A Lightweight, Cross-Platform System for an Immersive Experience in Virtual Exploration of Remote Environments

By

Fahed Al Hassanat

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa
2014

Abstract

In this thesis, we present a tool that is an extension of the NAVIRE Framework and used in remote environment exploration and navigation. It is available for end users on everyday devices such as desktops and mobile devices. The tool offers the characteristic of being cross-platform and easy to use and manipulate. Cubic panoramas that are generated by the NAVIRE Framework are loaded in the tool and rendered in 3D space on the user's device. The 3D panoramas are interactive by allowing the user to view the scene in all direction, move from one scene to another or through interaction with augmented objects. The tool is geared to be interactive in the most natural and intuitive fashion using the inputs available in the different devices of the end-user.

Keywords- *NAVIRE, Panorama, Remote Scene Exploration, Immersive Experience, Cross-Platform Application, 3D, Usability Study.*

Acknowledgements

Dr. Robert Laganiere for your inspiration, support and patience.

Contents

List of Tables

List of Figures

Chapter 1: Introduction

Virtual exploration of remote environment in its simplest definition is illustrated by viewing a representation of an environment that is not necessarily the current environment of the viewer; In other words, discovering a location through viewing a virtual representation of it, through one or more types of media. Virtual exploration can be seen visible in the following applications:

- Tele-Tourism: Visit hard to reach places, such as arctic or underwater environments.
- Entertainment: Games based on augmented reality.
- Education: Learn on site.
- Engineering: Site investigation and analysis.

Remote environment exploration can be performed in many ways that range from basic to complex. The following are examples of variations of remote environment exploration:

1. Viewing a collection of regular images of a scene: this is the most basic form of exploring a scene. The limitations lay in the availability of images that cover all viewpoints of the scene and being taken with close time proximity.

This is usually does not provide the user with a rich nor immersive experience.

2. Viewing images that are associated with a geographical location: While it is not much different than the first method, it does provide an extra link to reality by associating a physical location with the images.
3. Viewing Panoramas: This is adds an advantage over the earlier methods in that all viewpoints are visible, yet it suffers from the lack of realism of the scene, the possible distortions caused by panorama capture methods and the inability to enrich the scene with augmented objects.
4. 3D viewers: those are tools that take panoramas and wrap them with 3D realism allowing the user to have a more realistic experience.
5. 3D viewers with navigation capabilities: similar to the previous method, yet it adds the ability to move between scenes of the same environment simulating the user movement.
6. Fluid (4D) Navigation: this is a 3D panorama exploration approach with time dependency creating a video-like feel in 360 degrees.

If the intention is exploration of a scene, then a panoramic image serves the purpose better than a regular image since it contains a more complete set of information about the scene, i.e. multiple viewpoints of the scene in one image. One of the most basic examples of such concept is Flickr with the variation of its regular (planar) images and panoramic ones. The latter can be created easily via options on smartphone cameras or using a modern digital camera. Other fine examples are the efforts done by the University of Washington in the Photo Tourism project [30] and Panoramio, similar to Flickr but with image geotagging, which is powered by Google Earth [31]. However, in order to add richness to the experience of viewing a panorama, one can add the ability to navigate between panoramas within the same local with a certain degree of relationship (i.e. common views); Also one can incorporate the viewing mechanics natural to the human eye when exploring an environment such as looking up, down and sideways. So in the end, the quality of the virtual exploration experience is measured by the quality of the interactive tool that provides it.

1.1 NAVIRE Project at the University of Ottawa

This research project falls under the umbrella of the NAVIRE project [23] (virtual NAVigation in Image-based representation of Real world Environment) at the VIVA lab of the University of Ottawa. NAVIRE is a project that covers all aspects of virtual navigation in real environments, from the initial image acquisition to the final interactive navigation tool.

This thesis aims at leveraging the work done within the scope of NAVIRE in order to present the end user with the means of having a rich, immersive and unique experience in exploring remote environments.

The work in this thesis describes a system that allows the user to take a virtual tour through browsing one or more panoramic scenes. It uses images captured by specialized devices, such as the Point Grey Lady Bug, and processed with algorithms that correct and combine them, to construct a panoramic view of the location where the images were taken. It provides the user the capacity of exploring a scene in any desired angle as well as navigating from one scene to another by following a logical route that simulates reality. It also provides the user the ability to interact with virtual objects, multimedia 2D or 3D objects, placed within the scene for the purpose of enriching the navigational experience.

One of the most significant aspects of this project is to make the rich and immersive aspects of viewing and navigating, the remote environments, available on popular devices, thus considerations were taken for cross platform availability. Care was taken to address this challenge and the proper technology was used to extend the availability of the system through web browsers on personal computers and apps for smartphones and tablets for various operating systems with no device dependency.

1.2 Thesis Objective

The work in this thesis aims at providing a system that will translate a multitude of geo-localized panoramic images, for a certain environment, into a rich, immersive and interactive navigational experience. It also aims at allowing this experience to be discovered through as many, commonly available, devices as possible, and with the lowest level of complexity available. This project also aims at enriching the navigational experience by augmenting the panoramic images with virtual objects that can be static or dynamic and can contain any type of media i.e video, audio, 2D/3D object…etc. Taking into account all of the above, the elemental factor of this project is to emphasis on the rich and immersive aspects of the experience by giving the user the most intuitive approach to acquire it, this is achieved by building a game-like navigation process for personal computers, and an alternative for mobile devices, smartphones and tablets, that leverages the available sensors such as GPS receiver, Gyroscope sensor and the compass, if available, as well as the smooth experience transition from mouse based devices to touch-screen based devices.

1.3 Thesis Challenges

There were challenges that presented themselves in this project, many solved and few remain:

- The first and most noted challenge was to provide the user with the most intuitive way of exploring a scene and allow this intuition to remain intact as the user moved from using a mouse based device to a touch based mobile device with sensors.

- Create a map that will describe the scene programmatically and have flexibility to describe objects that do not belong to the scene, such as virtual objects. This map will also hold information about the

relationship between panoramas, orientation of the panoramas as well as location, distance and other descriptive attributes of the scene.

- Address bandwidth management for devices where high bandwidth consumption might incur high costs.

- Handle the preloading of assets to create a smooth navigational flow.

- Deduce optimal user interface design from blending known approaches with general guidelines for designing a cross-platform system.

- Explore the technology that will allow the best cross-platform integration with minimum development cycles, without affecting the performance on any of the targeted devices.

- Design to accommodate the rapid change in mobile device specification and allow for stable performance across all targeted devices with minimum operating system dependency.

1.4 Thesis Contribution

The contributions brought forth in this thesis are in the fields of scene navigation, user interface design, and mobile device development, and can be summarized as follows:

- The ability to navigate a virtual scene with basic intuition and with a low learning curve.

- The use of mobile device sensors, such as GPS, gyroscope and compass, in interacting and navigating a panoramic scene.

- The ease of transition between different devices when working with a common interface such as mobile device and a personal computer.

- The creation of a tool to view and interact with panoramas on mobile devices.

- The ability to augment the scene with 2D and 3D objects, static or interactive.

1.5 Thesis Outline

Chapter 2 presents the user with an overview of the NAVIRE project and the methodology that it follows to produce the elements necessary for the viewer to function. This starts by describing the image capture mechanism and the hardware involved in the capture. It also describes the image correction stages that the captured images pass through as part of the preprocessing stage to the viewer. Finally it describes the viewer system which is the core topic of this research project.

Chapter 3 goes to describe in details the architecture of the system of viewing and navigating the collection of panoramas. It describes thoroughly the server side and client side components, including the software design approach, the various delivery technologies and benefits gained by adopting the stated approach and technologies.

Chapter 4 illustrates, with precision, the mechanics of the system and how the pieces of the system interact with one another. This starts from the construction of the elements of the scene: images, virtual objects and navigation elements, then the navigational process within the same scene and then from one scene to the next. Then it moves to describe the intricate aspects of the mobile implementation of the client side with all the variations between operating systems, those chosen to be supported, and the use of sensor data as an input to control the said client side. Finally, this chapter

describes the augmentation aspect of the system from the virtual element description phase to its construction on the client side and how it interacts with the navigation process throughout the scene.

Chapter 5 is a detailed investigation of the user interface usability of the client side on all targeted clients devices. It describes the influence and integration of known usability metrics and guidelines on the user interface implementation.

Chapter 6 states the result of this research project and the various implementation aspects that makes it different from other related works. It also describes the weak points of the system and the room for improvement along with future prospective of enhancements to the system.

Chapter 2: Related Work

In the previous chapter we introduced the topic of this thesis with an overview of its aspects from objectives to contributions. In this chapter we discuss the complete system in which this project is part. Navire is a project aimed at investigating and creating a virtual representation of real-world environment through the processes of capturing, correcting and rendering images of such environments. Although the capture and correction portions of the project are outside the scope of this document, and that the main purpose of this document is to understand the viewing aspect, it is important to grasp some of the concepts involved in the earlier mentioned aspects of the project.

Panorama, as a concept, has become more popular during the last few years through either its application or the means of capturing it. Almost everybody has come in touch with this concept by using applications like Google Street View or capturing a panoramic image using their smartphone, or a digital camera equipped with this feature.

Although there have been many research projects that have discussed and provided solutions to navigation in remote virtual environments, it was noticed that the viewer portion of the suggested solution have always had the minimum coverage within work. However, one must note that in order to provide the user with the

proposed rich experience, it is imperative that this be done from the point of view of the user.

2.1 Previous Work

2.1.1 Mark Fiala's work

In [1], Fiala proposes a framework that allows for the design of a system whereby users can explore and navigate remote environments. The system employs either a Head Mounted Display (HMD), an apparatus that users wears on their head and present the video or images describing the scene through a display screen in front of their eyes, or Cave, which is a closed room where the media describing the scene is projected on its walls.

With HMD, Fiala states that the users can explore the remote scene in a very natural way by moving their head in all directions to view the scene from different viewpoints, mimicking the behavior when exploring an actual scene. Resulting scenes are generated from a set of panoramic images captured by a specialized camera from locations that relate to each other through a reasonable geographical and visual proximity. Also, the system provides the user with a representation of intermediate scenes where the camera was not physically present to capture it, as long as it positioned along a linear path between two captured scenes.

