University of Nevada, Reno

TablIVE: A Tablet Interface for Immersive Virtual Environments

A thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Science in Computer Science and the Honors Program

by

Alexander L. Ruud

Dr. Eelke Folmer, Thesis Advisor

May, 2010

University of Nevada, Reno

Tablet Interface for Immersive Virtual Environment

Team 7 – Katie Browne, Andrew Dobson, Alex Ruud
Sergiu Dascalu
Fred Harris Jr.
Department of Computer Science and Engineering
March 7, 2009

Table of Contents (Andrew Dobson)

1. Introduction (Alex Ruud)

The main goal of our project is to create a wireless tablet interface for an Immersive Virtual Environment (IVE) that provides a persistent menu when working in a 3D space. Currently, when a user creates a menu in the CAVE, the menu is displayed either on one of the cave walls or in 3D space. The issue with this is that the menu will remain where you open it; closing it requires you to move back to where the menu was originally opened. It also takes up valuable screen real estate. A handheld device allows the ease of a persistent menu, without the worry of leaving the menu behind or missing something that is occluded by the menu.

Another goal is to create an aspect of the program that will allow you to create your own interfaces and attach functions to them. These interfaces could be saved, loaded, and modified at later dates. Ideally, the user would be able to use the new interfaces within the IVE immediately.

A secondary goal is to use tracking hardware for the tablet so the client can use it as a 'window' of sorts, viewing specialized information about the environment via the tablet. This is a good alternative to switching the entire IVE display between the normal view and the specialized.

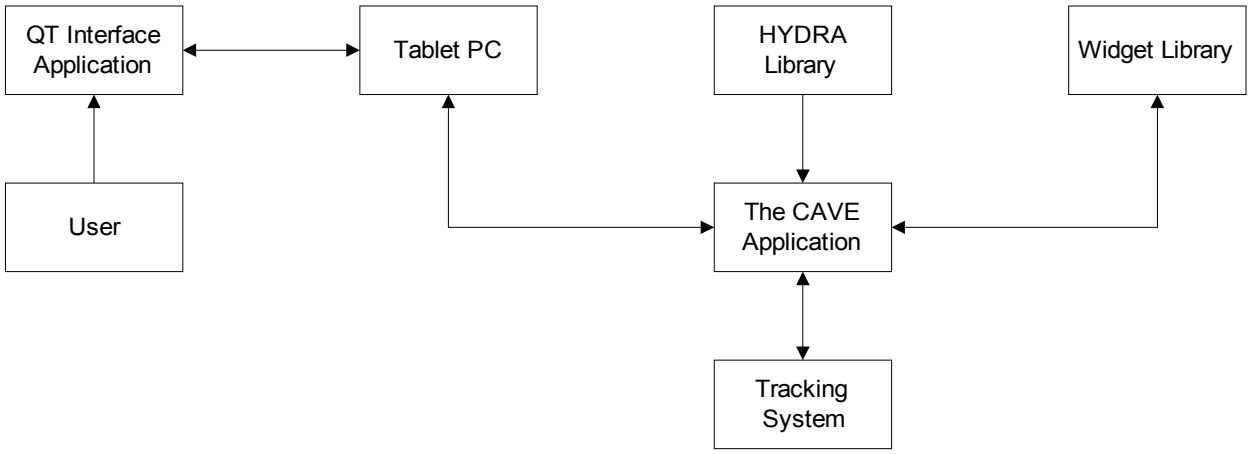
A key feature of this project is the ability to move menus into any combination of the two (2D handheld, 3D space) representations and have an action on either of them affect the other, if it's present. The actual interface will be programmed in Qt with a large OpenGL widget that will handle the menus themselves. The Qt wrapper is intended to allow for support in multiple operating systems, while the OpenGL should make programming menus straightforward.

In the time from the previous report, the group has laid down plans for the interface creator, and planned out the communication between the tablet and CAVE, thinking critically about the structure and classes needed. The design for this interface and the communication is specified in the following pages.

Our project requirements have not changed since the previous report. We are to create a wireless tablet interface for the CAVE, which also allows you to build interfaces and potentially view it in a different mode. There will be communication between the CAVE and the tablet, and the implementation should be hidden from the user.

