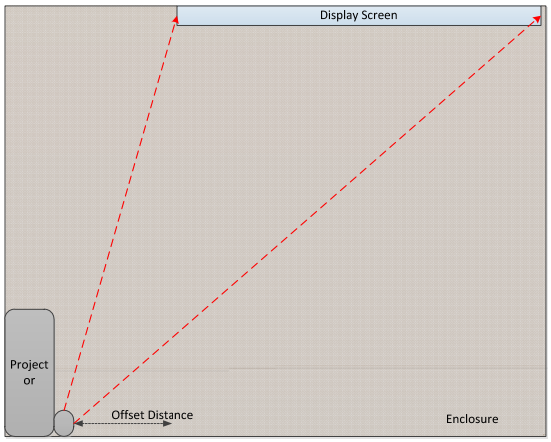



**Figure 20 - Mounting option of Short Throw Projector with Offset greater than 100%**

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:

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

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.

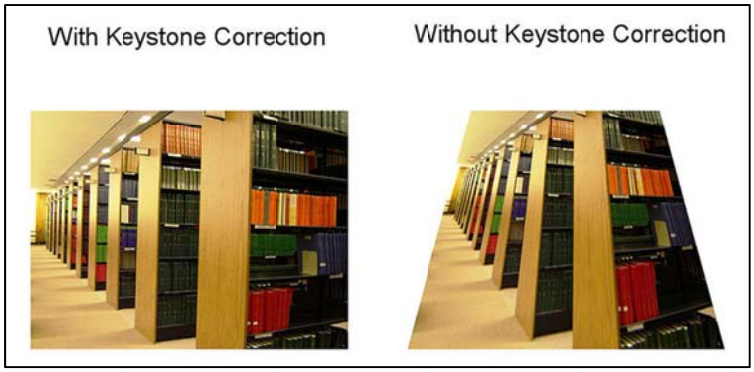



**Figure 21 - Keystoning effect (Reprinted under Creative Commons Attribution-ShareAlike 3.0 License)**

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 replaced 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 minimalized.

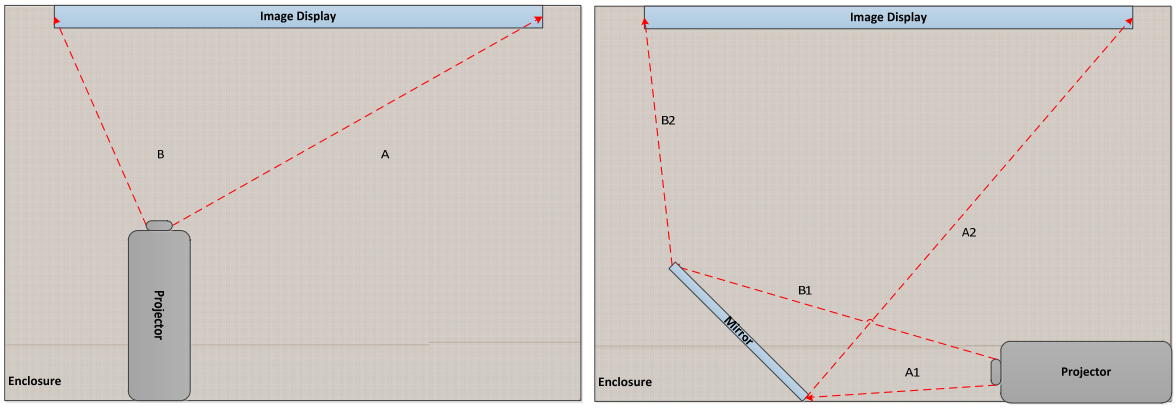



**Figure 22 - Minimalizing Single Mirror Keystoning Effect**

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.

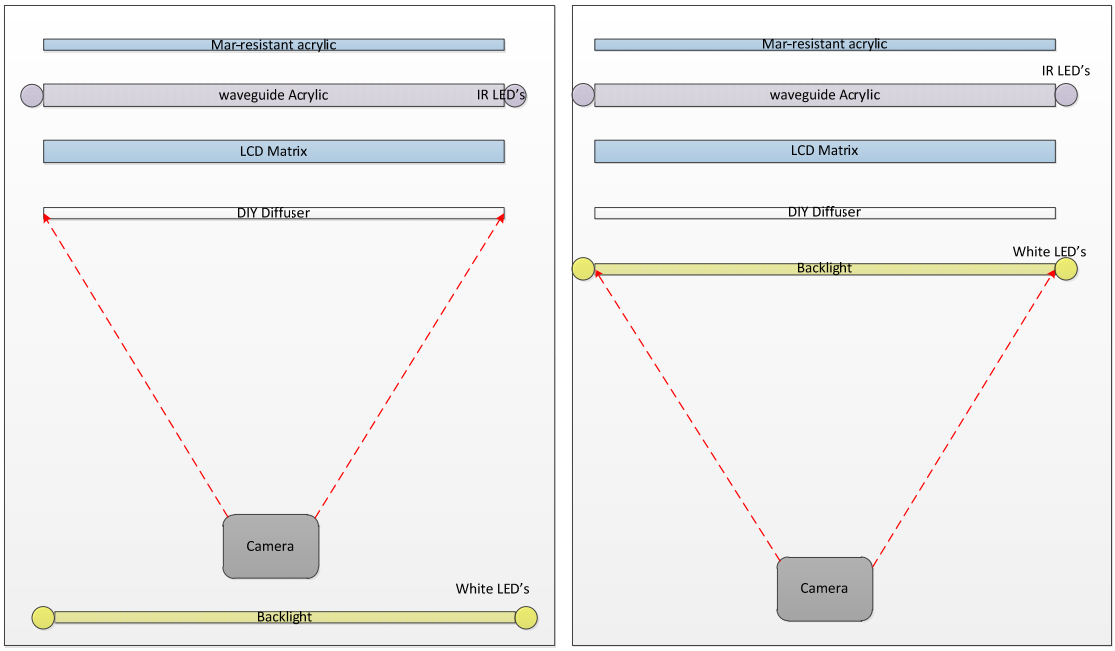



**Figure 23 - LCD Layering**

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 minimalize 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 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 7.

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 7 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 7. 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 7**

*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.

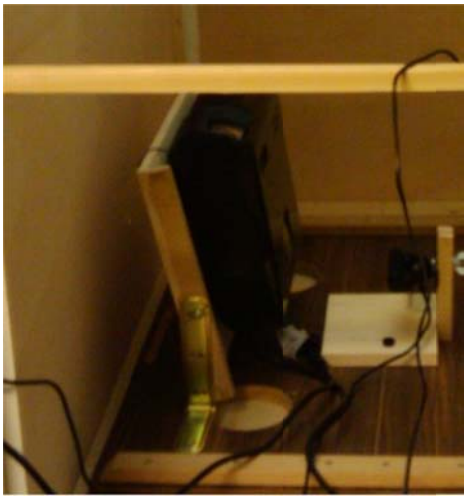


**Figure 24 - Projector Swivel Platform (Reprint pending approval)**

### *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 8 below.

**Table 8**

*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 8, 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 would be to use what is already within the device. 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 vole 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 3$^{rd}$ 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 4$^{th}$ 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 our 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 inhibit 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 MSP430F2012 or similar chip will 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).

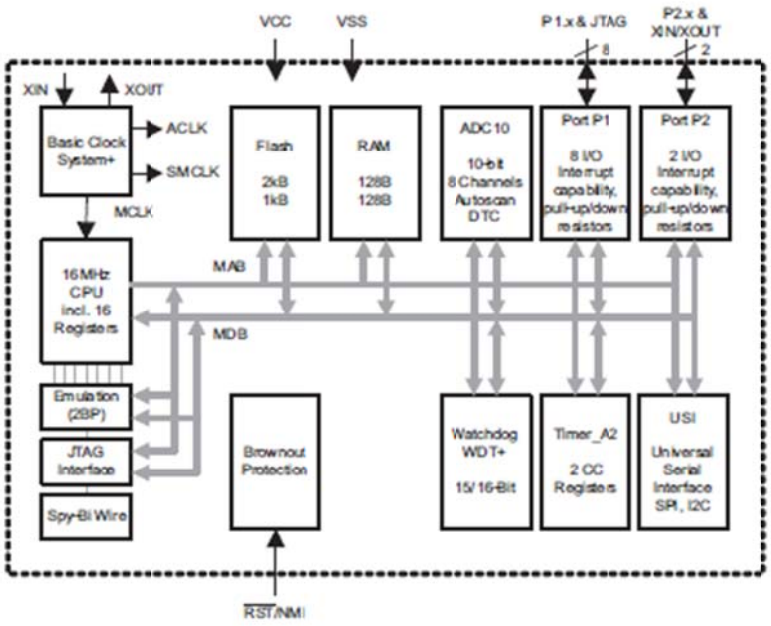



**Figure 26 - Functional Block Diagram of MSP430F2012 (Reprint pending approval)**

## *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 will be 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 would be 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 9**

*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).

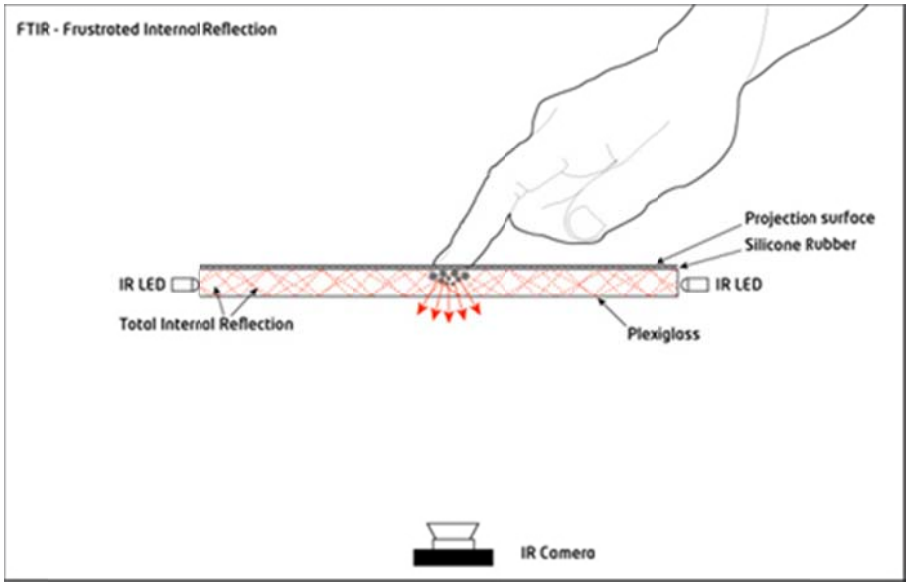



**Figure 27 - Illustration of Frustrated Total Internal Reflection (Reprint pending approval)**

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. Its 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).

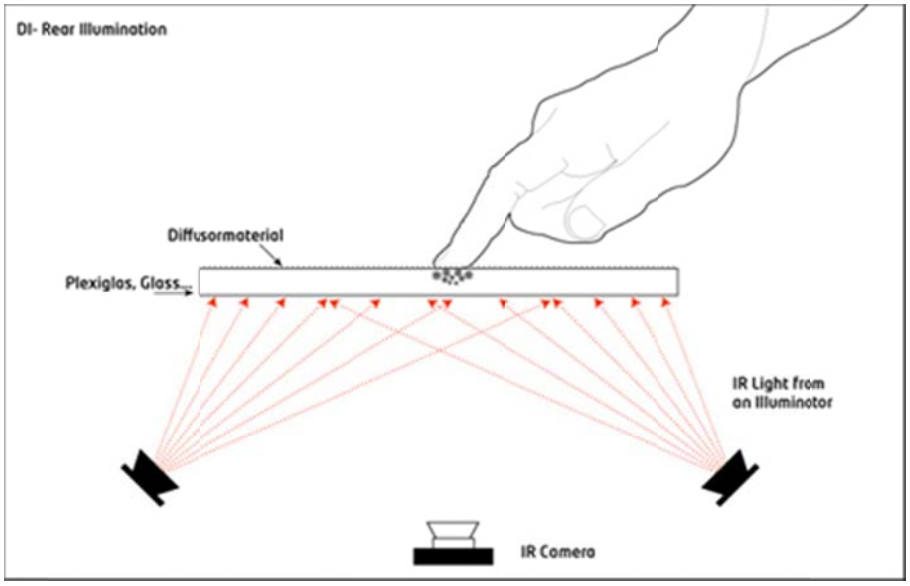



**Figure 28 - Illustration of Diffused Surface Illumination (Reprint pending approval)**

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).