Fiala [1] stressed that a low latency system is necessary to provide the users with the best possible experience in exploring a remote scene. To achieve this, he resorted to the following:

- De-couple image creation from image generation.
- Use cube format to describe panoramic images.
- Use a fast algorithm to generate intermediate views based on a pre-calculated disparity maps.

The system described in this document follows in the footsteps of Fiala's proposed system in de-coupling the creation and generation of the images, as well as using cube panoramic images, both in which are described in the following sections of this chapter.

Fiala describes in [1] that intermediate panoramas can be generated by applying view morphing techniques on the two bounding panoramas. Depending on the direction of navigation between the two original panoramas, the intermediate panorama will be calculated through two steps:

- First, the radial expansion /contraction of pixels from the portion of the front/back panoramas that is directly on the line that dictates the direction of navigation, and
- Second, the horizontal displacement of the panorama pixels , along the line of navigation between the two original panoramas

The first step is based on a disparity map with radial component only, and the second is based on one with the horizontal component only. Those two steps will be revisited in section 2.3 to shed more light on the association of the view morphing technique with the use of cube representation of the panorama.

It is important to note that there are essential components such as disparity maps and rotation matrices that are calculated offline as part of de-coupling process.

Figure 1: 360 Degrees catadioptric cameras, as in [32].

Fiala's implementation experimented with three omnidirectional cameras. All cameras were catadioptric. Catadioptric cameras are composed of a combination of lenses (dioptrics) and mirrors (catoptrics). In this instance, a mirror is used to collect the light from 360 degree environment, and the camera is used to record the captured light. Figure 1 shows few catadioptric 360 degree cameras, and figure 2 shows a sample panoramic image taken with this camera. One must note that the image in figure 2 is not processed yet for the use in Fiala's implementation.

Figure 2: Panoramic image acquired with a catadioptric camera, as in [32].

In the end, HMD or Cave, both options require either specialized hardware or specific viewing environment in order to complete the experience. While it is understandable that these solutions are meant for specific audience, the challenge arises as to provide a solution that befits almost all users and with the capacity of viewing on hardware that is popular within the user environment.

2.1.2 Google Street View

Perhaps one of the most well-known, well established and most widely used applications for remote environment viewing is Google Street View. Developed by Google and seamlessly integrated with the famous Google Maps, Street View complements Google Maps by bringing the user to an actual panoramic street level view of the remote location. The user has the

ability to zoom-in from the Map to the Street View and zoom out in the opposite direction. The user can navigate within Street View based on the availability of nearby captures, as well as zoom-in on parts of the currently viewed scene.

To be able to capture, collect and compile the large set of data currently available, Google uses various terrain-specific vehicles to gather the data. The vehicles used to traverse the wide array of terrains are a car, trike, trolley, snowmobile and a trekker, as illustrated in figure 3.

Figure 3: Terrain- specific vehicles used by Google to collect data for Street View, as in [33].

When the conditions permit, the capture system typically consists of the following components:

- The transportation vehicle.
- 15-lens camera system capturing 360 degrees view.
- Motion sensors.
- GPS sensor.
- Laser for capturing 3d data from the scene.
- A hard drive to store the data.
- A computer running the system.

Google plans capture sessions in good weather and precise timing of day in order to have the proper balance of lighting and shadow as well as good scene visibility. It is also important in order to acquire an accurate GPS position, heading and altitude which allows for the proper matching of the scene with its location on the map. When GPS data is not available or

inaccurate, it is supplemented by information gathered from other sensors on the system.

Individual images of the scene captured by each lens is stitched together to form a continuous 360 degree panorama. Google then applies special image processing algorithms to remove or lessen any undesired effects created by the stitching process. The stitched panorama is eventually projected on a sphere in the product presented to the user.

Figure 4: individual images and stitched panorama from Street View system, as in [33].

In the viewing process, Google uses 3D models constructed from the data collected from the laser sensors at the time of the capture to evaluate distance to structures. Based on this 3D model, when the user selects a far area to view, the system selects the best panorama to show.

Despite being an impressive project, Street View remains limited in certain areas which can be justified by the global use of the project and how important these limitations are to the end users. Limitations are visible in the variable results of the stitching; it is not always seamless, limited indoor collection and abrupt transition between panoramas.

Street View is available as an extension to Google Maps for desktop browser and an application for Android and iOS based mobile devices.

2.1.3 Other contributions:

Other than the two projects mentioned earlier, there exist other solution that can be set into the following categories:

- Image database:
    - Flickr: an online service that allows users to upload their personal images, and amongst those images there exist panoramic images that are captured with various equipment.
    - Panoramio: similar to flickr but adds geolocation association to the image. it is used as a layer in Google Earth and Google Maps.

- Virtual Navigation:
    - QuickTime solution [24]: similar to the solution proposed in this document, it is a system for navigating virtual environment based on panoramic images.

- Image-based exploration:
    - Photo Tourism [30]: a project at the University of Washington that aims at exploring a scene based on a collection of its images. The images are placed in 3d space based on their viewpoint of the scene according to a sparse 3d model of the scene.
    - Other systems worth mentioning is proposed by Yuttendele et al.[11], as well as the one proposed by Guarnaccia et al. in [13].

2.2 Overview of the NAVIRE System for Virtual Navigation

2.2.1 Capturing the environment

The capturing process starts with the Point Grey Ladybug [14], a multi-sensor digital video camera. Each of the six sensors is 1024 x 768 CCD. The sensors are aligned in such fashion that the camera would capture more than 80% of the spherical view. Five of the sensors are aligned equidistantly on a horizontal ring, and the last sensor is placed vertically pointing to the top. The pixels of the sensors are made to, almost; cover a sphere, with an 80 pixel overlap between each sensor and its neighbor, thus having approximately a total of 4.7MP for the whole scene. Figure 5 shows how a group of images that describe a scene are grouped and associated with other scenes in order to facilitate navigation.

Figure 5: Navigating remote environments.

The camera is mounted on a vehicle, a scooter [10] that will be used to traverse the areas of interest and allow the camera to capture the desired amount of information needed to represent the said areas. Figure 6 is a good example of the earlier described setup. The information is fed to an onboard system that will post-process data from the camera, as well as, append data captured from other devices on board such as GPS coordinated or traveled

distance. The vehicle is suitable to be operated in both inside and outside environments.

Figure 6: The scooter used in the collection of outdoor images as in [10].

2.2.2 Correcting the image

The sensors of the Ladybug camera [14], shown in figure 7, are Bayer-mosaicked CCD, thus producing a mosaicked image. A raw mosaicked image contains undesired color effects that can be closely resembled to a ripple or distortion of colors. The mosaicked image contains grid-groups of four pixels that are: two pixels with filtered green, one with blue and the last with red; each pixel contains one spectral value. When each pixel holds the value of one color, obtaining the values for the other colors will be done through estimates according to the values in the surrounding pixels.

Figure 7: Point Grey Ladybug as in [14].

This process is called demosaicking [2, 3], and it is achieved through different algorithms with varying degree of results and side-effects such as bilinear interpolation, gradient based interpolation or high quality linear interpolation. Estimating the other colors through the process of demosaicking produces a better quality image without the color distortion.

Locally adaptive filtering is used on the demosaicked image to reduce the effect of the color component crosstalk. Cross-talk is when a pixel shares its value with its neighbor, essentially a bleed of color across adjacent pixels. Considering the color bleed “noise”, then applying a local adaptive to the image will greatly remove that effect.

Since there are 6 sensors, each can point to an area that has different lighting conditions. This is addressed by having independent exposure controls for each sensor. Later the images of one scene are balanced and aligned according to the measures of exposure.

Finally, to address the visual realism of the panorama the Retinex algorithm is used [12]. Since each image originates from a different/independent sensor, variations in the data held by the image can occur. The Retinex algorithm combines the information from all the sensors, resulting in a global information scope of all the data. The multiscale nature of the algorithm weighs the contribution of each different sensor and helps enhance the overall presentation

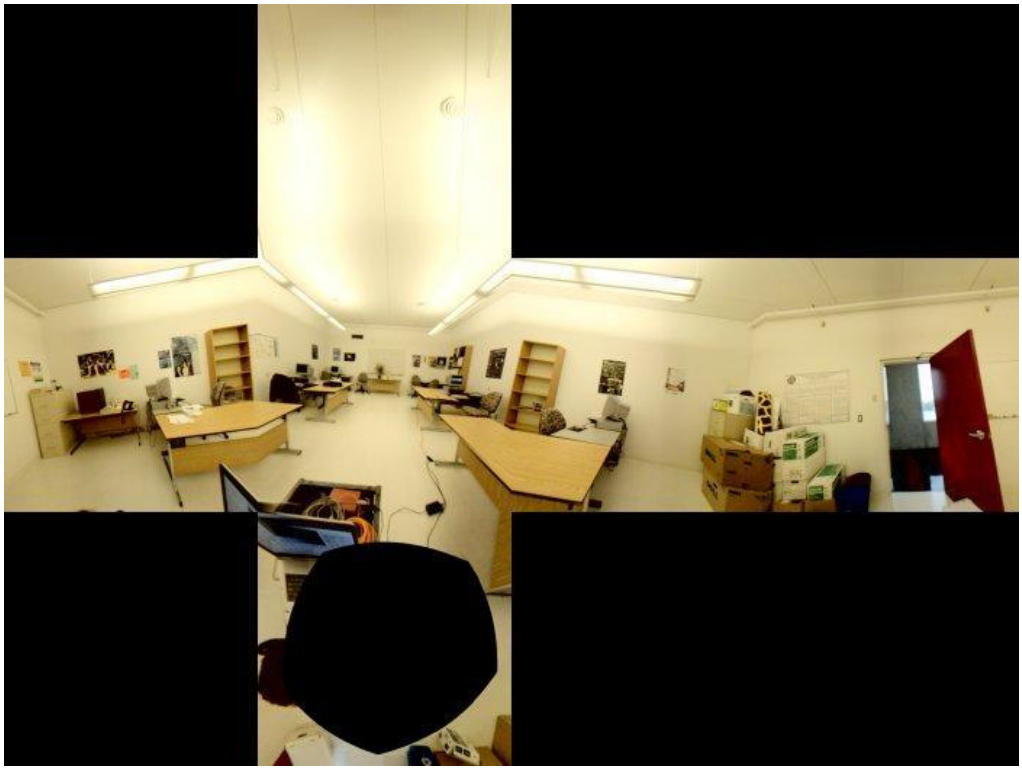

Figure 8: Flat cube with panorama images mapped to its six sides as in [23].