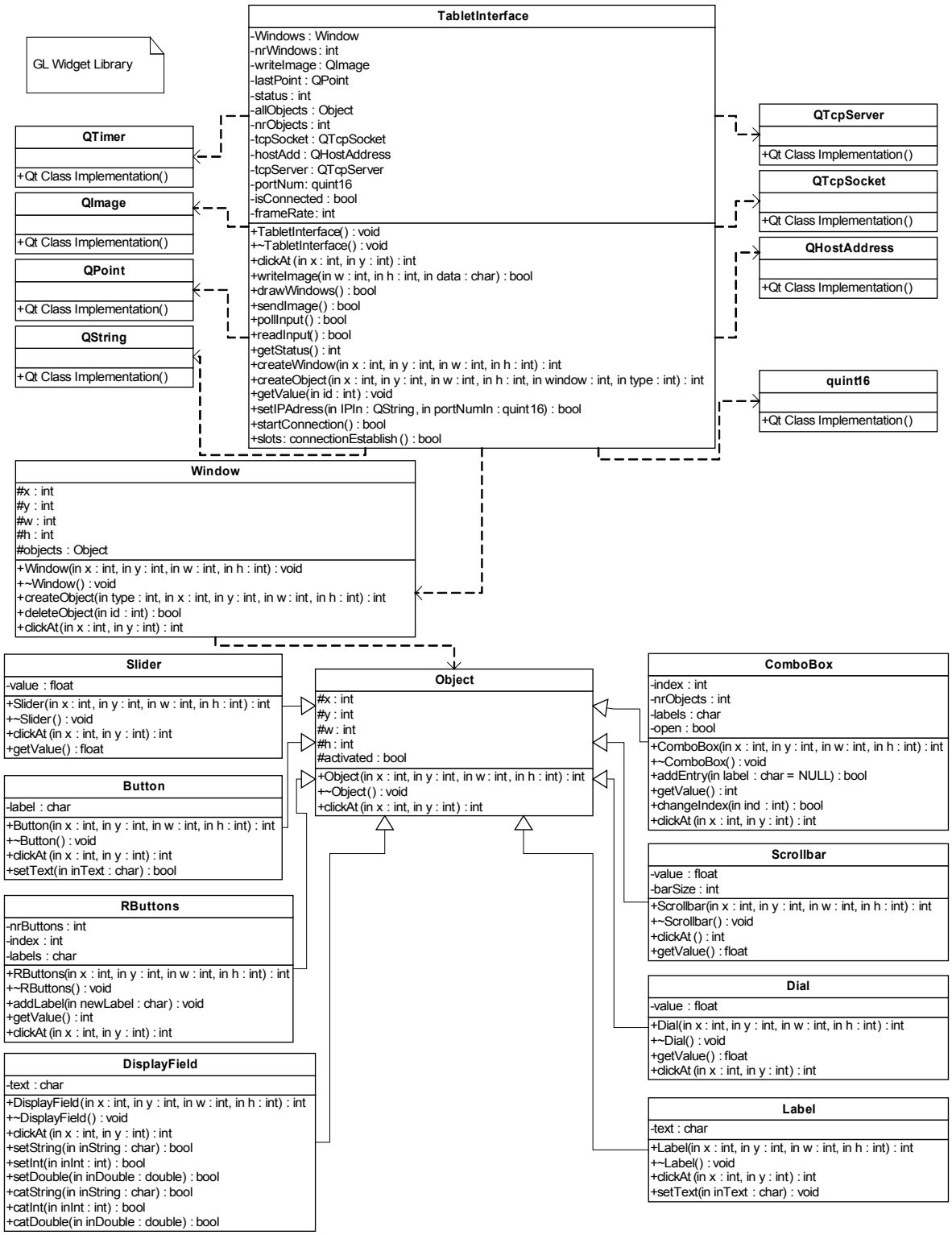
2. High/Medium-level Design (Katie Browne, Alex Ruud, Andrew Dobson)

2.1. System Context Model:



2.2. GL Widget Library

2.2.1. Program Units:



2.2.2. Methods:

public: bool tabletInterface::pollInput()

This function checks to see if a heartbeat or a point is passed from the caveInterface class and then raises a flag indicating whether or not there is input to be read.