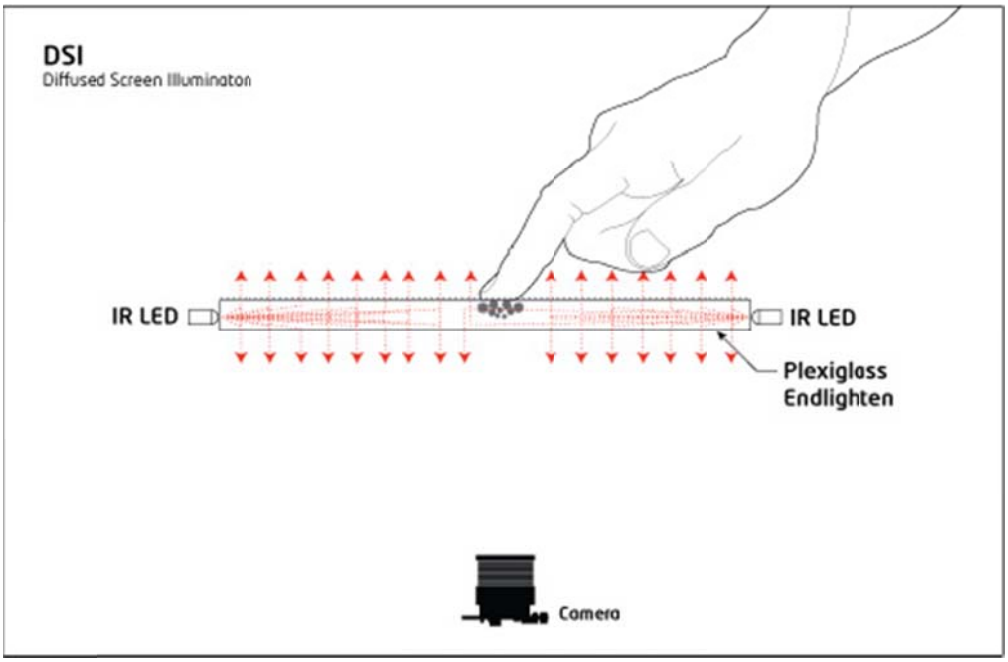



**Figure 29 - Illustration of Diffused Surface Illumination (Reprint pending approval)**

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 Imagining)*

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.

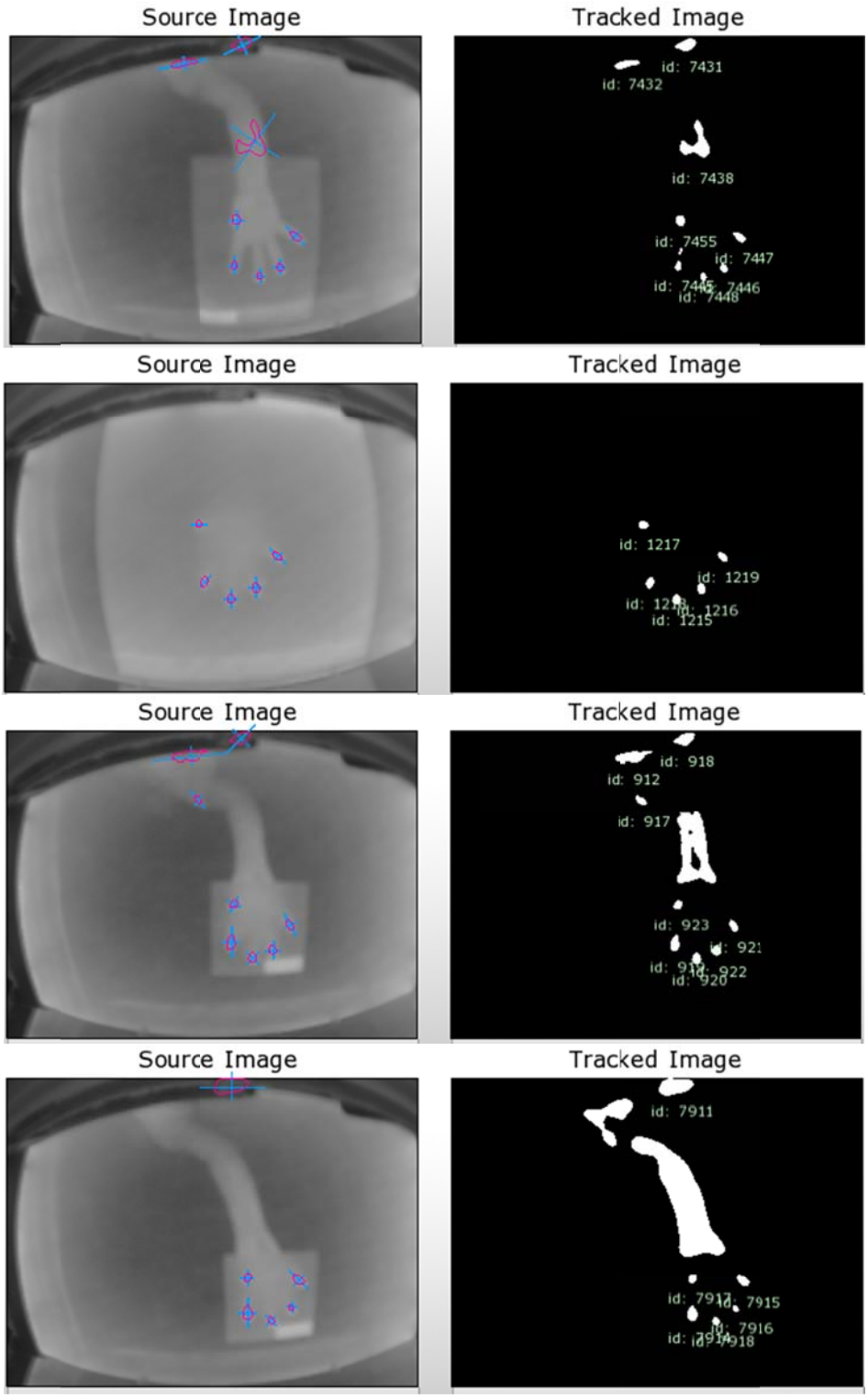



**Figure 30 - Inclusion of Various Diffuser Materials on DSI Table. From top to bottom, 0D002 Colorless, 7D512 Light Grey, 7D006 Grey, 7D513 Dark Grey(Ramseyer) (Reprint pending approval)**

## *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 censor season 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 censor. 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 "loss-y". This means that quality would be lost between what the camera's censor 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 very and are not firmly known. The cutoff frequency may vary from one disk to another or depend on the development process of the film.

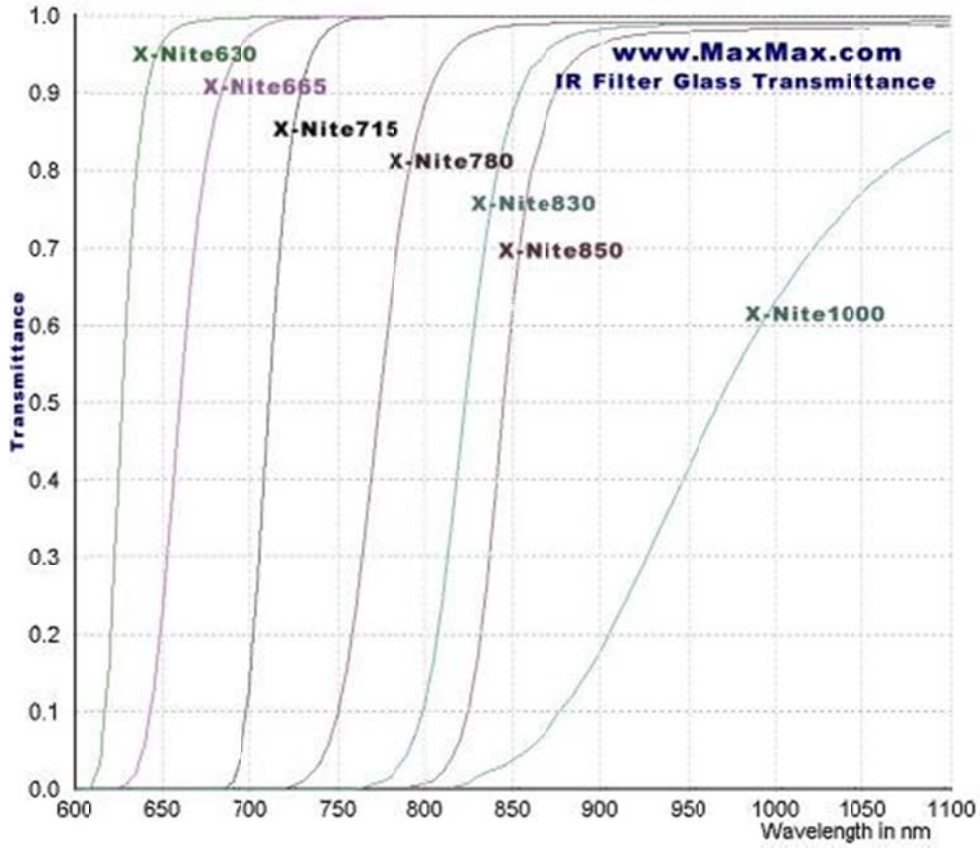



**Figure 31 - Comparison of Optical Low Pass Filters offered by one manufacturer (Reprint pending approval)**

A more reliable method of filtering would be to purchase an optical low pass filter. These vary in price, and can vary from around $50 for a basic optical filter to a few hundred dollars for a filter designed to be used with cameras for photography. These filters are rated to a specific cutoff wavelength, much like low pass filters would be in electrical circuits. Also, much like in electronics, the cutoff frequency is not the point where the amplitude begins to decrease, but the 6dB point. Because of this, the cutoff frequency should be sufficiently above the required frequency. Because the cutoff is given in wavelength, he inverse, then be a higher wavelength.