The preprocessing stage is capable of producing multiple image formats. This viewer uses JPG image format. The images produced at the end, to be handled by the viewer, are the six sides of a cube, figure 8. Each represents the actual placement of the camera at the time of taking the images, which are: front, back, right, left, top and bottom.

Although panoramic images can have varying representations such as the spherical, as in figure 11 (equirectangular [4]), cylindrical, as in figure 10, and

cubical as in figure 9, the cubic representation of a panorama is the most effective for our objectives, also it is the best choice for general purpose world projection as argued by Greene in [26]. Unlike cylindrical representation, a cube panorama offers the viewer the ability to look straight down to see the ground or straight up. Another advantage for cubical format is that is formed from six planar images that are easily edited or augmented without transformations.

Also, the size of a cubical panorama is fixed regardless of the size of the vertical field of view contrary to the matter in both cylindrical and spherical representation. Figure 12 shows a graph that relates the size of the panoramic images of a panorama type in relation to the field of view.

Figure 9: Example of a cubic panorama, as in [4].

Figure 10: Example of a cylindrical panorama, as in [4].

Figure 11: Example of a spherical panorama, as in [4].

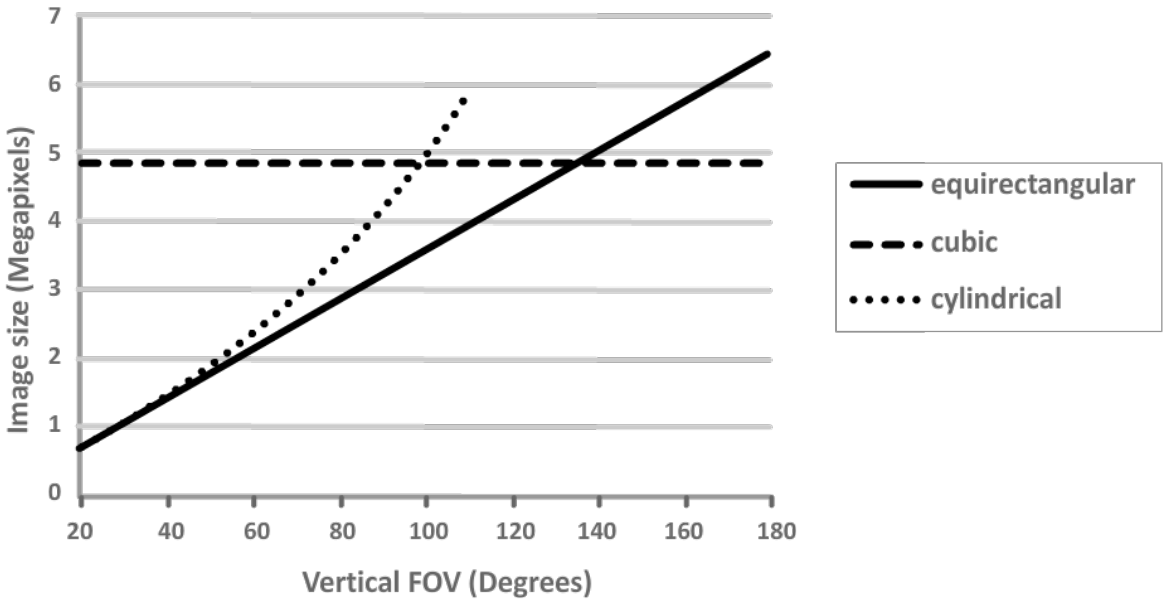

Figure 12: Relation between size and vertical field view for various panorama representations as in [4].

Earlier, we have mentioned how Fiala [1] employed a two component approach to create intermediate views between two actual scenes. Now that cube images are explained, we can shed more light on this approach. In order to produce an intermediate view on the linear path between two panoramas, Fiala proceeds to align the image cubes to the same image plane, then proceeds to calculate the horizontal displacement of the pixels laying on the top, bottom, right and left sides of the image cube based on the placement of

the intermediate view and according to the pre-calculated disparity map with horizontal component only. The Front side will experience radial expansion from the centre of the image, to signify objects coming closer, and the back side will go under a radial contraction to the centre of the image, to illustrate objects going farther. Both the expansion and the contraction of the back and the front of the image are according to a pre-calculated disparity map with radial component only.

2.2.3 Matching the images

After collecting the panoramic images from the environment, the next step is to designate the association between the panoramas. Normally the images of the scene are taken sequentially following a certain path. This will facilitates the direct association, i.e. previous panorama vs. next panorama; however, it becomes inherently difficult to associate panoramas directly if the path of capturing the images crosses itself at a later stage. De Carufel et al. [5] discusses that using classical approaches for matching [15, 16, 17] does not produce adequate results for the following reasons:

- Since panoramas are captured while on a linear trajectory provides a large variation in the point of view, too large to be accounted for using the classical approaches.
- Outdoor panoramas can contain large amounts of open space which can limit the amount of features available to use for matching.
- Outdoor panorama can host a large variation due to shadows, changes in lighting, occlusions and reflections, which in turn can make it more difficult to produce positive matches.
- The initial reasoning in by De Carufel was the non-linear distortion created by cylindrical panoramas, which is not applicable to this document since the panoramas are planar images fitted for a cube.

The approach adopted in [5] supplements the classical approach [15, 16, 17]. It follows the steps of Affine Scale Invariant Feature Transform ASIFT but differs by applying local transformations in the panorama rather than global

transformation. It proceeds by extracting limited field of view images from predetermined directions and then performing multiple view re-projections to match them.

2.2.4 Correcting orientation and pose

Kangni et al. [10] presented a solution that addresses the orientation and pose of the panorama in a set. This work was inspired by desire to create an immersive experience when transitioning from on panorama to another. A panorama is an independent entity when describing the particular view of the environment, but it is logically dependent on its predecessor in terms of its pose and orientation. When a user navigates from one panorama to another, he/she should end up in the same orientation that existed when the transition started.

Although this problem may be addressed by taking GPS readings of the heading when the camera captured the image; however, this is enveloped with a large error margin since its accuracy depends on clear view of the sky, intensive existence of tall building, satellite position at the time of GPS acquisition.
The solution involves two steps; the first is to find the orientation of all the panoramas with respect to a global reference, i.e. true/magnetic north. The second step is to estimate the relative position of aligned cameras relative to each other.

Other work in this domain includes the efforts of Yan et al. [9] in finding the position of the camera from a sequence of images.

2.2.5 Transitioning between panoramas

A noted work that describes an approach to add immersive experience to navigating in a remote environment is the GPU based solution provided by Kolhatkar et al. [6]. In [6], the solution is based on enriching the simple linear interpolation approach by finding the optical flow between two adjacent panoramas with mutual point view, then applying the view morphing algorithm at GPU level to achieve impressive transition at a rate of more than 300 Panoramas (frames)/sec. This work sparked by the need to alleviate the jitter caused by the sudden jump from one panorama to the next, even if within a very close proximity.

This solution achieves high success rate, but falls short when there are moving object in the panorama and when the images are not relatively close.

Dubois et al. in [25] and Shi [27] suggest an approach to generate virtual viewpoints between existing views based on backward ray tracing of a point the demonstrates the highest color consistency.

Other works in this field includes the one done by Speranza et al. [8] which aimed at determining the smoothness of a transition between scenes based on view point density, which represents the number of viewpoints that are traversed throughout the transition from one scene to the next. The smoothness of transition between scenes was found to be dependent on viewpoint density, and the minimum value to ensure the persistence of transition smoothness was 4 views per foot.

2.2.6 Other work

There are many other efforts that have been made to add value to captured panoramas. Earlier in chapter 2 when describing Google Street View, it was mentioned that Google uses laser to collect 3D data about the scene. For a similar purpose, Wojtaszek et al. [7] have proposed a system that, with the

presence of calibrated panoramic images, will describe the scene with sparse but rich 3D representation.

2.2.7 Visualization

In order to bring to light the efforts expended on the NAVIRE project, one must offer a system that allows the end users to gain knowledge and experience of the goals set forth by this project, with ease and through means that are readily available and at their disposal.

This system will allow for the visualization of the remote scenes created through the NAVIRE framework and enable navigation in between related scenes in a seamless fashion. A seamless experience means the system should have the following qualities:

- Easy and simple deployment of the backend portion of the system: with the heavy lifting being done offline by the NAVIRE framework, the system must cater to a deployment with minimal complexity in order to give weight to large scale deployment. Simple deployment minimizes delays and errors.

- Backend portion of the system is multi-platform (platform independent): this allows for a wider adoption of the system and along with the easy deployment will create a favorable system amongst others.

- Flexible communication protocol between frontend and backend: a flexible protocol gives the system the ability to grow and adapt faster with changes in requirements.

- A multi-platform frontend that has systematic performance throughout all the various platforms: a multi-platform frontend has the ability to present the user with the same experience across

multiple devices, and with cross-platform in mind, the frontend will also care that performance is constant with a wide range of hardware specifications.

- Easy and intuitive deployment and use of the frontend: a user friendly frontend will have the advantage over others in gaining approval and having a wider spread amongst consumers since it offers an experience that matches the user expectations, natural reactions and has a low learning curve.

The next chapter, chapter 3, will demonstrate how we go about designing and building the system with the needed qualities mentioned above. Chapter 4 will discuss in details how the frontend portion of the system will put together the pieces that construct the scene and how a complete scene exploration and remote environment navigation is going to function. Chapter 5 will show how the system is measured from a usability x point of view when presented to the user.