public: int tabletInterface::createWindow(int x, int y, int w, int h)

This function creates a window of size w by h at location x, y and returns the index of the window. This index will be used to identify the window created with this function call for future manipulation.

public: int TabletInterface::createObject(int x, int y, int w, int h, int window, int type)

This function creates a specific object in a specific window of size w by h at location x, y in the window. It returns the index of the object in the window class. Type will be one of a set of predefined object types, each implemented by a different class.

public: bool Window::deleteObject(int id)

This function deletes the object with the given id in the objects array in the window class.

public: bool ComboBox::changeIndex(int ind)

This function moves the index that currently points at a label to point at a different label.

public: bool DisplayField::setString(char inString)

This function sets the private data member text to inString.

public: bool DisplayField::setInt(int inInt)

This function takes the inInt and turns it into a character array, which it puts in the private data member text.

public: bool DisplayField::setDouble(double inDouble)

This function takes the inDouble and turns it into a character array, which it puts in the private data member text.

public: bool DisplayField::catString(char inString)

This function concatenates the inString with the private data member text.

public: bool DisplayField::catInt(int inInt)

This function takes the inInt and turns it into a character array, which it concatenates with the private data member text.

public: bool DisplayField::catDouble(double inDouble)

This function takes the inDouble and turns it into a character array, which it concatenates with the private data member text.

public: bool tabletInterface::writeImage(int w, int y, char* data)

This function allows the Application Programmer to write image information into the tabletInterface's image buffer. The passed in information is width and height information, and then an array of characters. The characters allow for information to be passed in 3 bytes, one for red, one for green, and the last for blue. Writing to this buffer allows us to send only complete images to the tablet application.

public: int tabletInterface::clickAt(int x, int y)

This is the highest-level clickAt function that will be called after the tablet sends a signal indicating that there has been a mouse click at some location. This function will be called with the x/y coordinates extracted from the message and will eventually resolve what object, if any, has been activated by the click. This function can call the clickAt function for a window, which in turn can call an Object's clickAt function. The final return value is the id of the activated object, and a -1 will indicate that no object was hit.

public: bool tabletInterface::drawWindows()

This function will draw the interface windows on top of the currently existing image in the tabletInterface class' buffer. Once these windows are drawn on top of the image, then the image is ready to be sent to the tablet PC.

public: void* tabletInterface::getValue(int id)

This function returns a pointer to the information stored by the object. Each object is presumably made by the application programmer to do some sort of I/O with the user using the tablet PC. The Application Programmer is responsible for typecasting the returned void pointer to the appropriate data, and the Application Programmer should be keeping track some way of what information should be coming from each object, as they will be creating the objects in the first place to facilitate some sort of I/O interaction.

public: virtual int Object::clickAt(int x, int y)

The Object's clickAt function is an abstract function to be implemented by all of the different types of classes inheriting from the abstract Object class. Each clickAt function must

be able to calculate values for the object uniquely, and oftentimes, depending on where within the bounds of the object the mouse click occurred.

public: double Dial::getValue()