## *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 32 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.

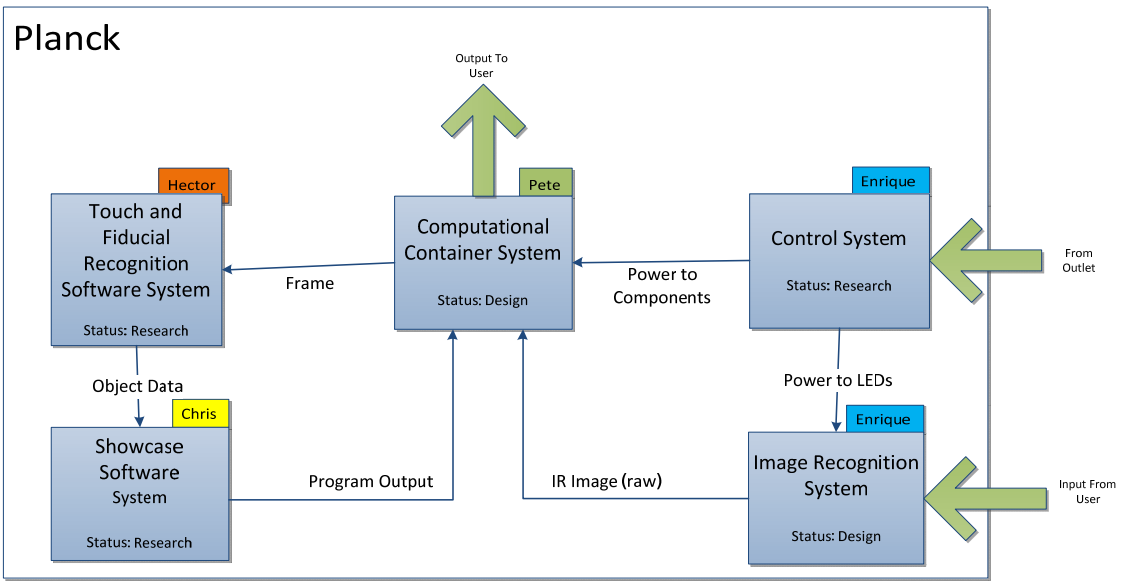



**Figure 32 - Top level block diagram**

## *8.1 Touch and Fiducial Recognition Software System*

The figure below illustrates

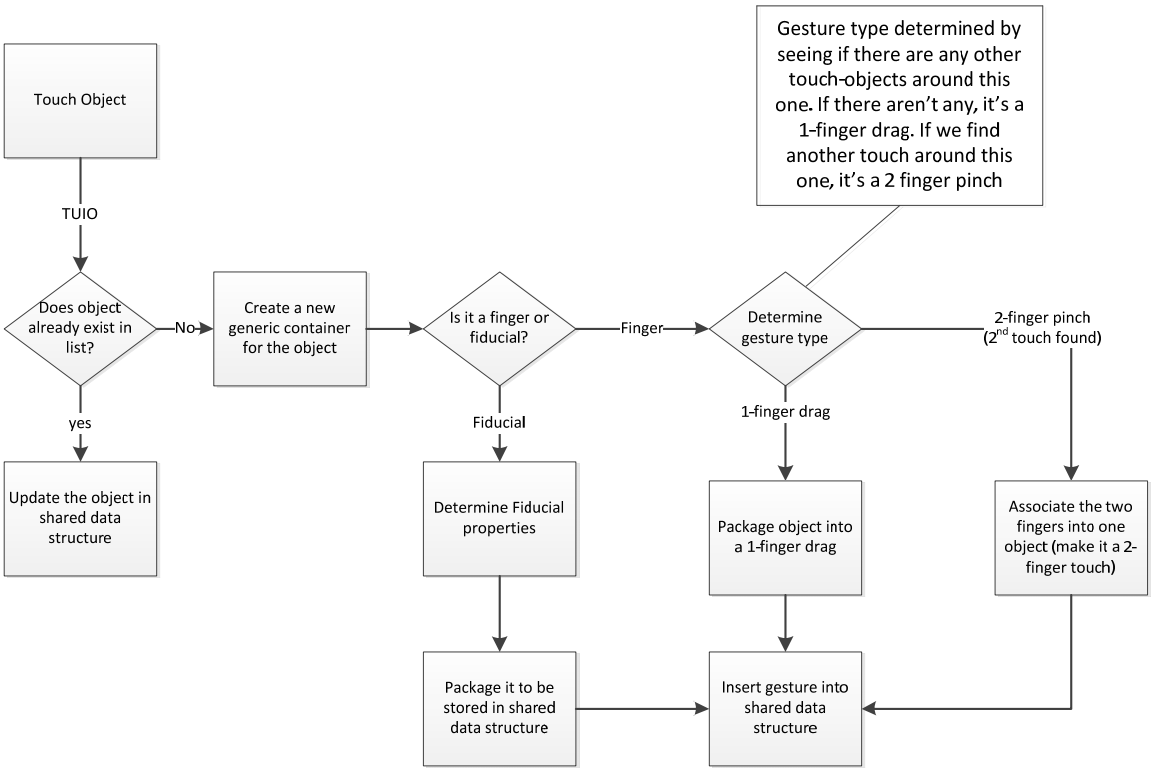



**Figure 33 - Block Diagram of Touch and Fiducial Recognition Software**

The UML Sequence Diagram 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 below.

The first step is the current user of the simulation placing either their finer or a fiducial onto the screen. The vision library CCV 1.5 will recognize these touch and objects as a generic Touch object or a generic Fiducial object and throw both into their respective container. From here the objects will be sent over TUIO to the gesture framework. Once the gesture framework receives they generic options it will run the method isFid() on each generic option, passing them as a parameter. This function, isFid(), determines whether or not the generic object is a touch object or a Fiducial object. If it is a Touch object, the function will return a Boolean false. If it's a fiducial object, the function will return a Boolean True.

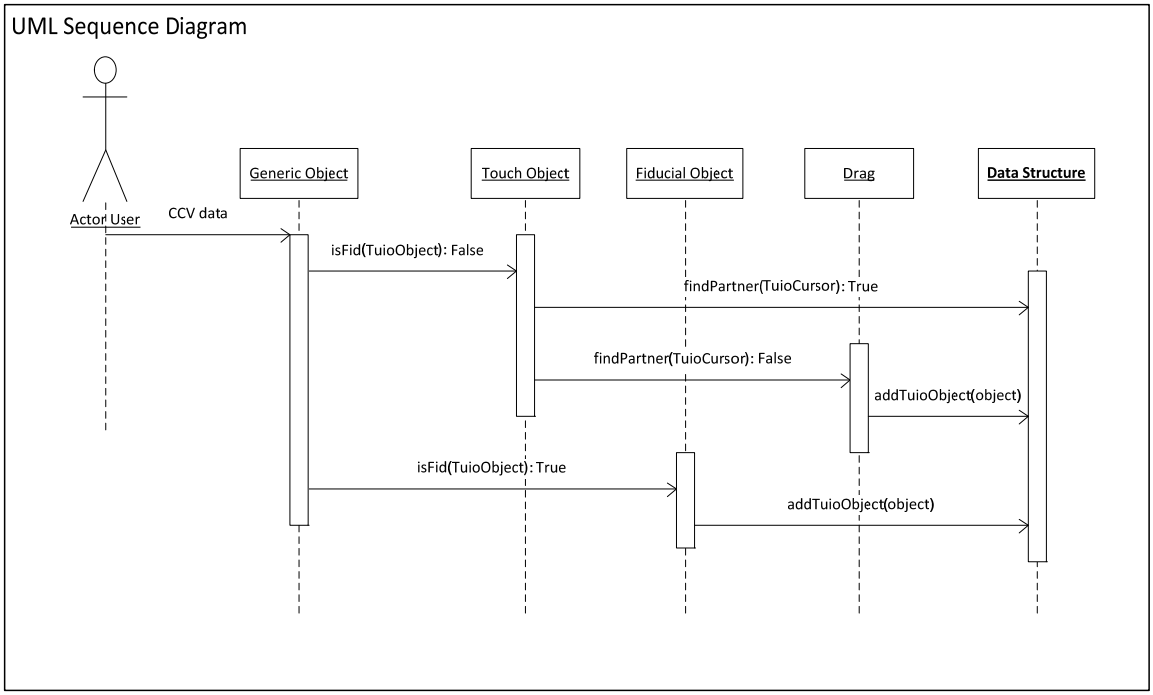



**Figure 34 - Sequence Diagram of Touch and Fiducial Recognition Software**

If the generic object is recognized as fiducial object, the gesture framework will add it to the fiducial object container. No other calculations have to be done; the fiducial object can be immediately added to the Data Structure using the addTuioObject() method. If the generic option was recognized as a Touch object, the object will be packaged into the Touch object container. More calculations still have to be done on the Touch object. The function findPartner() will be run on each individual touch, passing both Touch objects as parameters. The function findPartner() determines if the Touch object is with a certain distance of another Touch object. If the object does not exceed the maximum threshold for allowed distance, then the function returns a Boolean True. If this happens, both the new Touch object and the Touch object found to be within the threshold distance are linked as a group, and the new Touch object is added to the shared data structure. If the function returns false, then no Touch object was found and the Touch object will be packaged inside a new container, the Drag object. The Drag object will enter the shared data structure using the method call addTuioObject().

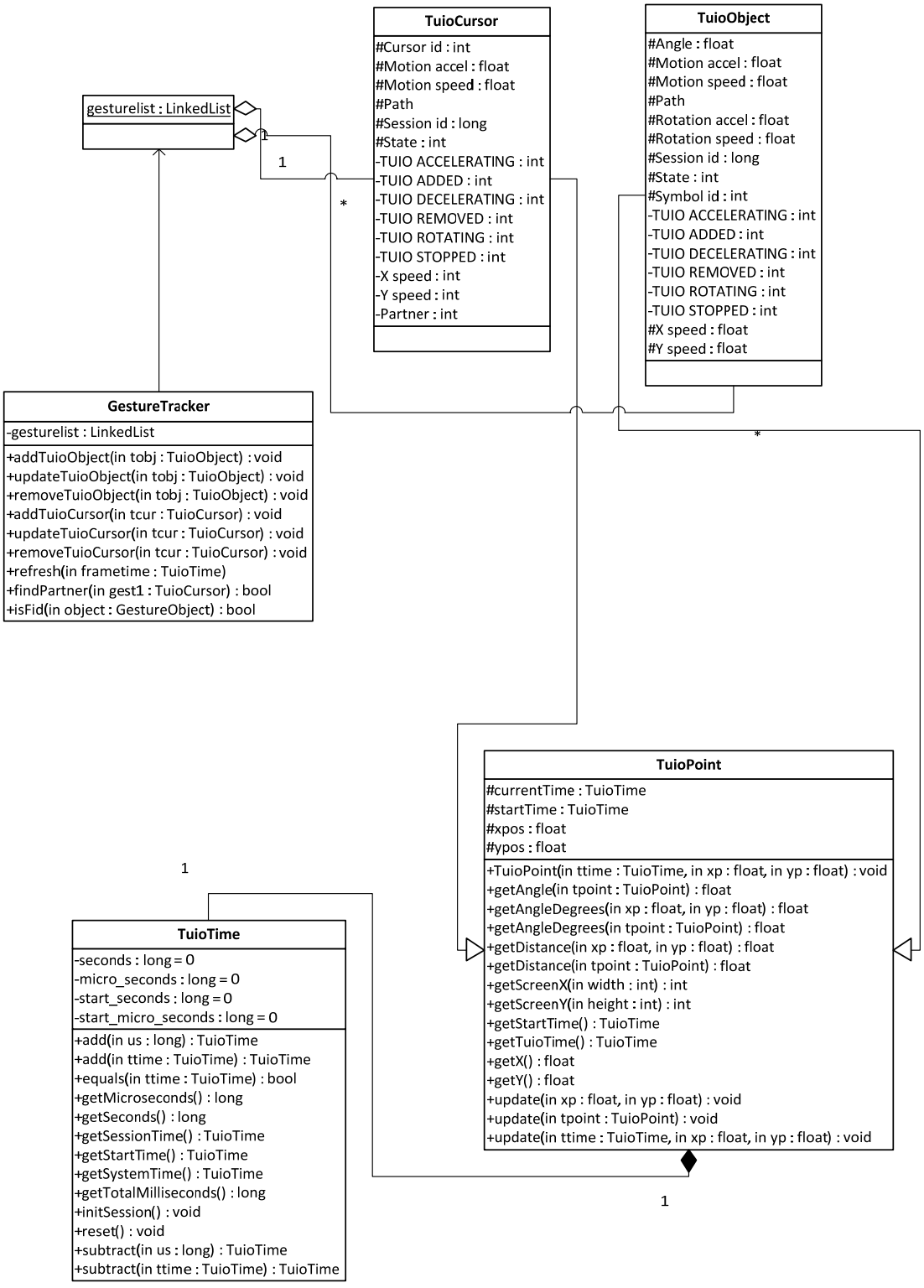



**Figure 35 - Touch and Fiducial Recognition Software System Class Diagram**

## *8.2 Showcase Software (weDefend)*

Figure 36 below 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 will parse them into gestures. The Application will include the game logic function and will call the drawing function. It will also contain 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.