Chapter 3: Architecture of the NAVIRE Viewer

The previous chapter reviewed the concept of Navire, the system that contains this project. This chapter will walk us through the architecture and the specifics of the elementary components of this project. The prerequisites of the Navire Viewer are the necessary components needed by the developed application in order to function on any of the targeted platform. It is important to note that the design and placement of these components has been greatly influenced by the multi-platform compatibility requirements mentioned earlier. One of these compatibility requirements is the nature of limited resources available on mobile devices. Mobile devices suffer from the following limitation:

- CPU : as of the time of writing this document, CPU capacity remains an expensive resource. With the various aspects of the OS and other native processes tugging at CPU resources, there is a very limited amount of CPU available for the usage of application. Mobile OS such as Android or iOS automatically terminate any application that exceeds in demand what is available in CPU resources.

- Memory (RAM): very similar to CPU, RAM is a precious commodity in mobile platform, and as in CPU resources, apps that require more RAM than is available are terminated.

- BANDWIDTH: This may vary from one provider to another across the world, but never the less, internet usage on the device is constrained by cost, in some countries, and power usage as 3G/4G,HSPA or Wi-Fi as modems on mobile devices are known to consume large amounts of power to send and receive data.

3.1 Overview of the Architecture

The nature of the architecture is a simple client server setup over internet, as shown in figure 13. The components are a configuration directive, “config.xml”, a collection of images representing the scene(s) to be rendered as panoramas, and a map that relates and links the scenes together and specifies any visual or logical attributes of the scene.

The logical flow of the architecture is an answer to the following design questions:

- What are the images that describe the scene?
- Where are they stored?
- How do images relate to each other to construct a scene?
- How does a group of images that make a scene relate to other scenes?
- How are a scene and its content presented to the user?

The following is primary use case that contains the answers to the previous questions:

- **Use Case Name:** Exploring a remote scene
- **Actors**: Explorer (user)
- **Description**: The user desires to explore various points of a remote scene.
- **Trigger**: The user starts the system.
- **Preconditions**: None.
- **Post conditions**: None.

- **Normal Flow**:
    - The user starts the viewer on their device.
    - The viewer reads from a configuration directive, which is packaged with the viewer at the time of installation, the location of the images that constructs the scene and the map file that describes the scene layout and its content as well as its relationship to other scenes.
    - The viewer sets the location of the images internally.
    - The viewer loads the content of the map file and parses it.
    - The viewer reads the first scene description from the map file content and loads the related images.
    - The viewer reads the content of the scene and its relation to other scenes.
    - The viewer stitches the images of a scene together to create a panorama.
    - The viewer adds to the panorama any other information such as the presence of neighboring scene or augmented objects in the scene.
    - The user interacts with the virtual scene through the input devices available through the used platform.

3.2 Configuration Directive

This is a simple xml file that contains initial configuration information needed by the NAVIRE Viewer to launch. As of the time of writing this document, this file only contains information about the location of the following two components: the collection of images and the map. This file will be the one to accompany the main app throughout the multiple platforms, i.e. in the root directory of the web app, or bundled with the installation app for mobile devices.

With the current information this file holds, one has the flexibility to place the information pertaining to the panorama locally or remotely. While local

placement of the other components provides faster access, it is not always an optimal choice to include large amounts of information with the app at installation which will occupy precious memory space. The location of the other components does not have to be the same. In this directive, a “paths” element will contain two attributes: “images”, which points to the location of the collection of images and “xmlmap”, which points to the location of the xml map of the environment to be explored. A sample of the contents of this file can be found in appendix A.

The xml content of the configuration file shall follow the following xml schema declaration:

```
<?xml version="1.0"?>

<xs:element name="root">

        <xs:element name="paths">

                <xs:complexType>

                        <xs:attribute name="images" type="xs:string"/>

                        <xs:attribute name="xmlmap" type="xs:string"/>

                </xs:complexType>

        </xs:element>

</xs:elemen>
```

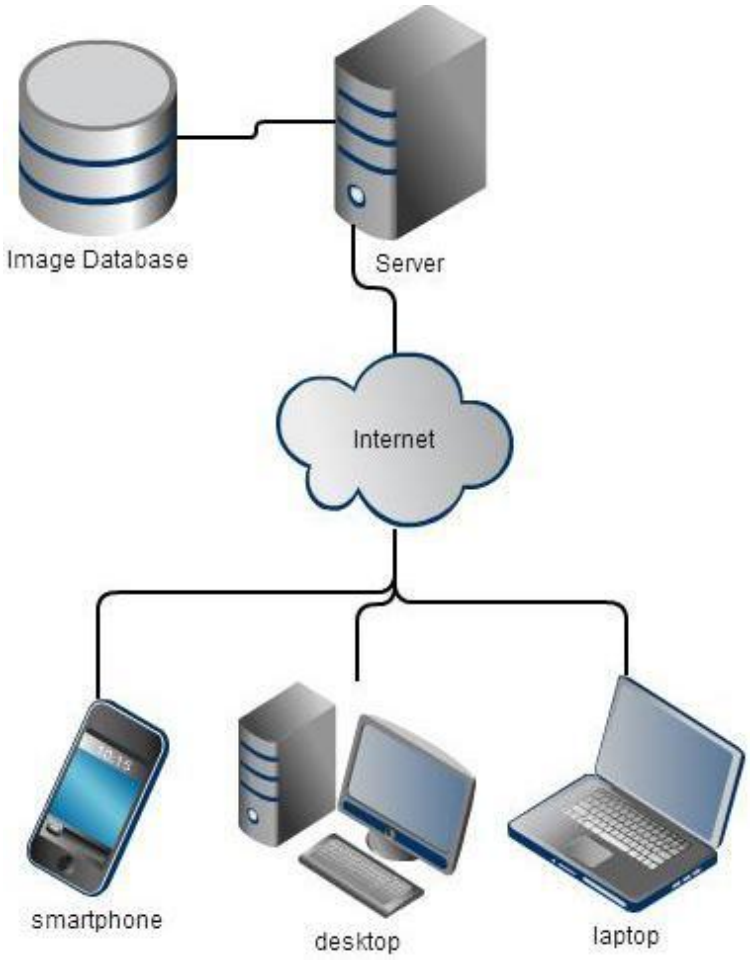

Figure 13: System client server global architecture.

3.3 Collection of Images

This is represented by a folder that contains all the images that resulted from the processing of the environment to be explored. The format of the images is in JPG for best size to resolution value. There is no naming convention and it is left to scene composer to create the proper names for the images that create the cube. There is no theoretical limitation set for the maximum size of the file, but care should be taken that a practical limit should be considered in accordance with the hardware that will host the collection.

Also, one must care that permission are set for “Read” for the folder and its contents.

3.4 Environment Map

The environment map represented by an XML file “panoramas.xml” is one of the main dependencies of the Navire Viewer application. It can be considered as the dictionary that contains pointers to the main assets of the virtual tour as well as the relationship between those assets. Assets here refers to the images that will cover the six sides of the cube, the 2D/3D objects that will augment the cube and any visual elements that will be used to indicate directional material which helps in moving from one panorama (cube) to another.

The XML file is based on a parent-child relationship. The main node is called “root”. This node contains all of the data required to represent the virtual tour. Any child of the node “root” is strictly called “cube”. The node “cube” contains the following elements:

- name: this attribute indicates the name of the cube. For descriptive purposes.
- front: this attribute points to the image, full name and extension that will occupy the front side of the cube.
- back: this attribute points to the image that will occupy the back side of the cube.
- left: this attribute points to the image that will occupy the left side of the cube.
- right: this attribute points to the image that will occupy the right side of the cube.
- top: this attribute points to the image that will occupy the top side of the cube.
- bottom: this attribute points to the image that will occupy the bottom side of the cube.

- lat: this attribute identifies the latitude portion of the GPS coordinates that reference the center of the cube.
- long: this attribute identifies the longitude portion of the GPS coordinates that reference the center of the cube.
- orientation: this attribute refers to how the cube is oriented globally from the true north and clockwise in degrees.

Each “cube” node may contain one or more “neighbor” child nodes. The “neighbor” node refers to a cube directly adjacent to the current one. The only case where a “cube” will not have a child node of type “neighbor” is when it is the only cube in the panorama file. In this case there will be no navigational elements since it’s the only one. “cube” nodes that have one “neighbor” child node can be classified as end or start cubes, although this might not always be the case. However, the first “cube” node in the XML file is the beginning of the virtual tour, and similarly the last “cube” node is the end of the virtual tour. The first and last “cube” nodes in the file are not necessarily the actual start and end of the tour.

“neighbor” node contains the following elements:

- name: this attribute indicates the name of the neighboring cube. For descriptive purposes.
- direction: this attribute indicates the position of the visual navigational element that exists in the current cube and pointing in the direction of the neighboring cube. The “direction” attribute is measured from the normal to the front side of the cube, clock-wise and in degrees. Note that this measurement is local to the cube vs. the orientation attribute mentioned earlier which is measured globally.
- distance: this attribute identifies the distance between the centre of the current cube and the neighboring cube in question. The distance is measured in meters. The purpose of this attribute is to work in conjunction with the “direction” ,“lat” and “long” attributes mentioned earlier or just the “direction” attribute to provide mapping information and give the user of the virtual tour quantitative sense of realism in the tour experience.

This document can hold virtually an infinite amount of cubes, although it is recommended that the size of the file be limited to a maximum that is manageable in terms of download for average internet connection speeds. Large virtual tours can have their descriptive XML divided over multiple files that can be loaded at stages in order not to exhaust or overload the internet connection.

Also, each cube can have an infinite amount of neighbors, but it will be a visual hindrance to have more than 50 neighbors since the navigational indicators will be overlapping causing confusion to the user.

A sample of the environment map can be found in appendix B. The xml in the environment map shall follow the following schema definition:

```
<?xml version="1.0"?>

<xs:element name="root">

  <xs:element name="cube">

      <xs:complexType>

          <xs:attribute name="name" type="xs:string"/>

          <xs:attribute name="front" type="xs:string"/>

          <xs:attribute name="back" type="xs:string"/>

          <xs:attribute name="right" type="xs:string"/>

          <xs:attribute name="left" type="xs:string"/>

          <xs:attribute name="top" type="xs:string"/>

          <xs:attribute name="bottom" type="xs:string"/>

          <xs:attribute name="lat" type="xs:decimal"/>

          <xs:attribute name="long" type="xs:decimal"/>

          <xs:attribute name="orientation" type="xs:integer"/>

              <xs:element name="neighbor">
```

```
                <xs:complexType>
                    <xs:attribute name="name" type="xs:string"/>
                    <xs:attribute name="direction" type="xs:string"/>
                        <xs:attribute name="distance" type="xs:string"/>
                        </xs:complexType>
                    </xs:element>
                </xs:complexType>
            </xs:element>
        </xs:elemen>
```

3.5 The Viewer

The viewer is required to be built with cross-platform compatibility in mind. Adobe Flash lends itself well to this cause. Flash player is supported on all major browsers, and with it being a plugin in the browser, it is almost guaranteed that it will behave the same across all major browsers on all major desktop operating systems. One can also restrict the version of Flash player to be used in order to guarantee complete compatibility with specific builds of the viewer.