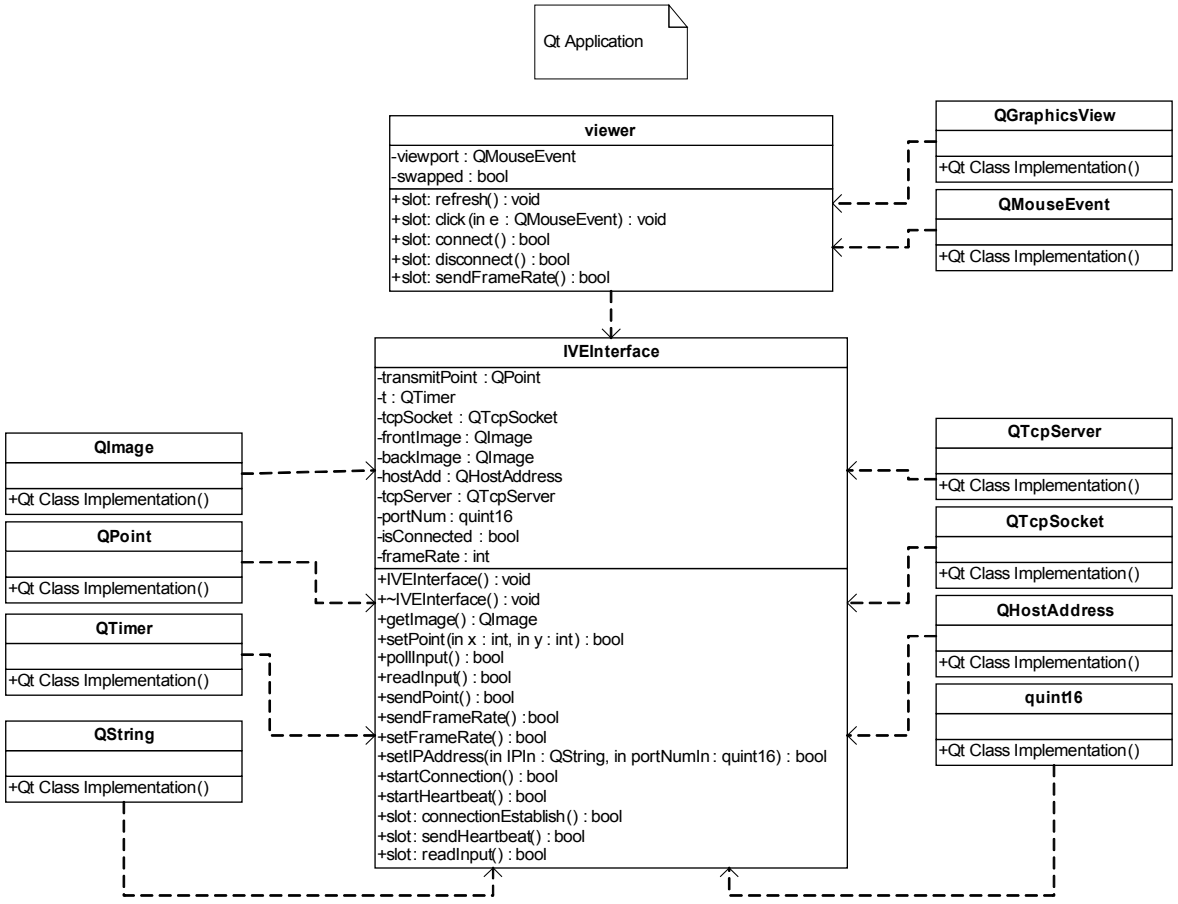
This function will simply return the value of the dial, though that is not as straightforward as it may seem. It is simple enough to calculate a double representing the angle on the dial; however, it will be based off of the same absolute angle (0 degrees starting from the right) for all dials, and depending on the application, the programmer may have to do some interesting manipulation to get the returned value to work as desired.

public: bool caveInterface::sendHeartbeat()

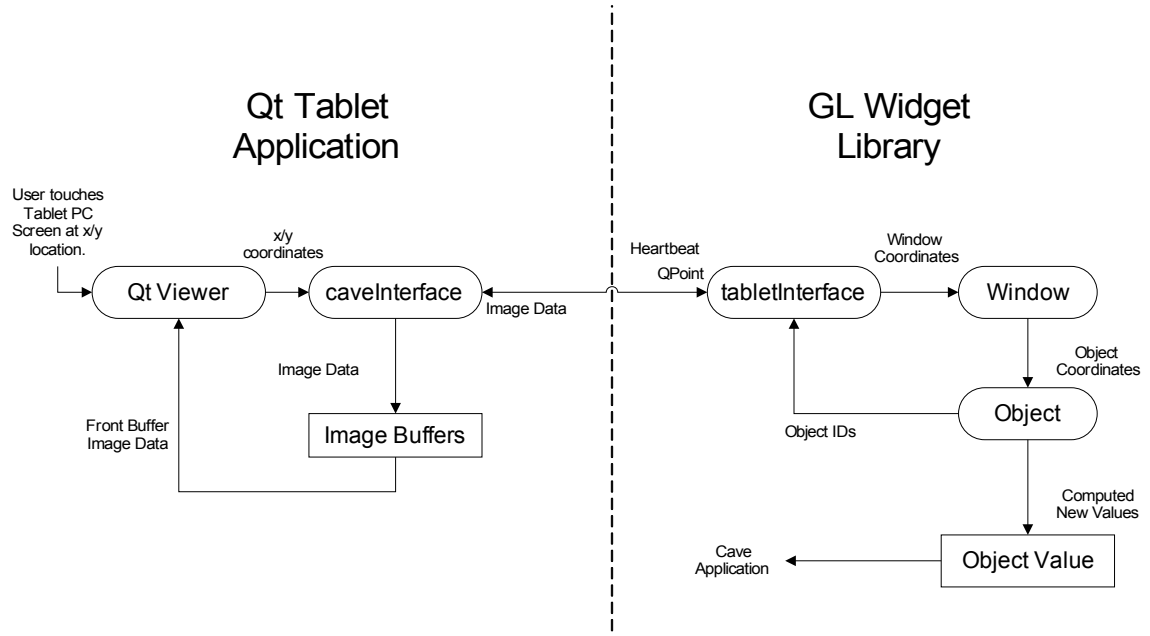
This function will be called on a timer interrupt in order to send out a "heartbeat" signal to the CAVE. This signal will be sent out periodically to inform the tabletInterface class that the tablet is still active and in good communication.