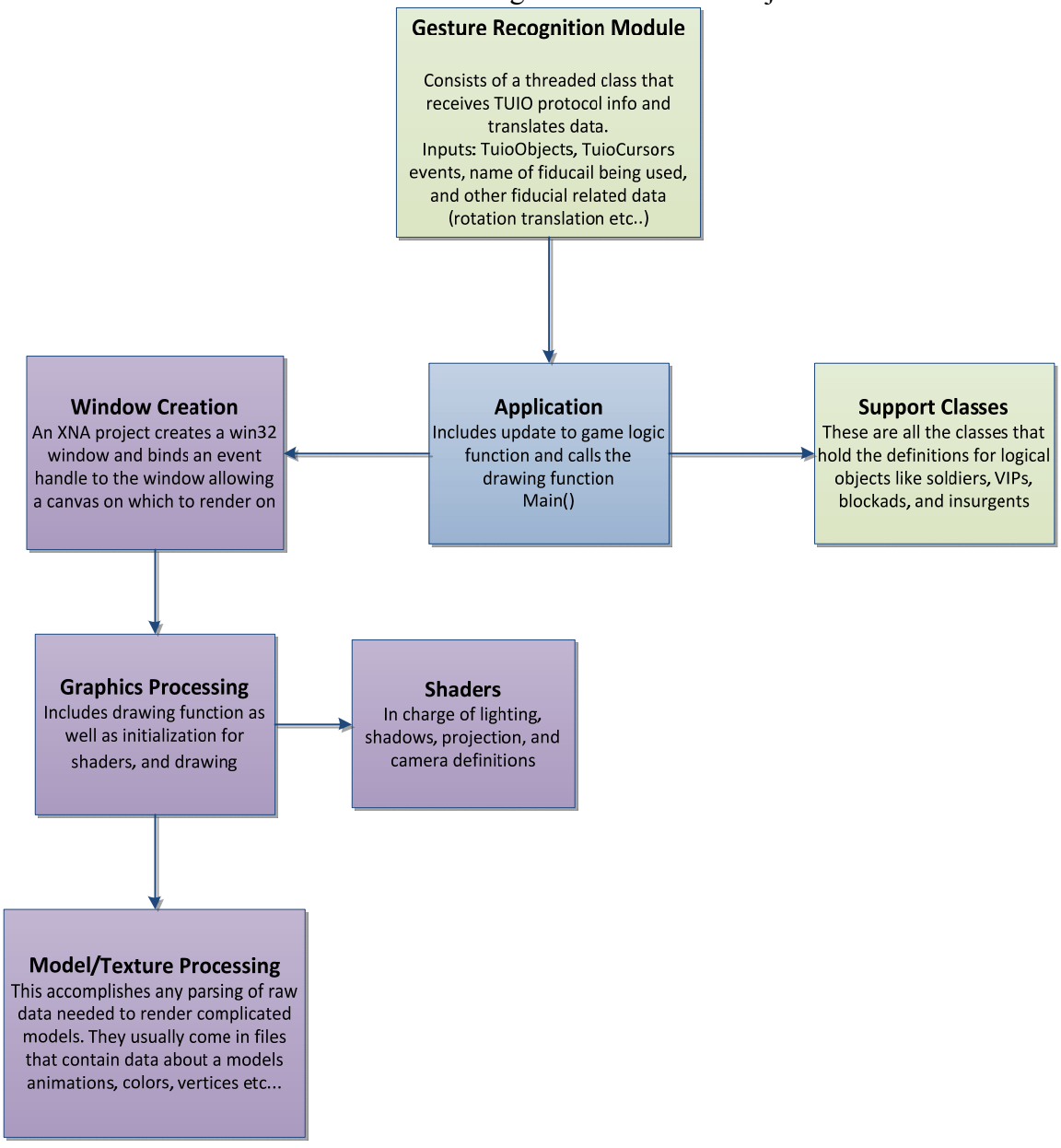



**Figure 36 - Block Diagram for weDefend**

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. Project Planck deemed the Use-Case diagram to be useful to get a better understanding of the many choices the user has when interacting with weDefend. 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. This helps the developing team understand the flow of the application and makes holes in the design easier to realize. Another benefit was the realization of the need for a State diagram after the completion of the Use-Case diagram.
The game will initially start to a black screen after Windows boot-up, with no interaction required by the user. 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 four options the user can take. One option the user has available is to choose between the defense and VIP escort scenarios. The second, the user can stamp his first unit anywhere on the surface using the Stamp Unit Fiducial object. The third, the user can stamp the Save State fiducial anywhere on surface and the state of the game will be saved away to that particular object that the current user shows ownership for. This will be his personal saved game. The fourth and last 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 three 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 four 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. Last, he can stamp a Patrol fiducial object to set a patrol route for the unit. The user can then stamp the Save State fiducial or he can return to select any other option available in Preparation Mode. 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. If the user decided to stamp the Save State fiducial, then he would return to Preparation Mode and have all the available options already mentioned in Preparation Mode. If the user's executed the FOV gesture, Set Patrol gesture, Lasso Group pin gesture, or Stamp a Patrol, the user has the ability to stamp the Save State fiducial to save the simulation. In fact, the user can stamp the Save State fiducial anywhere on surface, and at any time in Preparation mode and the simulation will save the current state.

**Use-Case Diagram**

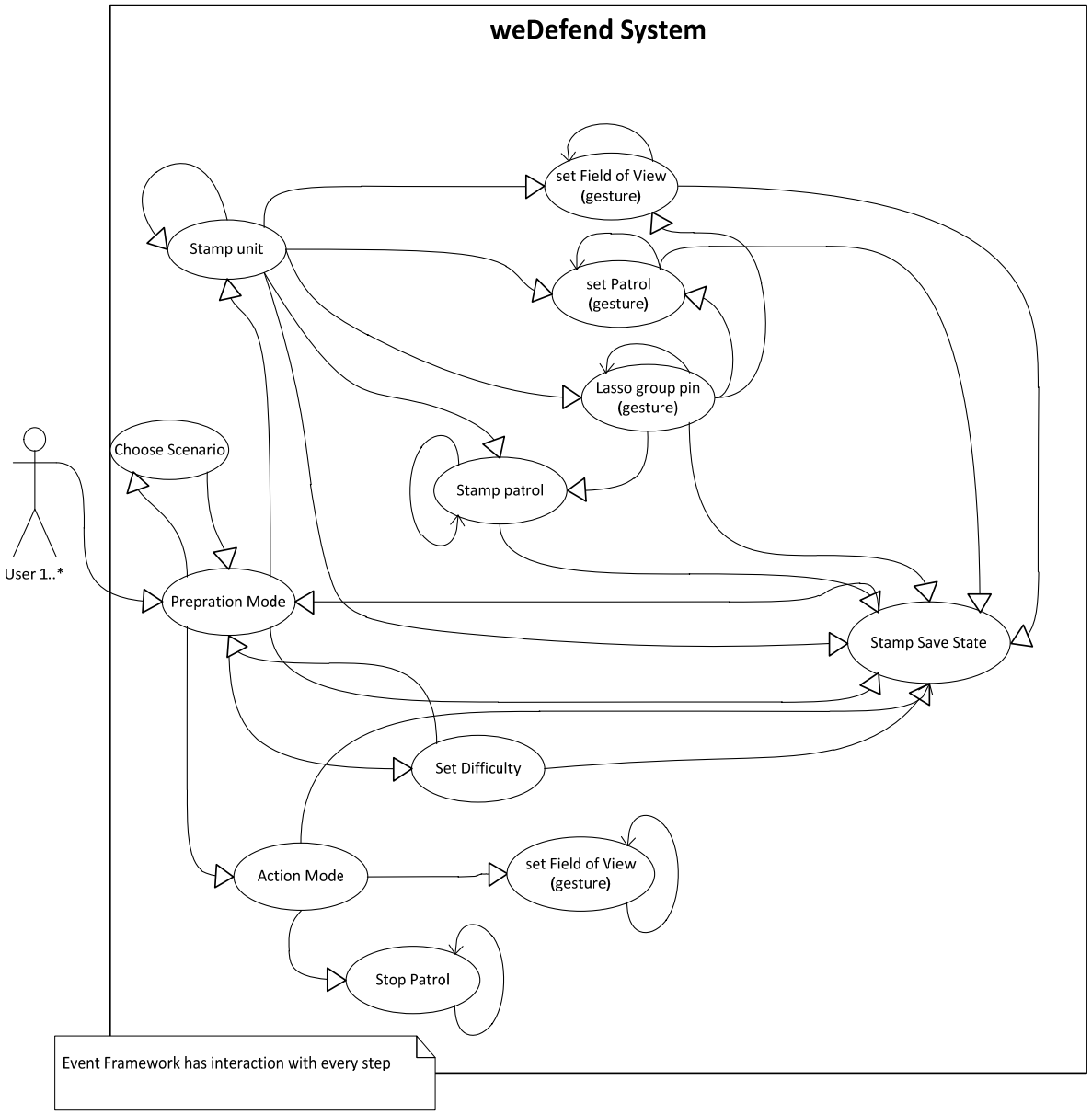



**Figure 37 - weDefend Use Case**

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. At the end of Windows boot-up, weDefend will start automatically. This is the initial state of weDefend. The simulation will enter the Preparation mode. No other use of gestures or fiducial objects will have any effect on the surface. 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. The user cannot, however, return to the start of the simulation.

The user only has one option available to him if he wants to change states. The user must stamp the Action Mode gesture 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.

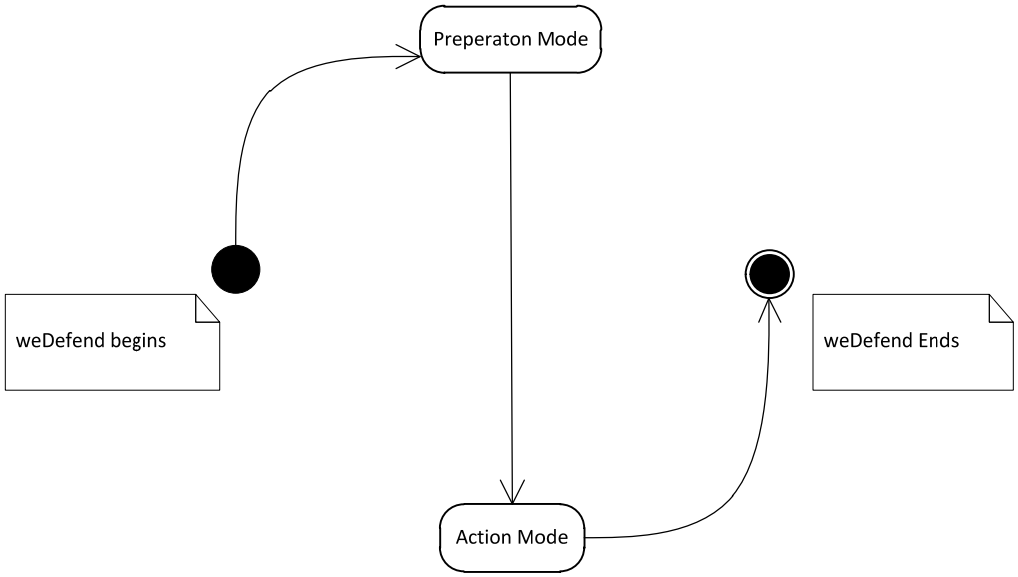



**Figure 38 - weDefend State Diagram**

weDefend is made up of five classes. Four of these classes are objects which have a role in the scenario, while the other is the main class. The main class which will be renamed from game1 to weDefend has eight fields and ten methods. This class is responsible for holding the majority of the logic and running the two possible scenarios 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 will be severely overhauled to incorporate Plank's novel input. The update method will run in a two parameter switch case which decides whether weDefend is in the preparation phase or the action phase. It will then use the decode gestures method to link gestures to soldiers. Then the update units state method will adjust all the fields of the existing objects given they have an input gesture linked to them. Once these two methods are done the create and destroy unit methods will be called to delete or create any new soldiers or insurgents. At the end of the update method the victory and failure parameters will be checked. 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 Plank. The decode gestures method will access a linked list that is part of the gesture recognition class. This linked list will contain many gesture objects which contain the identification number, screen coordinates, and gesture type. With this data an algorithm will pair gestures with objects and assign the gesture identification number to the gesture link ID field which each usable object has. 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. The update unit state method will go through each instance of insurgents and soldiers and update their fields based on input, for example if a field of view was changed, and based on whether or not enemies came into contact with soldiers or the VIP. Create and destroy units methods will instantiate and or remove objects that need to be created or have died. The last method is actually the constructor of the weDefend class which initializes the content pipeline and identifies the graphics device being used.

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 preparation mode Boolean which tells the update function whether weDefend is in the preparation or action phase. The VIP dead Boolean which is a failure condition of the VIP scenario and lets the update function knows if the VIP has been killed. The asset overrun Boolean again lets the update function tell if a defense scenario has ended in failure. And lastly the enemies infiltrated integer will be checked in the update function to see if one-hundred or more enemies have infiltrated the area to be defended. The other two fields are linked lists called soldiers and insurgents. These two linked lists will contain each instance of soldiers and insurgents. It will be 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 four classes are the units that take part in the different weDefend scenarios. The four units are soldiers, insurgents, blockades, and the VIP. The VIP and blockade classes are sub classes of the soldier class with a few differences. Both the VIP and blockade only differ from their parent class in that they have a different sprite image and amount of hit points associated with them. The VIP will have one-hundred and twenty hit points, while the blockade will have four-hundred hit points.

The soldier class will contain many important methods and fields which the VIP and blockade will inherit. The soldier class has thirteen fields which allow all game logic and graphics to be updated. The soldier has a hit point integer which represents his health. Every soldier has two-hundred hit points. Another integer the attack power is a constant that never changes for any unit. The soldier will reduce an enemy's hit points by sixty every time he attacks an enemy. The field of view distance defines how far from the unit the field of view must be drawn. The field of view angle specifies how large the field of view is. 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 will be a linked list of point objects that contain the waypoints of where a patrolling soldier needs to go to next. This data structure will have a start point and endpoint to denote the beginning and end of a patrol route. The gesture link ID is a linked list of gesture IDs that belong to a specific soldier. The unit model is a XNA texture2D object or .x file object that contains the image or model for the soldier. The enemy target ID is the enemy that the soldier is firing at. A soldier will fire at the first enemy that enters his field of view and will 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. If two enemies are in his field of view when he switches than the unit IDs of the soldier will be compared and the smaller ID will be the one that gets attacked. The gesture exists 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 consist of set and get functions for the setting of fields explained above. The get methods will allow the weDefend methods to look into the state that each soldier is in so that they can be updated by using the soldier set methods. The only unique method is the render method. The render method uses all of the field data that a soldier contains in order to properly set up the drawing parameters for an object such as the location of the unit, the image that makes up its sprite and so on. This method will make it so that the draw method does not have to look into all the data in each object but rather call the render function of every object that still exists.