On the mobile side, smartphone and tablet; Adobe makes available its AIR technology [34] which enables one to develop for multi-platform devices such as desktops (Mac and Windows), mobile phones and tablets (iOS, Android and Blackberry) and TVs. The tools are available to accommodate specific

builds of the viewer by restricting the runtime environment, similar to the Flash player [35] on the browser side.

The interaction between the user and the device used to run the viewer differs from one to the other. Desktops users utilize input devices such as keyboards and mice to interact with the application, while mobile users use touch screens, hardware navigational buttons and sensor input i.e. compass and gyroscope sensors.

Although not all Flash APIs are common across all the targeted platforms, proper object oriented design using carefully implemented abstraction can provide a solid base that can be extended across all the intended targets.

A major player in building the viewer is PaperVision3D [32, 33], which is a set of robust libraries that utilize the Flash 3D engine to allow developers to write 3D application with the common concepts of 3D science and programming. In this application, PaperVision3D [32, 33] is used to create the three major components used by the viewer:

- a Cube.
- a Plane.
- and a Camera.

The cube is the portion of the viewer where the six images representing the scene are mapped to its sides. Front image to front side and so on. The plane is the portion that holds any augmentation need for the scene. One essential augmentation is the navigational and loading indicators used by the user to navigate from one cube to the next. The plane is also used to contain any 2D media that can augment the view as well, such as images or videos. 3D objects can be used as well to augment the scene, but they are not in the implementation at the moment of writing this document. The camera is a crucial element in the viewer. It is used to respond to the user input and navigate within the scene. In other words, one can imagine sitting in the centre of the cube and looking at in all direction as to explore it. The camera imitates the eyes of the person sitting in the centre of the cube. The camera is rotated about the x-axis and y-axis in order to simulate what a person in

the centre would see. Chapter 4 contains elaborate description of how these components interact in order to offer the user an immersing experience.

Chapter 4: Mechanics of the Viewer

The previous chapter introduced us to the architecture and the components of the tool developed in the context of this thesis. This chapter will discuss in details the process by which the components interact and produce the desired results. The Viewer's main objective is to render the images, collected from a scene, in a sense that is most appealing to the perception of the user. In order to do that, one must replace the absence of the real scene and its dynamics by the closest possible alternatives. Although there are no accurate metrics that measure how realistic a simulation is to the user, we have adopted the following rules:

- The simulated scene must reflect accurately the information it contains within the bounds of the average user physical capacity, i.e. focus, distances…etc. such as:
    - Distance: distance of visible elements of the scene should match what a user experience in real life, i.e. no fisheye effect and no wide angle distortion.
    - Focus: a user does not experience out-of-focus elements in real life since he/she is not looking at it; therefore, the simulated scene must have all elements in-focus.
    - Field of view: In order to allow for a natural experience, the user should not see more or less of a scene from a certain point of view than what he/she would experience in a natural scene.

    - Transition: the speed at which the user can explore the scene should be in the range of the capacity of the user in real life, not faster or slower. Also, the transition from one scene to the other should match as close as possible a real life scenario.
    - Reflexes: the user should interact with the scene and its elements in a natural and intuitive way that substitutes with minimal complexity a real live situation.

- The free-form ability to navigate the actual scene must be replaced by a similar mechanism for the simulated one, i.e. mouse cursor movement to replace head movement.

- The scene will contain native elements; belong to the original scene, and virtual objects. The virtual objects must respect the physical bounds of the scene in static, position and size, and dynamic, rotation and scaling.

- All elements of the scene, native or virtual, if designated as interactive, must reflect their interactivity.

These rules are the result of observation made to industry practice in similar applications, such as the gaming industry, and to conclusions extracted from the usability study done on this project, which will be discussed in details in chapter 5. It is important to note that some of the methodologies used in this project had very popular alternatives and the choice of one aspect over another in earlier was completely arbitrary and it remains open for discussion.

4.1 Global Scope

4.1.1 The fundamentals

Although many of the concepts discussed in this section are hidden behind the scene of the development environment, one must come to an understanding of how they are defined in order to derive the relationship between the final product and the elements that construct it.

The following exploration is based on the general implementation of 3D engines, and specifically of the implementation Flash 3D engine and PV3D. Although some may use different concepts, the variation is very little.

- **3D shapes basics**:

A 3d object is composed in its most elemental form from one or more vertices. A vertex (plural vertices), is a point in 3d space with (x,y,z) coordinates. Vertices connect with each other forming polygons that define the surface of the 3D object. Triangular polygons are the most widely used in 3D engines as a building block to produce 3d surface planes. An alternative to triangle faces is quad faces, but we will only use triangle faces. Figure 14 shows how vertices form triangle faces.

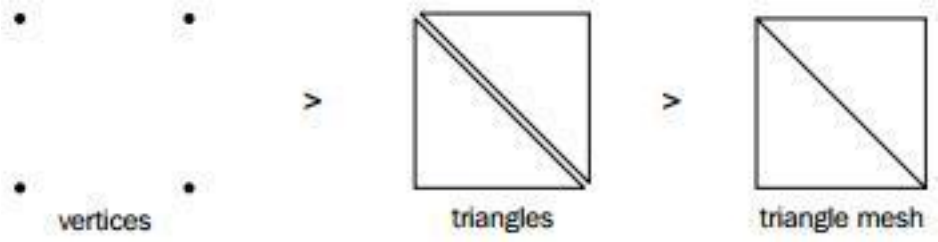

Figure 14: Vertices and triangle faces as in [28].

Combined triangle-faces forms a mesh that will result in a 3d object with height, width and depth, as shown in figure 15. Colors can be added to the surface of each triangle-face, thus adding more realism to 3d object. Figure 15 displays a valid 3d mesh of triangle faces.

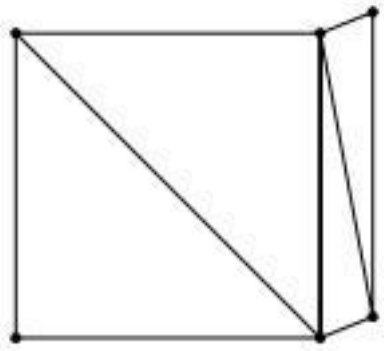

Figure 15: 3D mesh of triangle faces as in [28].

One can start to imagine real-life objects as constructed by a large number of vertices and triangle faces. The larger the number of vertices and triangle faces the more realistic the object is in real life.

In section 5 of chapter 3, we mentioned that the viewer uses three components to produce the scene: a plane, a cube and a camera, while the camera is not really a 3d object in its construction, it does behave as a single vertex that can be manipulated. As for the plane it can be considered the most basic element in the scene since the cube is constructed from combining multiple planes. As per our definition earlier, a plane is composed from two or more triangle faces. We will see later in this chapter how the number of triangles composing a 3d object impacts the scene.

Although it is logical to consider a cube as a combination of six planes, we do treat as a single element since its form creates a dependency between its constructing planes in terms of position and rotation, but we preserve the independency of the underlying planes in terms of the color components.

UV Mapping is the last of our basic components to complete the basic set of understanding 3D, and it is the process by which a texture (image) is projected on a 3d element. A normal image is in 2d and our 3d object has an extra axis. This process can be simply described as flattening out the 3D object, and manipulating the texture to fit the flattened surface, then de-flattening it. The

result is an image that covers all of the surface space of the 3d object, as seen in figure 16.

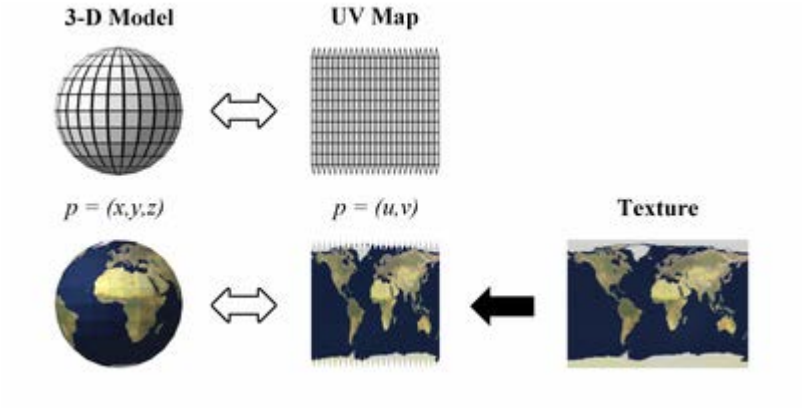

Figure 16: UV mapping illustrated as in [21].

The UV coordinates have a range of 0 to 1 (a percentage). U denotes the horizontal axis of the texture (image) and V represents the vertical axis. Using range values rather than pixels helps in assigning the pixel values in the image to the 3d surface when they do not match in dimensions. For example, if the 3d surface is larger than the image, the pixel values can be duplicated to cover a larger area, while if the opposite is true, then the pixels can be condensed (may be merged) to fit the 3d surface. The following equations illustrates mapping UV to a sphere surface, assuming that the sphere poles are aligned to the Y-axis:

$$u = 0.5 + \frac{\arctan 2(d_z, d_x)}{2\pi} \tag{4.1}$$

$$v = 0.5 - 2.0 * \frac{\arcsin(d_y)}{2\pi} \tag{4.2}$$

- **Motion in 3D:**

Almost any motion in 3D space is a result of the application of one or a combination of the three forms of motion [22]: Scaling, Translation and Rotation usually referred to as the big three.