2.3. Qt Tablet Application

2.3.1. Program Units:



4. Data Flow Diagram (Andrew Dobson, Katie Browne)



5. User Interface Design (Andrew Dobson, Alex Ruud)

Fig 1. The basic Interface Window for the Tablet Application

Fig 2. An example of a Virtual Realm as viewed through the Tablet Display.

Fig 3. A plausible alternate view as viewed through the tablet interface.

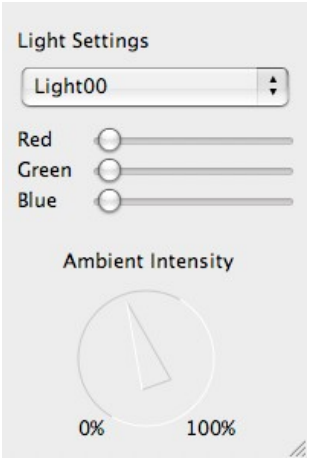


Fig 4. A window prototype, as would be created from the GL Widget Library.

Fig 5. An example of the GL Widget Library rendering a menu on top of the virtual realm view.

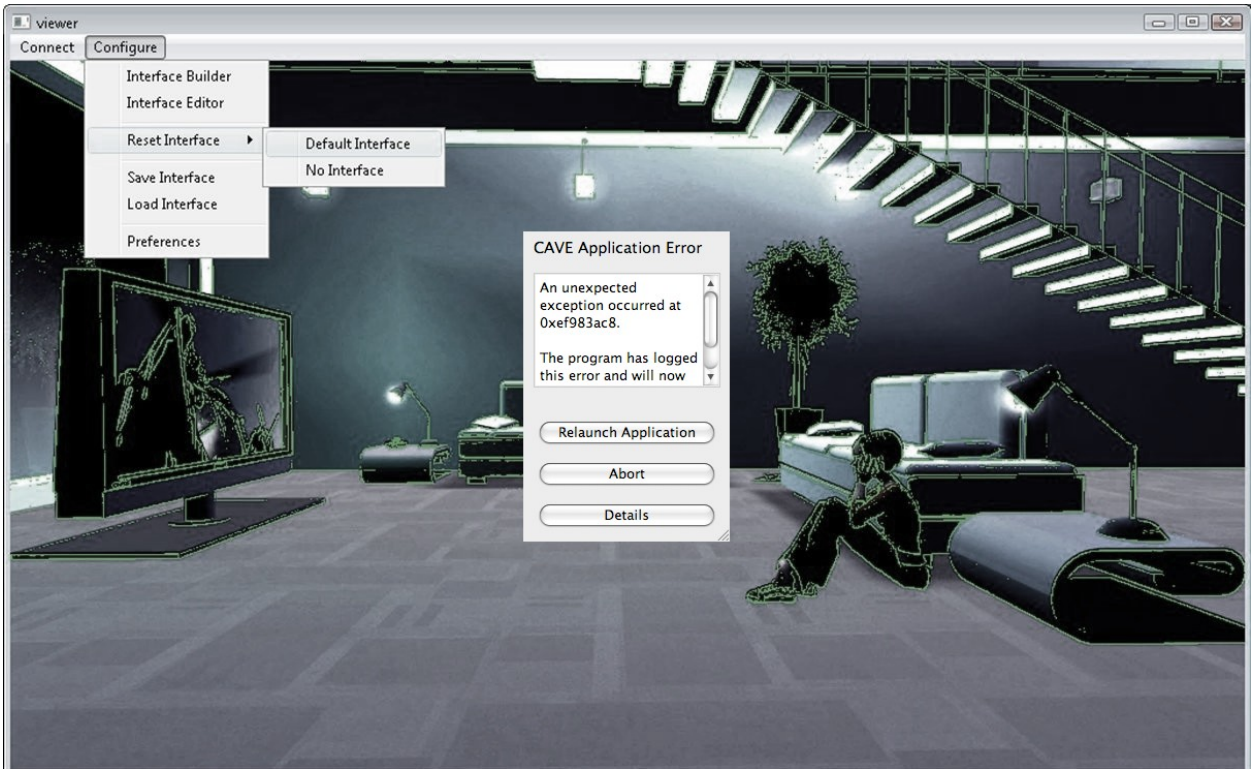


Fig 6. An example Application Error window that Application programmers could make.

6. Annotated References (Katie Browne)

Angel, Edward. OpenGL: A Primer. Boston: Pearson Education, 2008
This book is a reference on how to use the open graphics library (OpenGL). OpenGL allows a programmer to use the graphics card on a computer to display graphics. One of the main parts of this project is to make an OpenGL widget library and this has all the information to help in the process of making the library. Chapter 2 is all about programming two-dimensional objects in OpenGL which will come in handy for all of the widgets in the library. Also the section on double buffering will help with displaying an image on the tablet while reading in another image from the cave application.

"Bandwidth Explained." 2002. 1 November 2009
<http://www.findmyhosting.com/bandwidth.htm>.
A major part of this project is being able to connect the tablet PC to the CAVE through networking as well as being able to update the image on the tablet PC 10 times a second. This article provides an explanation of networks and how they work. It also talks about what bandwidth is, how to calculate bandwidth, how to get the most out of bandwidth, and how to host bandwidth which may be necessary if the image on the tablet PC takes to long to load. This will help with finding the bandwidth needed to pass the menu image from the CAVE to the tablet and the bandwidth needed to pass the coordinates from the tablet PC to the CAVE.