The last class in weDefend is the Insurgent class. The insurgent class has six fields and eight methods. Every Instance of insurgent created corresponds to an enemy that is drawn on the screen. Insurgents have some basic data that pertain to them. They all have one-hundred and eighty hit points. The hit point field holds this number which can be decremented as the scenario progresses; an integer will hold this value. The hit points can be updated by using the get and set hitpoints methods. The update unit state method in the weDefend class will have access to the hitpoints of all units via the get and set hit points methods. The attack power integer is a constant which cannot be changed for any insurgent. The insurgent is capable of inflicting forty hit points worth of damage every time they attack an enemy. Any time a unit is attacked by an insurgent the update unit state method will be able to access the attack power of the unit that is attacking by using the get attack power method. In this way hit points can be subtracted from the unit being attacked. In order for an insurgent to attack a soldier or VIP the distance from soldier field must fall below a certain threshold. This variable will be constantly updated to see if an enemy is close enough to a soldier to attack. Enemies will always attack the closest opposing unit to them which will be calculated using the screen coordinate field of the enemy and the screen coordinate field of the soldier that is closest to them. If the closest soldier's distance falls under the threshold then the insurgent will begin inflicting forty hit points of damage for as long as they are in range. The screen coordinate and unit ID fields will also be used to update the positions of the enemies as they move through time about their set paths. This movement route parameter will define how each insurgent that

comes into the scenario will move through the map. Each map will have a set number of areas that insurgents can spawn from and each of these areas will have their own route which all insurgents that spawn in that area will follow. This means that the route each insurgent takes will be based on their starting location and only a set number of routes will exist that the insurgent can take. One of these routes will be assigned to each instance of insurgent's movement route field. The set screen coordinate will be used to continually update the insurgent's position based on the movement route. The last method an insurgent instance uses is its render function. The render function will use the attributes such as screen coordinates and hit points to update the insurgent's appearance on the screen. The insurgent's unit model will be used to draw a visual representation of an enemy on the screen and all the draw method in the weDefend class will do is loop through all the insurgents that are present in the existing Insurgents linked list and call each of their respective render methods which contains all the updated attributes of each instance of insurgent.

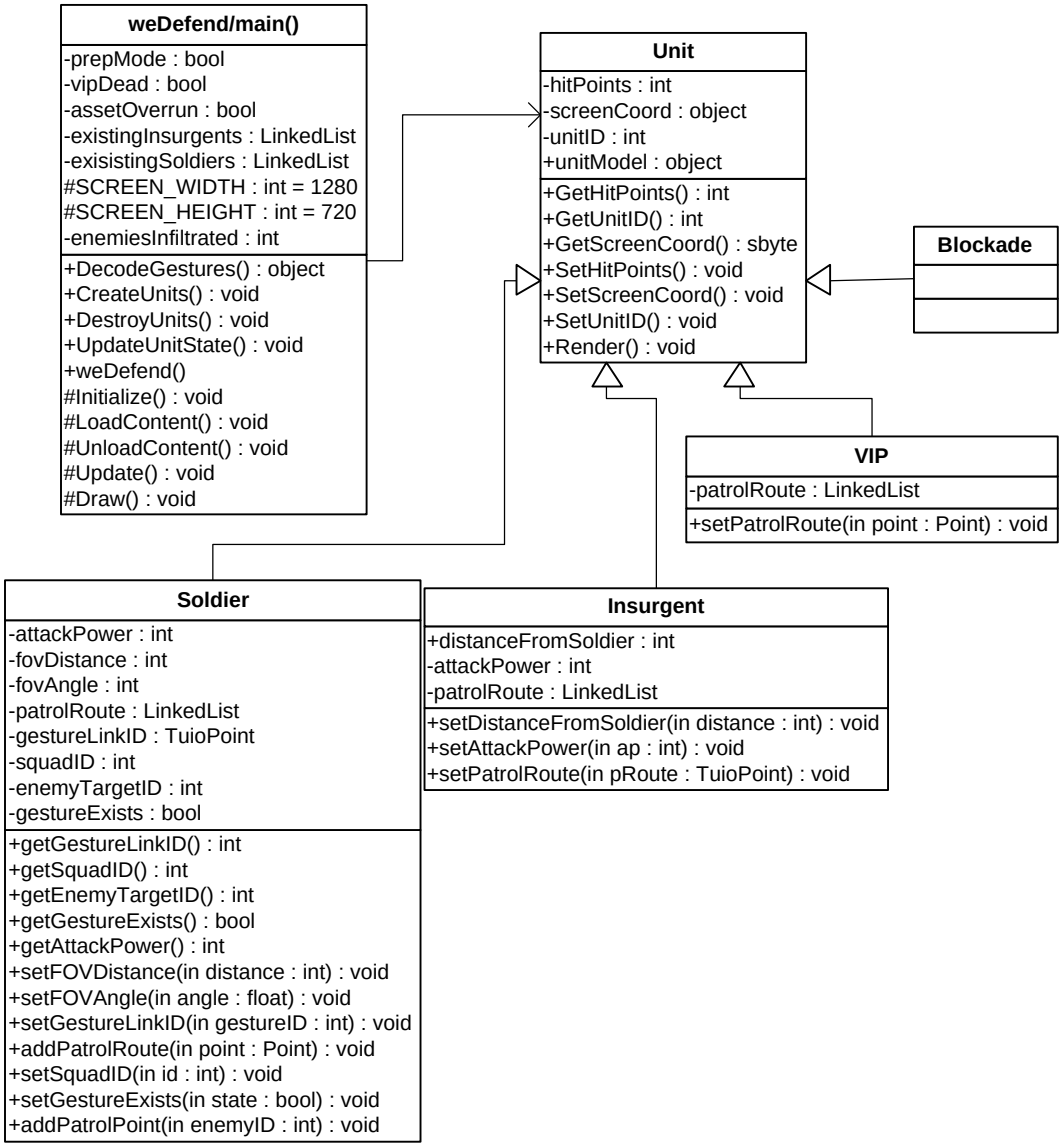



**Figure 39 - weDefend Class Diagram**

## *8.3 Computational Container System*

The computer enclosure will be solely designed for the showcase application. All decisions on the design of the system as a whole will be made with one goal in mind, optimizing the showcase application. There were a number of constraints during the design of the Computer System that had to be taken into consideration. The enclosure had 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 or glass 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-AW250N 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 will be 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 40. Using trigonometry, theangle that defines the vertical offset was calculated. This is seen as $\alpha$ in Figure 40. 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 41. 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 +/-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.

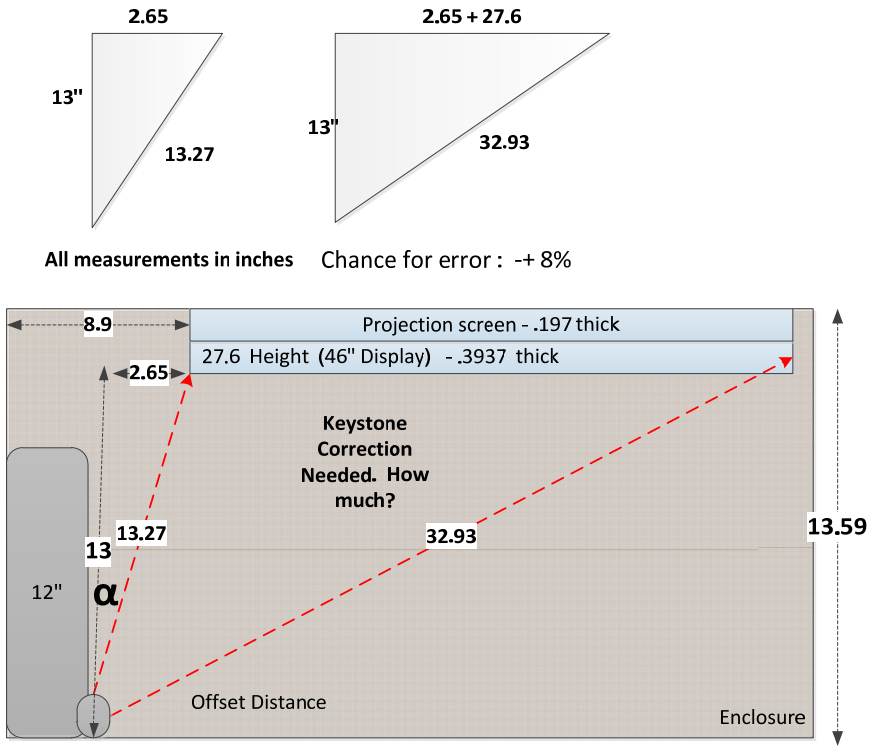



**Figure 40 - Projector Placement**

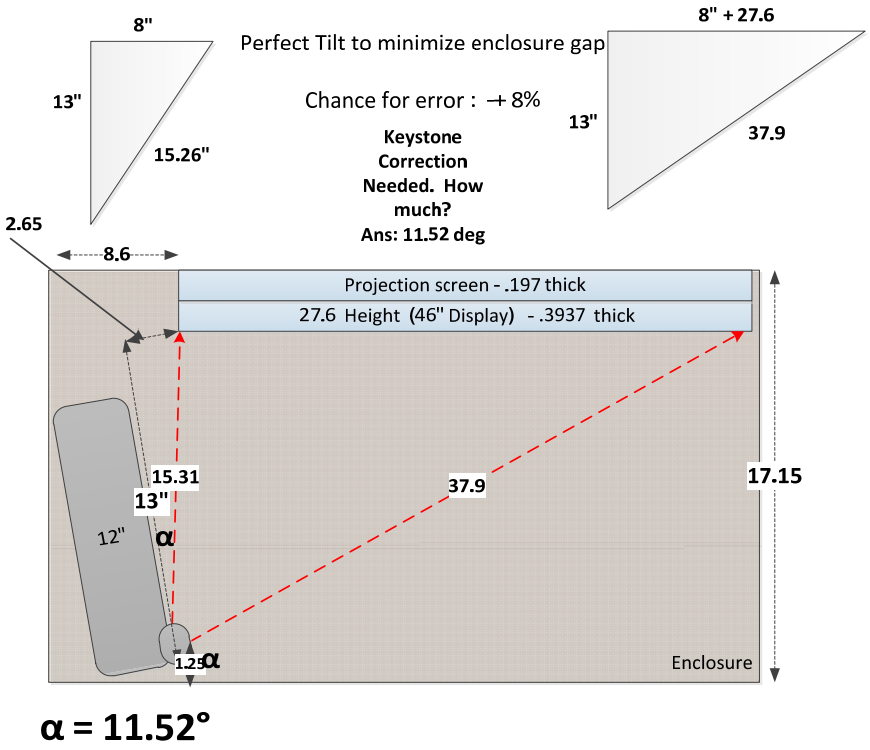



**Figure 41 - Titled Projector Placement**

## *8.3.2 Computer*

The computer that will be chosen for the system will have a quad-core Intel. The video card will support DirectX 11. To effectively take advantage of Intel's aggressive prefetching scheme, the computer should have 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. The exact computer will not be picked out at this time as computer prices rapidly change and new technology is always right around the corner. To maximize the dollar, it is also smart to seek out active deals at the time. There are many good brands on the market and it is arrogant to limit yourself to only considering one. Figure 42 shows an example of the type of computer that will be bought for the project. An Intel quad-core 2nd generation i5 2500K was picked and paired with a compatible Asus micro-ATX motherboard. Current micro-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 will only help 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 will meet 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 Intel 320 Series 80gig was chosen for their claimed read time of 270mb/s and write of 90mb/s. This will boot up Windows 7 faster than a magnetic drive allowing the user to start using the showcase application more quickly. This is because the Intel SSD has an average read throughput of 258mb/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 |  COOLER MASTER Silent Pro M700 RS-700-AMBA-D3 700W Power Supply<br>Item #:N82E16817171037<br>Return Policy: Standard Return Policy | | ~~$139.99~~<br>$109.99 |
| 1 |  G.SKILL Ripjaws X Series 8GB (2 x 4GB) 240-Pin DDR3 SDRAM DDR3 1333 (PC3 10666) Desktop Memory<br>Item #:N82E16820231440<br>Return Policy: Memory Standard Return Policy | | $47.99 |
| 1 |  LITE-ON Black SATA DVD-ROM Drive Model IHDS118-04<br>Item #:N82E16827106276<br>Return Policy: Standard Return Policy | | $17.99 |
| 1 |  ASUS P8H67-M PRO/CSM (REV 3.0) Micro ATX Intel Motherboard<br>Item #:N82E16813131711<br>Return Policy: Standard Return Policy | | ~~$124.99~~<br>$114.99 |
| 1 |  Intel Core i5-2500K 3.3GHz LGA 1155 95W Quad-Core Desktop Processor<br>Item #:N82E16819115072<br>Return Policy: CPU Replacement Only Return Policy | | ~~$219.99~~<br>$214.99 |
| 1 |  Intel 320 Series SSDSA2CW080G3K5 2.5" MLC Internal Solid State Drive (SSD)<br>Item #:N82E16820167047<br>Return Policy: Limited Replacement Only Return Policy | | ~~$184.99~~<br>$159.99 |
| 1 |  Microsoft Windows 7 Professional SP1 64-bit<br>Item #:N82E16832116992<br>Return Policy: Software Standard Return Policy | | $139.99 |
| 1 |  MSI GeForce GTX 560 (Fermi) N560GTX-M2D1GD5 Video Card<br>Item #:N82E16814127592<br>Return Policy: VGA Standard Return Policy | | $189.99 |
| | | Grand Total: | $995.92 |

**Figure 42 - Detailed Computer Components with Current Prices**

## *8.3.3 Enclosure*

The enclosure will be made out of ¾" thick Maple veneer. Maple 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 would require a lot more gluing and is more expensive.