  - Scaling: is changing the distance between a set of points forming a distinct shape along one or multiple axes causing the distortion of the shape

  - Translation: is moving a set of point from one position to another without impacting the relation of each point to the other, i.e. the shape created by the points remains the same.

  - Rotation: is rotating a set of points around one or more axis.

Figures 17, 18, 19, 20 and 21 denotes the matrices that control the three transformations in 3D and figure 22 shows the complete equation to get the transformed point.

$$Translation = \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 17: Translation matrix.

$$Scaling = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 18: Scaling matrix.

$$Rotation(x) = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & cos\Theta & -sin\Theta & 0 \\ 0 & sin\Theta & cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 19: Rotation about the x Axis matrix.

$$Rotation(y) = \begin{bmatrix} cos\Theta & 0 & sin\Theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\Theta & 0 & cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 20: Rotation about the y Axis matrix.

$$Rotation(z) = \begin{bmatrix} cos\Theta & -sin\Theta & 0 & 0 \\ sin\Theta & cos\Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 21: Rotation about the z Axis matrix.

$$P' = Rotation(z).Rotation(y).Rotation(x).Scaling.Translation.P$$

Figure 22: Transformation operation.

4.1.2 The virtual scene

- **Basic elements:**

From a rendering point of view, the basic elements of our virtual scene are a cube and a camera, as in figure 23. While the cube is exactly what it is, a 3d rendition of a cube; the camera is a representation of the user's eyes, or more accurately what the user sees. The cube will contain the visual elements of the scene, and the camera will produce the render of the portion viewed from the cube to the user. In essence, what the camera looks at is what will be seen on the screen.

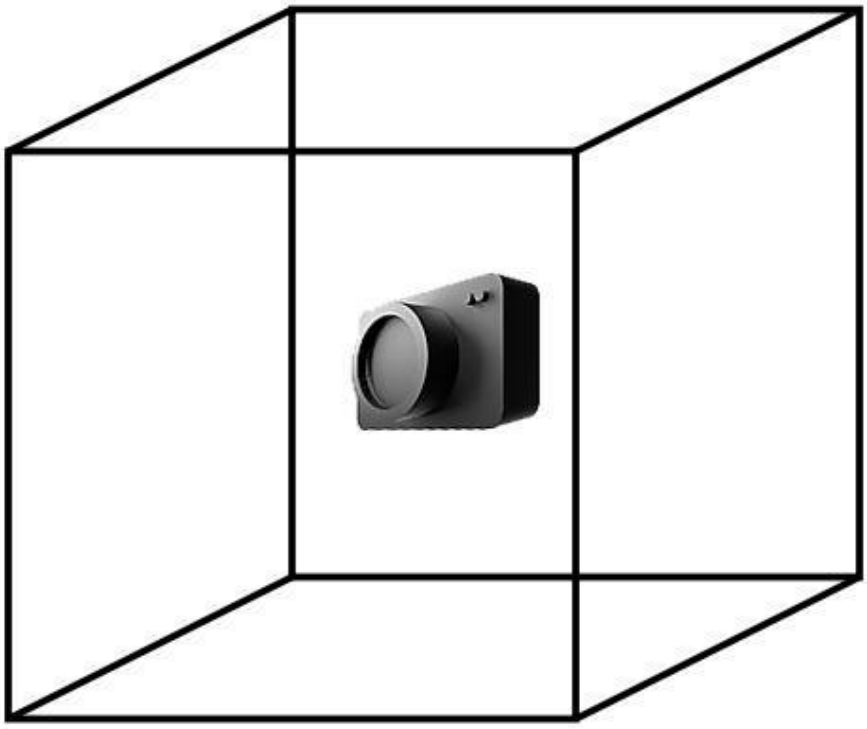

Figure 23: Camera and cube elements of the virtual scene.

As part of the pre-processing operation of the system, the scene is translated into six images; each represents a side of a cube. In the virtual environment, a

hollow cube is created with its sides designated as top, bottom, left, right, front and back. Each of the six images is mapped to the proper side of the cube as to reconstruct the original scene. One must note that in order to reflect the dynamics of the scene accurately, it is essential to have the images within the cube in the upright orientation of, as well as the cube as a whole. In other words, if one is standing upright inside the cube, he/she will see an upright representation of the scene. One can also imagine that the image of the flat cube, seen in figure 8, is folded in to reconstruct a cube and placed upright; that will be the orientation represented in the virtual world. The images in the horizontal view are placed upright, and both the top and bottom images are placed in relation to the other four.

The Camera, as mentioned previously, is not a display element, but a mean to evaluate and render a display element. In order to understand the function of the camera in a 3d environment, one must tackle the concepts of near plane, far plane, frustum and viewport.

In a 3D environment, the near plane and the far plane are two imaginary planes, perpendicular to the camera, parallel to one another and bound the area in which 3d objects are visible. Any display element, beyond the far plane or closer than the near plane, is not visible in the render. The volume that is bound between the near plane and the far plane is called frustum. Any display element that exists within the frustum will be rendered. This process is common in 3D rendering techniques and it is used in order to save processing resources; it is commonly known as culling. For instance, if the camera is moving through a natural scene in a straight line, pointing forward, and is passing scene element, such as trees; any tree behind the camera is out of the view and cannot be seen. Similarly, in a virtual 3D environment, display object that are outside the field of view (frustum, figure 24) of the camera are not visible and any processing that involves their presence is eliminated.

In this thesis project, the viewport is the projection of the viewing volume bound between the near plane and far plane on a plane equals in dimension and position to the near plane. Please note that the definition of the viewport is strictly in relation to the 3D engine used in this project. Other engines will use the far plane coordinates as the projection plane, or a third plane that is halfway

in between, parallel to both the near and far planes and perpendicular to the camera, figure 24.

- **Construction of the Virtual scene:**

As stated previously, the virtual scene is made of a cube and a camera. We achieve a panoramic view of a scene by mapping the images of the scene to their designated side of the inside of the cube, and viewing the scene from a center of the cube through the lens of the virtual camera.

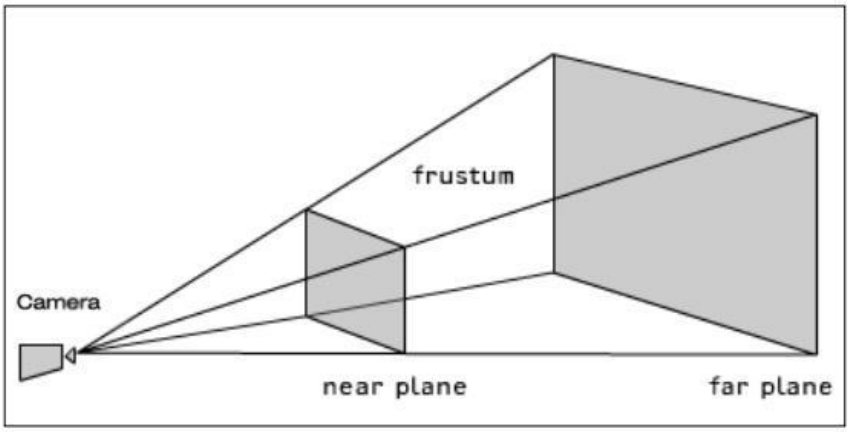

Figure 24: Near plane, far plane and frustum as in [28].

Placement of the virtual scene in 3D coordinated is also of importance to achieve optimal performance. We have chosen to have the center of the 3D assets placed at the point of origin [0, 0, 0], see figure 25. Although the arbitrary global location of the 3D components is not restricted, as long as the relative placement between the components is fixed, we chose to place the elements, globally, at the origin to avoid any unnecessary translation and rotation calculations that will weigh down the performance of the system.

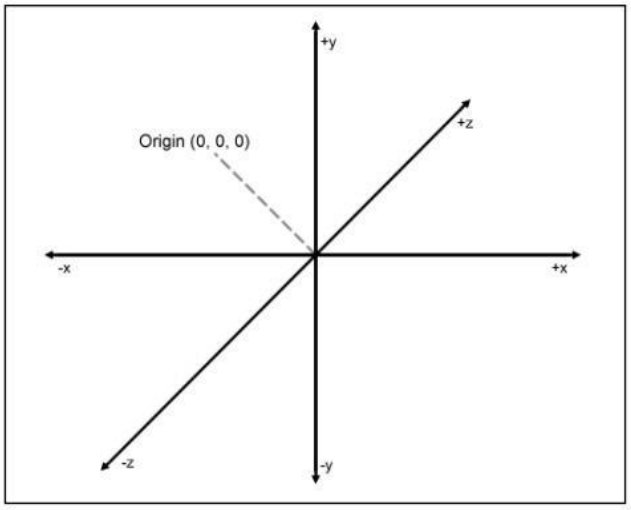

Figure 25: The Cartesian coordinates in the 3D engine of the project as in [28].

The placement of the cube is at the origin of the 3D scene, which means that the bottom half of the cube, if sliced in half parallel to the x-z plane, will be below the x-z plane and the rear half of the cube , if sliced in half parallel to the x-y plane, will be behind the x-y plane. In other words, if we assume the measurement of the side of the cube to be NxN pixels, then the coordinates of the eight edges of the cube will be:

E1 = (0.5N, 0.5N, -0.5N)
E2 = (0.5N, -0.5N, -0.5N)
E3 = (-0.5N, 0.5N, -0.5N)
E4 = (-0.5N, -0.5N, -0.5N)
E5 = (0.5N, 0.5N, 0.5N)
E6 = (0.5N, -0.5N, 0.5N)
E7 = (-0.5N, 0.5N, 0.5N)
E8 = (-0.5N, -0.5N, 0.5N)

Please note that these are the actual dimensions implemented in the system, but they are not restrictive measurement and can be changed to suit the environment. However, one must care to change the values associated with other elements of the scene that have dependency on the size of the cube.

The camera coordinates are the coordinates of the origin (0, 0, and 0). One can start to notice how the placement of the virtual elements in the virtual scene is lending itself to optimized calculations.

- **Movement in the virtual scene:**

For the purposes of this project, we are separating the fundamental movements in the virtual scene in two:

1. *The movement within the scene:* this is represented as rotation of the visual elements.

2. *The movement from one scene to another*: this is represented by changing the images mapped to the sides of the cube to the new scene.