Kenyon, Robert. “The CAVE Automatic Virtual Environment: Characteristics and Application.” November 1995. NASA 28 October 2009 <http://www.cs.uic.edu/~kenyon/Conferences/NASA/Workshop_Noor.html>.
This article explains everything about a CAVE. It talks about how a CAVE is put together, the video system, the audio system, the magnetic tracking system, CAVE calibration, update rates, and system lag. Since this project is about making a menu system and device to interact with the CAVE, it is important to know how the cave works. Also the CAVE will need to track the tablet PC and the section on magnetic tracking discusses how this is done. Furthermore, it is important to know the update rates of the CAVE in order to attempt to keep the tablet PC at the same update rates.

“Qt Reference Documentation.” 2009. Nokia. 1 November 2009 <http://doc.trolltech.com/4.5/index.html>.
This website is all about Qt. It gives tutorials, examples, and demonstrations on using Qt. It also provides a list of classes, functions, and namespaces in Qt along with their definitions. Qt will be used on the tablet PC as well as in the GL widget library. The widget library will utilize the QNetworkAcessManger class, the QImage class, and the QPoint class and the tablet PC will use the same classes plus the QTimer class. This website provides the properties of each class, the public functions of each class, signals of each class, inheritance between classes and examples of how to the classes. This website will also help with developing the window and menu bar on the tablet PC and will provide support if need be.

UNIVERSITY OF NEVADA RENO

THE HONORS PROGRAM

We recommend that the thesis
prepared under our supervision by

ALEXANDER L. RUUD

entitled

TablIVE: A Tablet Interface for Immersive Virtual Environments

be accepted in partial fulfillment of the
requirements for the degree of

BACHELOR OF SCIENCE, COMPUTER SCIENCE

_____

Eelke Folmer, Ph.D, Thesis Advisor

_____

Tamara Valentine, Ph. D., Director, **Honors Program**

May, 2010