The enclosure will be constructed essentially as an open-faced box with an all-encompassing door. Unlike the prototype where a frame was constructed to house the LED's and acrylic, the frame will be eliminated and the acrylic and LED's will be mounted directly to the sides of the box. This eliminates a step and allows us to achieve the specification of achieving the smart-phone appearance. Eliminating the frame shortens the border and allows the surface acrylic to span edge to edge on the enclosure. This can be viewed in Figure 20. 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 will be 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 will be installed on the front of the box. The door will be of maximum size to allow the person conducting maintenance on the system plenty of room to operate and repair devices. The door will be mounted on a hinge and have a handle for ease of use.

The enclosure will be 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 will be cut with a dado blade. The channel will be as thick as the EndLighten acrylic, 10mm. The channel will be cut 3/8 deep into the sides of the enclosure, half thickness of the wood. It will be ½" 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 will surround all sides of the EndLighten acrylic. The PCB will have holes drilled in equally distributed intervals down the entire frame. The spacing of the holes will be 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 43. 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 will be of approximate length of 3/8" and will be fastened using glue specially formulated to cause a chemical reaction that welds acrylic to acrylic. The rods will mount inside 3/8" holes drilled into the side of the enclosure. This detailed description is shown in Figure 43.

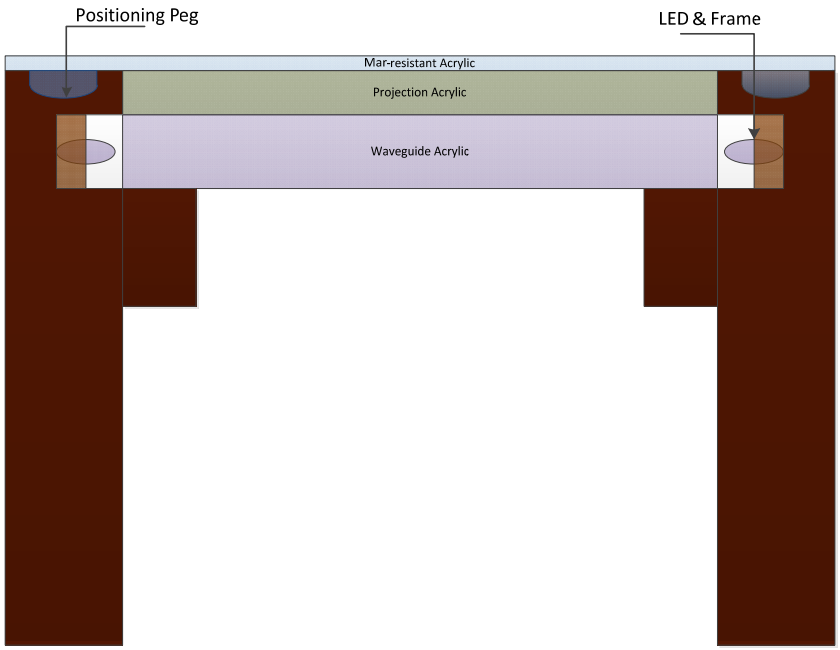



**Figure 43 - LED Frame (not to scale)**

## *8.3.4 Enclosure Features*

Two fans will be 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. 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. Rather than drilling more holes for optional fans like other projects have done, the need will be 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 will connect to the MSP40 in the Control System for power and will receive fan control based upon the MSP40's temperature sensors. This will be an accurate way of judging the enclosure temperature and controlling the heat. Extenders will be purchased so the fan's 4-plug can reach the microcontroller.

The projector needs to be mounted to allow precision tilting. To accomplish this, the projector will be mounted to a 15”x13” wooden board. The wooden board will have a long hinge, such as a piano hinge, attached and other end of the hinge attached another piece of wood glued to the bottom of the enclosure. This will allow tilting of the projector in an approximate -+90degree manner. This does allow precision tuning of the tilt though. To do this, holes will be drilled out of all 4 sides of the wooden platforms. Metal bars with channels cut in can be mounted along the side of both wooden platforms and attached with screws. Once the exact tilt is found, the screws will be tightened and the tilt angle will be set. The projector mount can be seen in Figure 44.

The computer will be mounted on wooden rods attached to the floor of the enclosure. The rods will be placed in alignment with the screw holes located on the motherboard. The motherboard will be directly screwed down to the wooden dow rods. All relating computer hardware will be mounted directly into the motherboard. The positioning of the computer will be off to the side of the enclosure and directly next to the intake fan. The intake fan will aid in dissipating the heat built up from the computer and carry it to the outtake fan. The power supply that powers the computer and various other devices will be mounted on the wall of the enclosure directly above the intake fan. The location of the computer and power supply can be viewed in Figure 44.

To meet the requirement of the enclosure being 36” in total height, legs will have to be designed. This is because of the efficiency of the projector. Rather than make the enclosure bigger than it needs to be, the enclosure will be built to be as minimal as possible. Legs will be 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 will be designed to be removed with minimal work. The legs will be made out of the same wood as the enclosure, Walnut. The legs will be 16.25” 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 will be fashioned out of 4x4’s. The legs can be viewed in Figure 44.

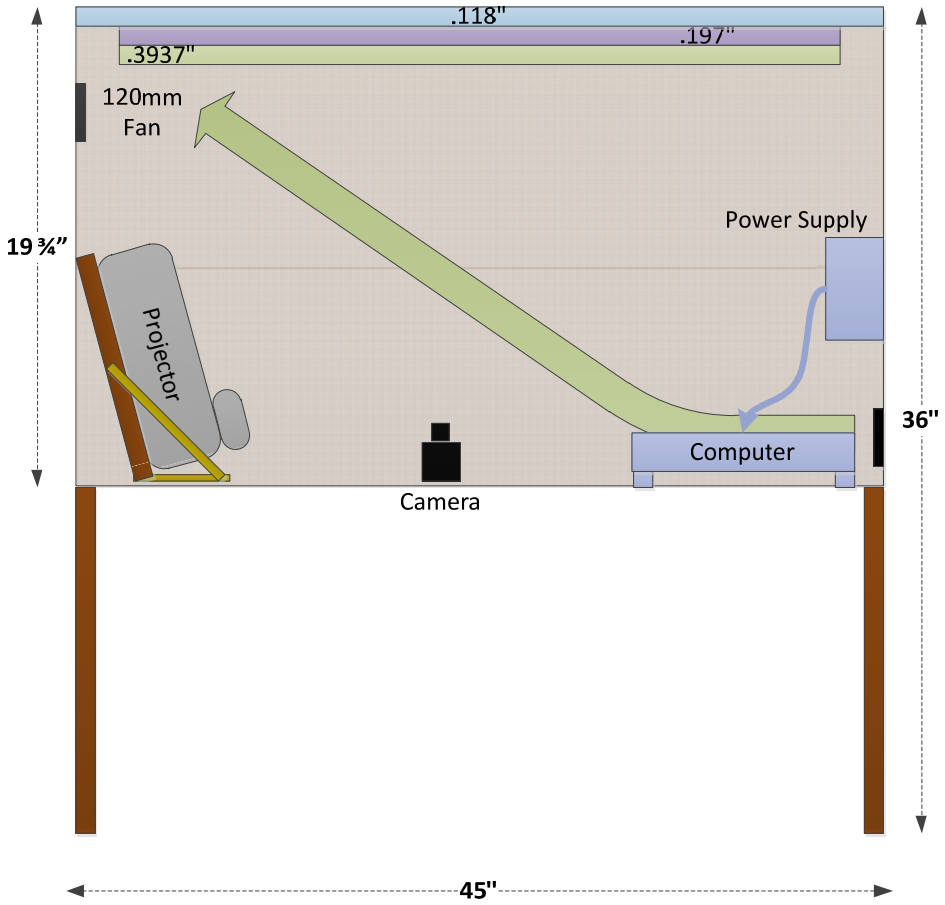



**Figure 44 - Hardware Placement Inside Enclosure**

## *8.4 Control System*

The entire control system will be 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 12v rail will be split using a 4 pin MOLEX splitter, with one end going to the computer components and the other end going to the control system. 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 maximum recommend voltage of 3.6 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 will have to be a 50kΩ resistance. The TMP37 temperature sensor will also be hooked up to the 3.6 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 clockrate of the MSP430 will be set to 8 Mhz, and will run in an approximately 320 cycle loop, or 25Khz. The input values from the potentiometers connected to the analog to digital converter will be converted to a digital value that will modulate the output pins. This frequency is required to minimize noise levels on the fans. These potentiomters will connect to pins 2 and 3 of the MSP430. The modulated signals will output from pins 8 and 9. Pin 8 will output to the 4th wire of the enclosure fans. These fans require that a input signal between 20 and 30 Khz be used to minimize noise. The 9th pin will output to the LED array. A bipolar power transistor will be used to amplify this signal to the LEDs.

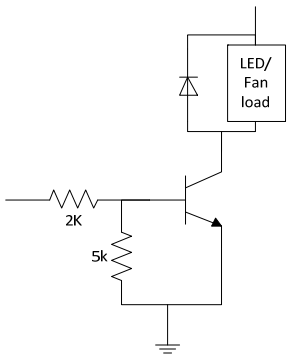



**Figure 45 - Power Transistor Circuit**

**Table 10**

| *MSP430 Pin Connections* | |
|---|---|
| **Pin #** | **Connection** |
| 1 | 3.6 Volts |
| 2 | 50kΩ Potentiometer (LED Control) |
| 3 | LM37 Temperature Sensor |
| 4 | LM37 Temperature Sensor |
| 5 | - |
| 6 | - |
| 7 | - |
| 8 | To LED Array |
| 9 | To Fans |
| 10 | - |
| 11 | - |
| 12 | - |
| 13 | - |
| 14 | Ground |

# 9. Testing

## *9.1 Touch and Fiducial Recognition Software System*

Testing for this piece of software will be implemented in phases. The gesture recognition module interfaces with CCV 1.5 on the input side and writes to a shared data structure. The first phase of testing will be to make sure the input from the screen is detected properly by the vision library. This includes touches on the surface as well as fiducials. There are a number of filters and options that can be set inside of CCV that we will need to configure and test under various lighting conditions to ensure we meet our requirements. CCV has configuration options for the touches that it considers fingers. It has a way of setting the size of an on-screen 'blob' that constitutes a finger. In the use of Phloe, our prototype, we have noticed that at times an approaching hand or a hand leaning on the table may be misconstrued as touches. Also, as a fiducial approaches the screen and begins to be recognized by the camera and vision system, CCV recognizes the points that make up a fiducial as touch points. It is only until the object is firmly and completely placed on the surface that it's patterns is recognized as a fiducial and the touch event is transmitted as such. There will need to be a test that checks for this scenario to make sure it doesn't affect the usability of Planck.

The basic TuioListener class will be used in the creation of the GestureTracker class. This communications module needs to be tested to ensure that the vision library can receive touches via the network on a computer using the written code. There is a Tuio simulator available online that simulates touches and fiducials on-screen. The simulator allows for fiducials to be placed on-screen, rotated, moved, and removed. The simulator delivers TUIO messages to any listening application on port 3333. This specific port can be changed to any port of the designers choosing. The TUIO simulator simulates the messages that would be sent from the vision library. Until Planck is fully operational and until the kinks are worked out with Phloe, the prototype of this project, the simulator will suffice to test the logic and network communication our program. The only things the simulator cannot simulate are simultaneous multi-touch events. The simulator is relegated to only moving one object on the screen (a cursor or fiducial) at a time because it is controlled by a mouse. Once the prototype and Planck are constructed, CCV 1.5 will be loaded onto them and network communication testing between the real live library and our gesture recognizer can commence. The firewall needs to be configured on the computers running the software so communication will be tested.