To achieve the first element of movement we were faced with the challenge of choosing what to rotate while fixing the other, the camera or the cube. From a rendering point of view, both options have no effect on the end result, but from an optimization point of view, the story was different.

We aim at reducing the amount of calculation in order to have as less resource-consumption as possible. Rotating the cube vs. rotating the camera will yield the exact same results on all fronts if the scene consisted of only the cube and the camera. Once the scene is augmented with extra display objects, which is inevitable as we will see in section 4.3, then the difference rises. Going through the cube rotation, and fixing the camera, route will only rotate the cube and not any other visual elements within the scene. Each additional element in the scene will have to be rotated separately and relative to the rotation of the cube. The problem becomes obvious when we imagine display elements closer to the camera than the visible side of the cube. The movement of such elements will be slower than the cube as they move on a closer radius. Each separate element will have to be rotated individually and that is an extra cost on the computing resources.

Rotating the camera, on the other hand, presents itself nicely as an ideal solution. One can relate rotating the camera to a realistic exploration of a

scene by human eyes. The scene does not move, but the head of the observer is rotating, while stationary, to explore the scene. All other elements of the scene can remain stationary as we rotate the camera and move the frustum of the camera around. The cost of this is moving one 3d-object regardless of how many others are in the virtual scene, thus obtaining the optimal performance and lowest resource-consumption in this aspect.

In order to achieve the most immersive experience, one must present the movement in the virtual scene in the easiest fashion. For standard viewers, via personal computers, we have chosen the position of the mouse to replace the head movement in the actual scene. Mouse movement can be one or a combination of the following:

- Movement from left to right
- Movement up and down.

One must note that in order to understand the rotation about the 3D axis, we will use a convention that describes rotations about the specific axis as clockwise or counter clockwise by looking down the positive side of the axis.

Up-down movement will be associated with rotating the camera around the x axis. Specifically, moving the mouse from the centre of the screen upwards will cause the camera to rotate counter clockwise about the x-axis causing the display of the upper portion of the scene. Similarly, moving the mouse from the centre of the screen downwards will have the opposite effect.

The rotation about the x-axis is bound between 90 and -90 degrees, since an observer cannot bend his/her head backward to continue exploring the scene. This is illustrated in figure 26.

Right-left movement was implemented in the same fashion. Moving the mouse from the centre of the screen towards the right will cause the camera to rotate around the y-axis counter clockwise, and the opposite for moving the mouse from the centre towards the left.

Rotation about the y-axis is not bound as it is in the axis, since it reflects what and observer is capable of in a real scene. This is illustrated in figure 27.

In order to give ease of navigation and exploration of the scene, we needed to give the user the facility of moving within the scene with varying speed. To accomplish this, we have implemented an equation of the third order to translate the movement of the mouse into rotation. The closer the mouse is to the edge of the screen, the faster the rotation will be, and this applies to both up-down, and left-right movements.

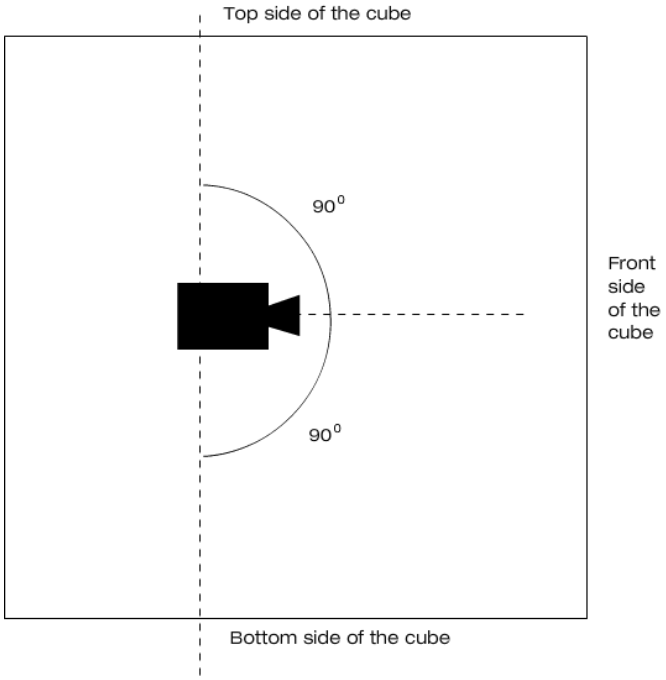

Figure 26: A side view of the cubic panorama illustrating the limitation of camera movements vertically.

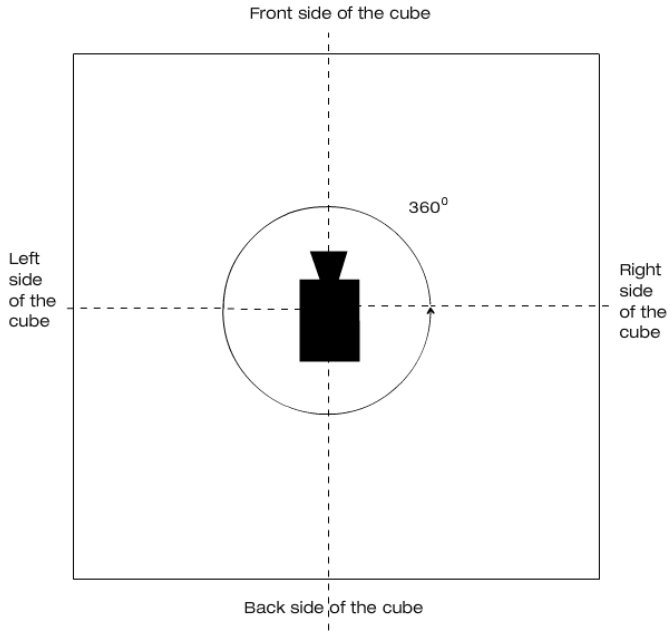

Figure 27: A top view of the cubic panorama illustrating the camera movements horizontally.

An equation of the third order has the quality of an exponential increase in the *p(x)* as the absolute value of *x* increases. The equation that represents the rotation is as follows:

$$p(x)= (c\ (x-(s/2)))^3 \qquad (4.3)$$

x = position of the mouse on the screen.
p(x) = the rotation value, in radians which represents the angle value at which the cube is to be rotated.
c = a constant to control the speed based on the size of the screen.
s = screen width.

Although there is no obvious component for the speed of the rotation, it is implicitly applied as the cube is rotated to the new value from the old. The greater the difference between the two values, the higher the speed at which the cube is rotated and vice versa.

4.2 Mobile Implementation

Although it was emphasized that a cross-platform technology is to be used, one must care that implementation does not translate fully from one platform to the next and that implementation should address the variation of the platforms based on the following criteria:

- Input devices.
- Screen size and orientation.
- GPU vs. CPU processing.

4.2.1 Input devices

Earlier in this section, we have described the virtual scene and the interactions within it, specifically: movement within the scene and from one scene to another.
Since mobile devices (tablets and smartphones) differ in terms of input devices from the conventional desktop/laptop setup, it is important to map the input device interaction to its proper equivalent as one is transitioning from one device to another.

As we have seen in earlier in this section, mouse interactions are the dominant method of navigating the virtual scene. We have mapped the head movement in a natural environment to the mouse movement in order to mimic, as closely as possible, a natural exploration of a scene. A mobile device does deprive the user from a mouse but offers an equivalent in the form of a touch screen as well as motion sensors.

For the mobile implementation, we decided to provide the user with two ways of interacting with the viewer. The first is through using the touch screen to pan through the scene. The second is through using the accelerometer sensor; this sensor reacts to motion and translates it to three values which represent the tilt of the phone about the x, y and z axis as illustrated in figure 31; and detecting the device orientation, i.e. tilt forward-backward and sideways.

The purpose of providing two methods of navigating the scene here is motivated by the nature of mobile devices and the multitude of ways to use them. The ultra-portable nature of mobile devices puts the user in a position where he/she can use the device in any position, and with one or two hands.

Another reason for providing more than one method is that some devices might have not been built with accelerometer sensors included, thus allowing the owners of lower end devices to access and explore the virtual scene with the other method.

One important consideration was the orientation of the device. Since panoramas tend to lend themselves to the landscape viewing aspect, it was determined that for best performance and usability the orientation of the mobile device will be fixed to landscape. Figure 28 shows an actual device in landscape mode in an idle state, where the panorama in not moving.