The shared data structure that contains the gestures will need to be tested as well. Since the GestureTracker writes to this structure and the showcase software reads from it concurrently, it will be of utmost importance that the data structure is implemented as early as possible. This will aid in the showcase software's testing. One possibility for implementing the data structure before the GestureTracker is complete is to write a small piece of software that will simulate events entering and exiting the screen/writing to the shared structure so that the showcase application can test its ability to read from the shared data structure as it is being written to.

Given the latency requirements imposed on the software, the GestureTracker needs to be tested to comply. The GestureTracker is an event driven program and only responds to events that come in from TUIO when something is placed on the surface of the screen. Latency in receiving the TUIO messages shouldn't be a problem. The bottleneck that needs to be tested for is in the gesture recognition calculations that happen in order to determine if something is a pinch-drag, just a drag, or a fiducial. The GestureTracker will be tested within a simulator and will be timed so that a thorough examination of its real-world latency can be established.

**Test 1: Test Input from Image Recognition System**
Description: This test will ensure that touches and fiducials are accurately detected on-screen and no erroneous input is detected by the vision library. Erroneous input would include a users arm leaning on the board and being misconstrued as a touch.
Results: We expect for the vision library to be able to distinguish between touches, fiducials, and other input and only send the correct information to the gesture recognizer

**Test 2: Test that the gesture tracker module works with CCV1.5 on Planck**
Description: In this test the gesture tracker module will need to be tested to ensure that CCV1.5 does communicate on-screen content with the gesture tracker via port 3333. Windows firewall on the machine being used may inhibit the communication between these two pieces of software. Also, the test will ensure that data from CCV1.5 will not arrive to the gesture tracker in a corrupted or incomplete state
Results: CCV1.5 is able to communicate effectively with the gesture tracker module. All objects on-screen are received by the gesture tracker with all of their associated attributes and data uncorrupted.

**Test 3: Test that the linked list data structure can be written to and read by both pieces of software**
Description: Create an XNA graphics based application and then create a threaded socket listener module within this application. Then use a shared link list to share information between the XNA graphics program and the threaded socket class.
Results: The XNA graphics application will be able to read the data stored in the shared data structure. The threaded socket listener module will be able to write to the shared data structure. Conflicts between the two will be handled in an error free way.

## *9.2 Showcase Software*

The most important part of project Planck is to create a novel user interface that demonstrates touch and fiducial input as a viable option for applications. Since this is the projects goal weDefend's primary purpose will be to interface with the Blob and Fiducial Recognition System. In order to achieve these goals and move on to the more ambitious aspects of weDefend proof that the gesture recognition module can integrate with an XNA Game Studio application must be provided. Therefore the first test that weDefend will undergo is the ability to have very simple graphics textures and or primitives be controlled by using touch and fiducial input.

The bulk of this test will involve using the update method in a basic XNA application and modifying it to use the decode gestures method and the update state method. These two methods, which lie inside update, will have to be able to receive TUIO objects from the gesture tracker class and associate the gestures with a primitive. The primitive will be a class similar to the soldier class. The pseudo soldier class created for this test will be able to associate gestures to a rendered object allowing the object's screen coordinate, gesture link ID, and gesture exist fields to be updated by the update unit state method. The successful result of this test will show a rendered object on the screen being moved across the screen using a touch and drag gesture. The next test will be exactly like the first except with more than one object allowing the logic of how to handle assigning multiple gestures to the correct objects.

Once gesture recognition and linking is complete more complex parts of weDefend can be tested. Once gestures can be recognized and linked to multiple objects testing the creation of units by using fiducials can begin. To test fiducial unit creation the gesture tracker class will have to provide a TUIO object that can be read and then sent as input to the create unit method. The create unit method will take the screen coordinate location reported by the TUIO object and instantiate a new soldier object with same screen coordinates as where the fiducial was detected. Another crucial test will be to test the pin-drag gesture and adjust the field of view that belongs to an object. To test adjusting a field of view, an object will be rendered that has an adjustable field of view. The update unit state method will then have to grab TUIO objects and link them to the correct unit and then finally be capable of setting the field of view distance and angle fields of said unit. The result will be an adjustable field of view that can be controlled with the pin-drag gesture. This test will also confirm the possibility of creating patrol routes due to the patrol route command having the same basic input as adjusting a field of view.

At this point weDefend will be capable of manipulating at least a single unit and all the commands associated with a unit in the preparation phase. At this point development of scenario logic for weDefend can be taken more seriously since the input coming from Planck will have been tested and working. Later testing of the action phase will involve a similar divide and conquer approach to aspects such as combat, victory and failure conditions, and implementing fully functional scenarios. A summary of the testing mentioned in this section can be found in list form below.

**Test 1: Move a rendered object by using touch input**
Description: Have the gesture tracker class and weDefend update function update a rendered object's screen coordinates in order to move it across the screen.
Result: Dragging units across the screen

**Test 2: Move two rendered objects using simultaneous touch inputs**
Description: Use the update unit state method in the weDefend class to link TUIO objects from the gesture tracker and properly associate the inputs to the correct object.
Result: Gestures can be properly linked to units using multiple inputs

**Test 3: Using fiducials to update the draw and update methods in order to create units during the preparation phase**
Description: Have the create unit and draw methods grab a TUIO object that recognizes fiducials from the gesture tracker class and have create unit method instantiate a new unit with the proper screen coordinate location and draw the unit on the screen.
Result: Capability to put units on the board using fiducials

**Test 4: Test that pin-drags can adjust field of view**
Description: Use a unit object class that owns a field of view and use the update unit state method to link the pin-drag gesture to the unit, adjust the field of view distance and angle and draw it on the screen.
Result: The field of view of a unit can be adjusted and now at least one user can control the preparation phase of weDefend assuming tests one through three succeeded. Also shows that creating patrol routes is possible because patrol routes are created by similar gesture.

**Test 5: Test that pin-drags can adjust field of view and create patrol routes for multiple units**
Description: Assure that linking of gestures to units is taking place correctly with a gesture that involves more than one finger per unit and that the extending of field of view does not interfere with other unit's adjustments.
Result: All gestures needed for the preparation phase have been tested assuming tests one through four were successful. Preparation phase of weDefend is a reality with only the need for graphics and scenario logic needing to be implemented.

**Test 6: Test all the scenario logic needed for the action phase of weDefend**
Description: This can be accomplished in parallel with tests one through five because it is about scenario logic and not input manipulation. Creating a simple prototype game can be used as a test bed for scenario logic
Result: The concept of weDefend is possible as an application that can be developed and used regardless of the input scheme.

## *9.3 Computational Container System*

Each component of the Computational Container System will be tested individually. After every component successfully passes every test case, components will be grouped together and tested together. After the Computational Container System is tested as a whole, the system can begin to be tested with other systems inside project Planck.

The first step to testing the projector is the power-on test. If the Projector powers on when the power-on button is pressed, the projector has passed the test case. It is confirmed to be powered-on if the fans start and any image are displayed onto the wall or screen of the enclosure. The second test case is to determine if the throw distance meets manufacturer requirements. The first step to this test case is to set a control for the screen image to 46" diagonal. If the measured image is less than 46", the projector needs to be moved farther away than the image location. If the image is greater than 46", the

projector needs to be moved closer to the image location. Once the image is confirmed to be 46” diagonal, the test case can proceed. The image throw-distance should range -+8% of 14.6”, the specified throw-distance for the Hitachi CP-AW250N. If the range is not between 13.432” and 15.768”, then the test case fails. If the throw-distance does fall in-between this range, then the second test case passes. The third test case is to determine if the vertical offset meets manufacturer requirements. The first step to this test case is to set a control for the screen image to 46” diagonal. If the measured image is less than 46”, the projector needs to be moved farther away than the image location. If the image is greater than 46”, the projector needs to be moved closer to the image location. Once the image is confirmed to be 46” diagonal, the test case can proceed. The image vertical offset should range -+8% of 8.9”, the specified vertical offset for the Hitachi CP-AW250N. If the range is not between 7.732” and 9.612”, then the test case fails. If it is in-between this range, then the third test case passes.

The first test case of the computer is the power-on test. The first step is to confirm that the power supply is plugged into the wall, the power switch on the power supply is set to on, and the main connector is connected to the motherboard of the computer. If this is all true, the test case can proceed. The second step is to make sure some form of a switch is connected to the correct pins on the motherboard for power-on when the two pins are shorted. If this is true, the test case can continue. The third step is to make sure all hardware components are connected to the motherboard. This includes the hard-drive, CPU, main memory, CPU fan, and video card. If this is true, the test can continue. The fourth step is to turn the switch connecting to the motherboard. If the power supply fan blades start spinning and the LED onboard light is on, then the motherboard is receiving power and the test case passes. If any of these is not true, then the test case fails. The second test case is to confirm that all devices are recognized and the computer completes Power On System Test, or POST. For this test, an image output device is necessary. The projector or a spare monitor would suffice. With the image device connected, the power can be turned on to the computer. Immediately press the command to enter BIOS. If the main memory, CPU, hard-drive, and video card are all found and labeled correctly, then the test case can proceed. The next step is to exit BIOS and allow the computer to attempt completing POST. If a beep sounds and the computer proceeds to attempt to boot from some storage device, then the computer has completed POST and successfully passed the second test case. This is to confirm that all hardware parts are receiving power and recognized.

The fans will be tested before project Planck is operational. To first and only test case to confirm that the fans are functioning is the power-on test case. The first step is to confirm the fans are wired correctly to the power supply. If no wires are crossed or disconnected, then the test can proceed. With the power supply plugged into the wall outlet, the power-on switch on the power supply can be flipped. If the fan blades start spinning, then the fans are receiving power. If now unusually loud or grinding noise is emitting from the fans, then the fans are functioning properly.

The first test case of the enclosure is to determine the enclosure is of the correct measurements. Using a Tap measure, the height of the enclosure should measure 36”

from bottom to top, including the legs. The height of the enclosure should measure 19.75" from bottom to top, not including the legs. The total width of the enclosure should measure 45" from side to side. The depth of the enclosure should measure 40.75" from back to front. If these measurements are met, then the first test case passes for the enclosure. The second test case is to determine the strength of the enclosure. By the requirements of project Planck, the enclosure must support 400lbs of weight. To test this, 400lbs of weight shall be put directly on top of the enclosure. The weight can be of any source; lifting weights, human weight. Hardware components do not to be inside the enclosure, for safety reasons. If 400lbs successfully sits on top of the enclosure for at least 10 minutes with no noticeable deterioration of the enclosure, then it passes the second test case.

Once every test case of individual components has been passed in the computer system, components can be combined and re-tested as a whole. There is no benefit to re-testing the enclosure and the projector's specifications, so these will be omitted. What is important is to test the power load on the power supply from all hardware components that will be connected to it inside the Computer System. To do this, Windows and 3DMARK11 must be installed onto the hard-drive. The first step is to confirm the fans are wired correctly to the power supply. If no wires are crossed or disconnected, then the test can proceed. The second step is to confirm that the power supply is plugged into the wall, the power switch on the power supply is set to on, and the main connector is connected to the motherboard of the computer. The third step is to make sure some form of a switch is connected to the correct pins on the motherboard for power-on when the two pins are shorted. If this is true, the test case can continue. The fourth step is to make sure all hardware components are connected to the motherboard. This includes the hard-drive, CPU, main memory, CPU fan, and video card. If this is true, the test can continue. With everything plugged in and connected, the switch can be turned to start the computer. If the power supply and enclosure fan blades start spinning and the LED onboard light is on, then the motherboard is receiving power. The computer should then complete POST and Windows boot. The next step is to startup 3DMARK11 and run a benchmark stress test. The stress test should be setup to run at least two hours. If the stress test completes successfully, then the power supply is providing enough power to the Computer System. The second test-case is to test performance of the hardware components. Following the steps of the first test-case exactly should lead to a successful completing of a 3DMARK11 stress test. A score should be provided that can be compared to the average score of a similar computer hardware setup. If the score is within the range recommended by 3DMARK11, then the test case passes and all hardware components are optimally functioning. If not, then re-installing drivers and contacting the hardware manufacturer is necessary.

## *9.4 Control System*

Each piece of the control system will be tested independently and as part of the whole. First, the voltage divider will be made to reduce the 12 volt input voltage down to 3.6 volts. The temperature sensor will be connected to the circuit and the output voltage will be measured at a known low temperature and a known high room temperature. These

measurements are taken not only to test the functionality of the devices, but also to calibrate the devices.

Once the temperature sensor is verified to be working, the temperature sensor and potentiometer will be connected to the microcontroller to verify the functionality of the analog to digital conversion. Visible LEDs will be connected to the unused general purpose input/output pins and set in the software to turn on at different voltages. The potentiometer will be varied and the temperature sensor will once again be heated and cooled. The software will then be programmed to modulate at a very slow frequency that is visible and distinguishable to eye. This will be the same program that will be implemented in the final design, only using 1000 times to the time interval. The modulated output pin will then be attached to a transistor, which will be connected to an LED and a small load which will bring it down to an equivalent to the expected current. The output will be visually checked through the LED, but will also be checked using an oscilloscope to ensure the output is giving the proper voltage. Vce will also be measured to ensure adjustments do not need to be made to other resistors

## *9.5 Image Recognition System*

Due to the nature of the components in the image recognition system, none of them can be accurately tested independently for use in this project. To accurately test the functionality of any component, at least one additional component is required. And even if a component passes testing, it may be still lacking performance in the final design. To mitigate this, each piece will be tested in pairs, as well as being tested against the components of the test bed. The exception to this rule will be the 8mm EndLighten XXL material. 6mm EndLighten XL was used for the test bed and is extremely similar to the 8mm XXL version. The nature of the enclosure that was built for the prototype is also too small and will not allow for the larger EndLighten to fit and be tested without damaging it.

The camera will first be tested with a computer with the same operating system as the final design to ensure that all features necessary can be modified manual or disabled. It will also be tested at various resolutions and frame rates that may be used. This test will be performed first since it is the only test that can be completed without voiding the return policy or manufacturer's warranty on the camera. If it is proven capable at the operating system level, then it will be disassembled so that the infrared block filter can be removed. Once the infrared block filter has been removed, the device will be tested against its ability to view the infrared LEDs used in the prototype. These LEDs are of a longer wavelength, and subsequently are further in the infrared spectrum than the LEDs used in final design. The camera will then be tested using CCV and the entire prototype to view its frame rate and visual quality versus the frame rate and visual quality of prototype's camera. Samples will be taken and quantitative measurements of resolution, grain, and sharpness will made.

A partial LED array will be made. This will be enough to cover at least half of the prototype's viewing area. The LED array will then be tested using the final design's

camera in the prototype. It will be first tested connected directly to a 12 volt power supply and tested for brightness. The camera's brightness will be adjusted so that the the brightness of the LEDs is just below peaking. Peaking would be when the LEDs are so bright that they are registered as pure white on the camera. A sample at this range will be taken. The LEDs will then be hooked up to the microcontroller and transistor. Without adjusting the brightness settings on the camera, the brightness of the LEDs will be adjusted using the potentiometer and sample will be taken at minimum and maximum brightness. These samples can be used in comparison to the directly connected version to give accurate comparisons of percentage on and off. The optical low pass filter will then be fitted over the camera. Without the LEDs illuminating the acrylic, the level at which the background is blocked should be 100% through the lens of the camera at the brightness levels that will be used during operation. Finally, the LEDs, camera, and low pass filter will then be tested together with the prototype's acrylic. Using a computer with CCV installed, touch events will be registered and the accuracy of the touches will be measured.

After the enclosure has been assembled, the camera will be fully mounted into the enclosure, along with the LEDs and low pass filter. The EndLighten and rear projection material will then be added. The camera will be calibrated for the new screen. After touch events have been registered, the screen will be tested for the minimum size of fiducials that can be registered while still being able to accurately view touch events. Several fiducials of decreasing sizes, starting at 3”x3” square, will be placed on the screen. The minimum size of the fiducial that can be recognized will be determined by the smallest fiducial that can be consistently recognized. This test will be repeated at several locations in the middle and around the edges of the usable area. The rear projection material will then be added to the system. Before the camera is recalibrated, the test for the minimum size of fiducials will be repeated again. If it is possible to obtain a better quality picture, then the camera will be further calibrated to try to reduce the size of the fiducials that can be seen while still being able to accurately view touch events.

# 10. 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.

# 11. Administrative Content

## *11.1 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.

## *11.2 Division of Labor*

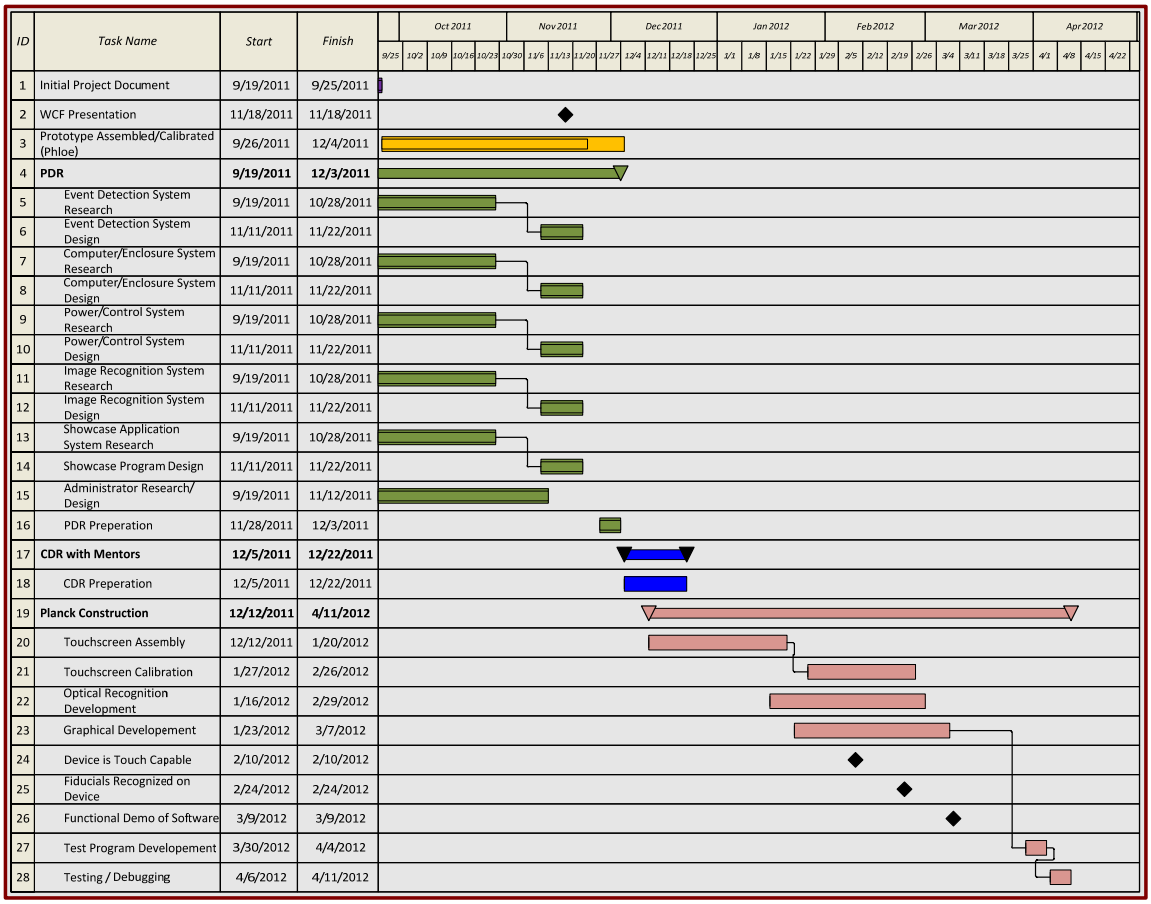
Project Planck as a whole was split up into five systems. These five systems are the Blob/Object Detection Software, Showcase Software, Enclosure/Computer, and Power/Control and Image Recognition System. Each System was allocated to one

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.

## *11.3 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.




| ID | Task Name | Start | Finish |
|---|---|---|---|
| 1 | Initial Project Document | 9/19/2011 | 9/25/2011 |
| 2 | WCF Presentation | 11/18/2011 | 11/18/2011 |
| 3 | Prototype Assembled/Calibrated (Phloe) | 9/26/2011 | 12/4/2011 |
| 4 | **PDR** | **9/19/2011** | **12/3/2011** |
| 5 | Event Detection System Research | 9/19/2011 | 10/28/2011 |
| 6 | Event Detection System Design | 11/11/2011 | 11/22/2011 |
| 7 | Computer/Enclosure System Research | 9/19/2011 | 10/28/2011 |
| 8 | Computer/Enclosure System Design | 11/11/2011 | 11/22/2011 |
| 9 | Power/Control System Research | 9/19/2011 | 10/28/2011 |
| 10 | Power/Control System Design | 11/11/2011 | 11/22/2011 |
| 11 | Image Recognition System Research | 9/19/2011 | 10/28/2011 |
| 12 | Image Recognition System Design | 11/11/2011 | 11/22/2011 |
| 13 | Showcase Application System Research | 9/19/2011 | 10/28/2011 |
| 14 | Showcase Program Design | 11/11/2011 | 11/22/2011 |
| 15 | Administrator Research/ Design | 9/19/2011 | 11/12/2011 |
| 16 | PDR Preperation | 11/28/2011 | 12/3/2011 |
| 17 | **CDR with Mentors** | **12/5/2011** | **12/22/2011** |
| 18 | CDR Preperation | 12/5/2011 | 12/22/2011 |
| 19 | **Planck Construction** | **12/12/2011** | **4/11/2012** |
| 20 | Touchscreen Assembly | 12/12/2011 | 1/20/2012 |
| 21 | Touchscreen Calibration | 1/27/2012 | 2/26/2012 |
| 22 | Optical Recognition Development | 1/16/2012 | 2/29/2012 |
| 23 | Graphical Developement | 1/23/2012 | 3/7/2012 |
| 24 | Device is Touch Capable | 2/10/2012 | 2/10/2012 |
| 25 | Fiducials Recognized on Device | 2/24/2012 | 2/24/2012 |
| 26 | Functional Demo of Software | 3/9/2012 | 3/9/2012 |
| 27 | Test Program Developement | 3/30/2012 | 4/4/2012 |
| 28 | Testing / Debugging | 4/6/2012 | 4/11/2012 |

**Figure 46 - Gantt Chart**

## *11.4 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.

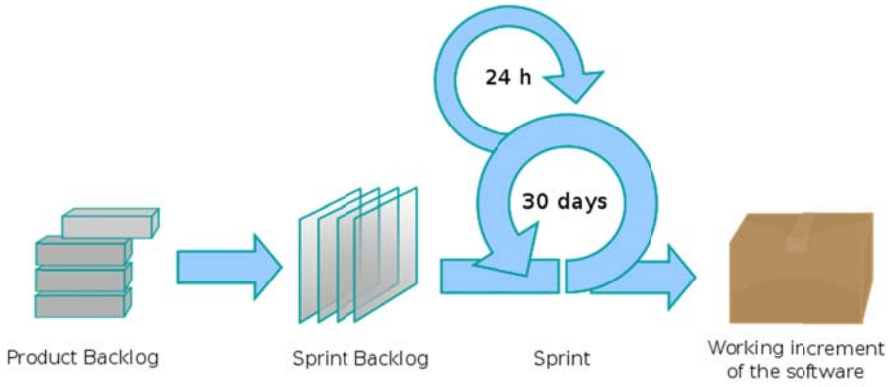



**Figure 47 - SCRUM Model**

## *11.5 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 GIT version control software is being employed. One of the members of the group has set up a server at their home where the GIT 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 GIT.

## *11.6 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.

**Table 11**

*Budget*

| Major Electronics | Projected Cost |
|---|---|
| Computer | $1,100.00 |
| Short Throw Projector | $1,450.00 |
| Cameras | $300.00 |
| Enclosure Materials | |
| Endlighten Acrylic | $230.00 |
| Rear Projection Acrylic | $220.00 |
| Mar Resistant Acrylic | $75.00 |
| Wood | $300.00 |
| Misc(screws, glue, etc) | $50.00 |
| Electronics | |
| LEDs | $100.00 |
| Fans | $50.00 |
| Other Components | $200.00 |
| Development Board | |
| Acrylic | $150.00 |
| Electronics | $50.00 |
| Other Costs | |
| Shipping & Cutting | $500.00 |
| Total | $4,775.00 |

# Works Cited


10 things you need to know aout 1080p/50. (n.d.).

4-Wire PWM Controlled fans Specification. (2005-2009).

*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

*720p*. (2010, 10 08). Retrieved 10 22, 2011, from CNET.com: http://reviews.cnet.com/4520-6029_7-6301006-1.html

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.

*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).

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. 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 fiduciary 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%20Application%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 Millitary Education Dilemma. *Forum for the Future of Higher Education* .

Martin, K. (n.d.). *Reactivision 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/
*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
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.
*Windows 7 system requirements*. (2011). Retrieved 11 2, 2011, from Microsoft.com: http://windows.microsoft.com/en-US/windows7/products/system-requirements