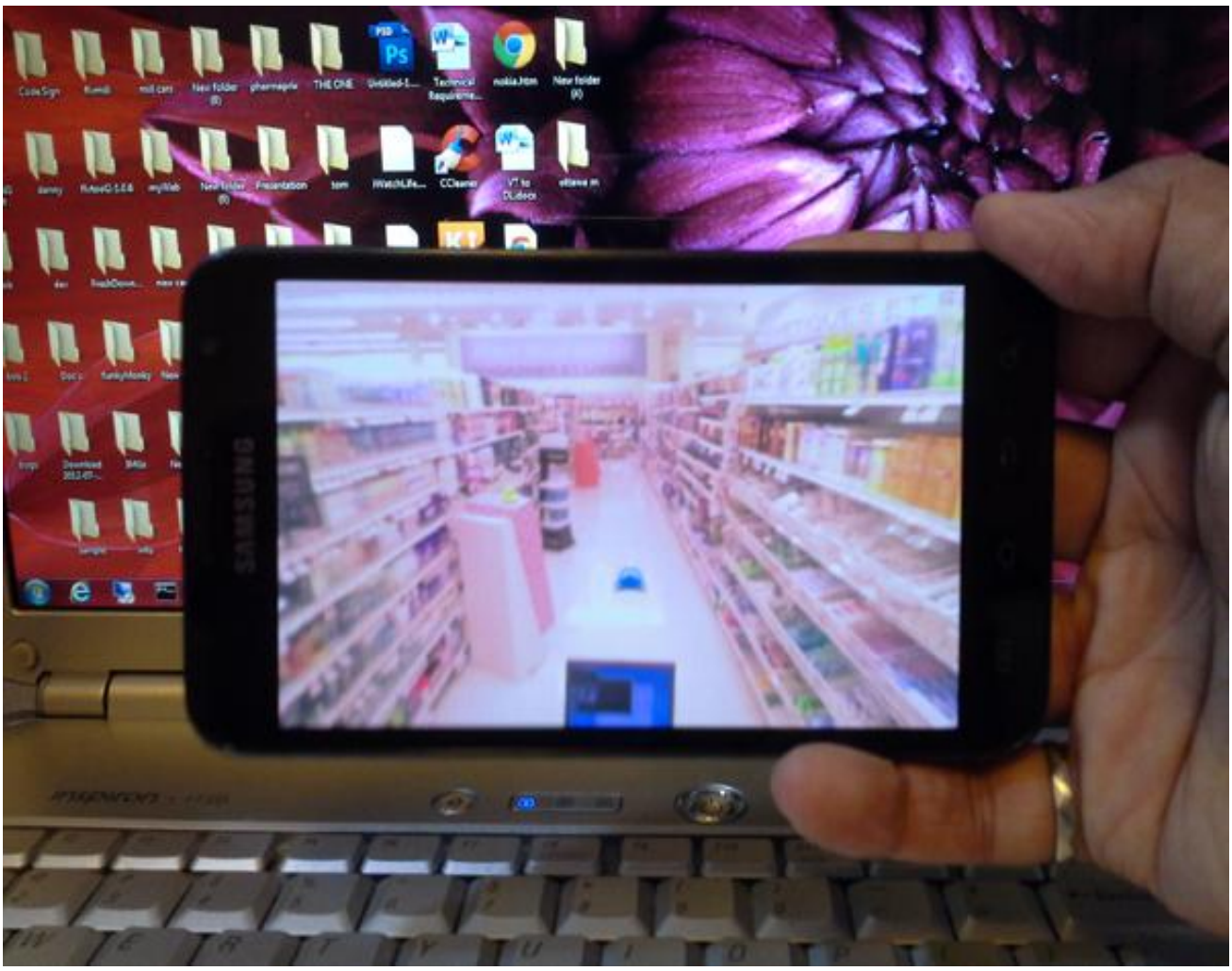

Figure 28: An actual device in idle state where no input from any sensor is detected.

- **Touch screen:**

For the purpose of this project, we have used two aspects of touch screen interaction: touch to replace a mouse click, and a touch-and-slide to replace the mouse click and drag, as illustrated in figure 29.

Obviously, the single touch will be used in two ways: one is to interact with a dynamic virtual object in the scene, such as the marker that loads the next scene; the other is to mark the beginning of a drag action. In section 4, we have seen how the navigation within the scene is done by rotating the camera object around both the x-axis and the y-axis based on the position of the mouse. In a

touch screen approach we analyze the notion of touch and drag on a touch screen to three different components:

- The location of the initial touch.
- The length of the x-component of the drag.
- The length of the y-component of the drag.

The location of the initial touch tells us where the motion will start. The length of the components in the direction of the x axis will translate to the base of an isosceles triangle with its vertex being at the camera. Defining the legs of the triangle as equal to the radius of the circle where the camera sits at its centre and is equal to one half of the length of the cube. The vertex of the triangle is then calculated and it will define the amount of rotation the camera will experience around its y-axis. Similarly, the component in the in the direction of the y axis will be processed and it will yield an angle that describes the amount of rotation the camera will experience around its x-axis. Figure 30 is an example of the touch and drag on an actual device.

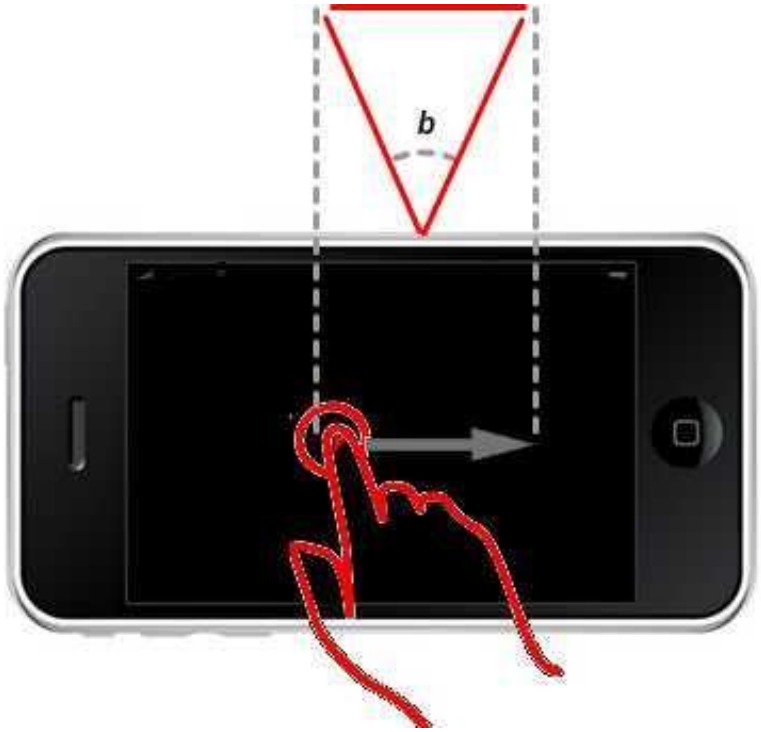

Figure 29: Mapping touch and drag to an angle

- **Accelerometer sensor values:**

Almost every mobile device, nowadays, is equipped with an accelerometer sensor. This sensor provides three values which represent the tilt of the phone about the x, y and z axis, as illustrated in figure 31.

Leveraging the existence of this sensor in mobile devices, we use the values for the tilt of the device about the z-axis representing sideways tilting, and the tilt about the x-axis representing forward-backward tilting. Differing from the previous touch method where there is a stop after the input, this method is closer to the desktop/laptop environment where there is a continuous motion depending on the position of the mouse, instead here there is continuous motion based on the tilt of the device. Figure 32, , is an example of a tilt action on the actual device.

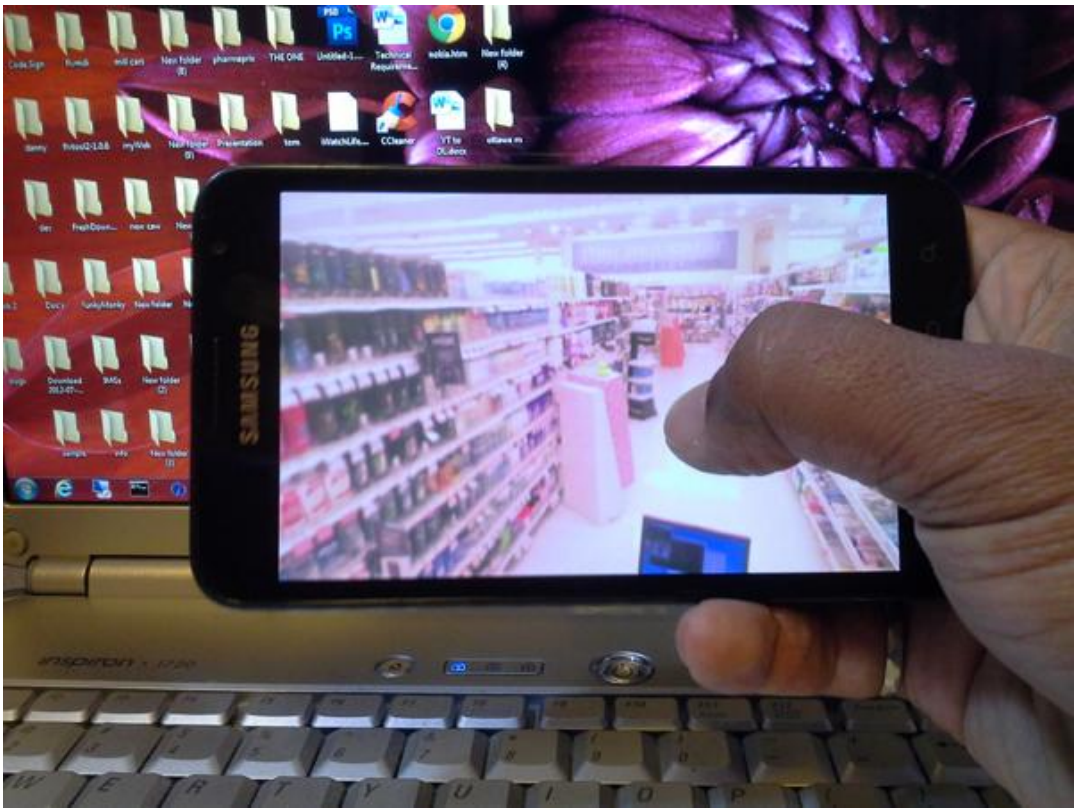

Figure 30: Touch and drag action on an actual android device.

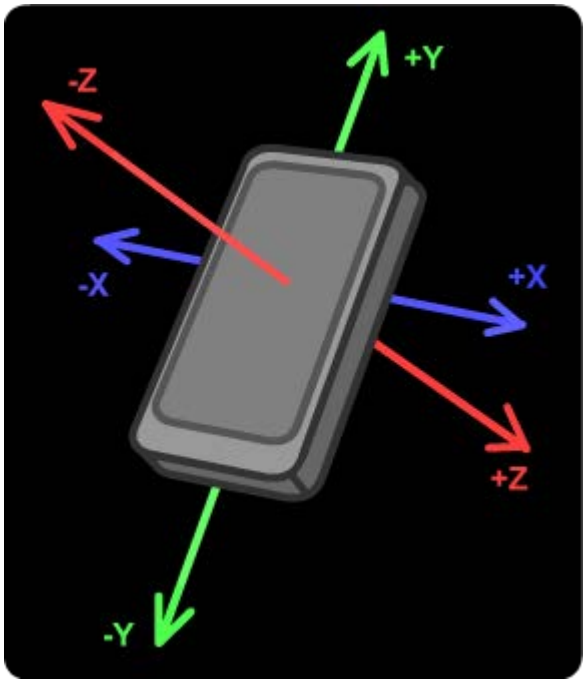

Figure 31: 3D axis definition for a mobile device.

When the device is tilted sideways, the value of the accelerometer in this direction is detected and translated into degrees, which will define the speed and direction of the camera rotation about its y-axis. Similarly, when the device is tilted backward or forward, the perspective value of the accelerometer will be translated into degrees that will define the speed and direction of the camera rotation about its x-axis.

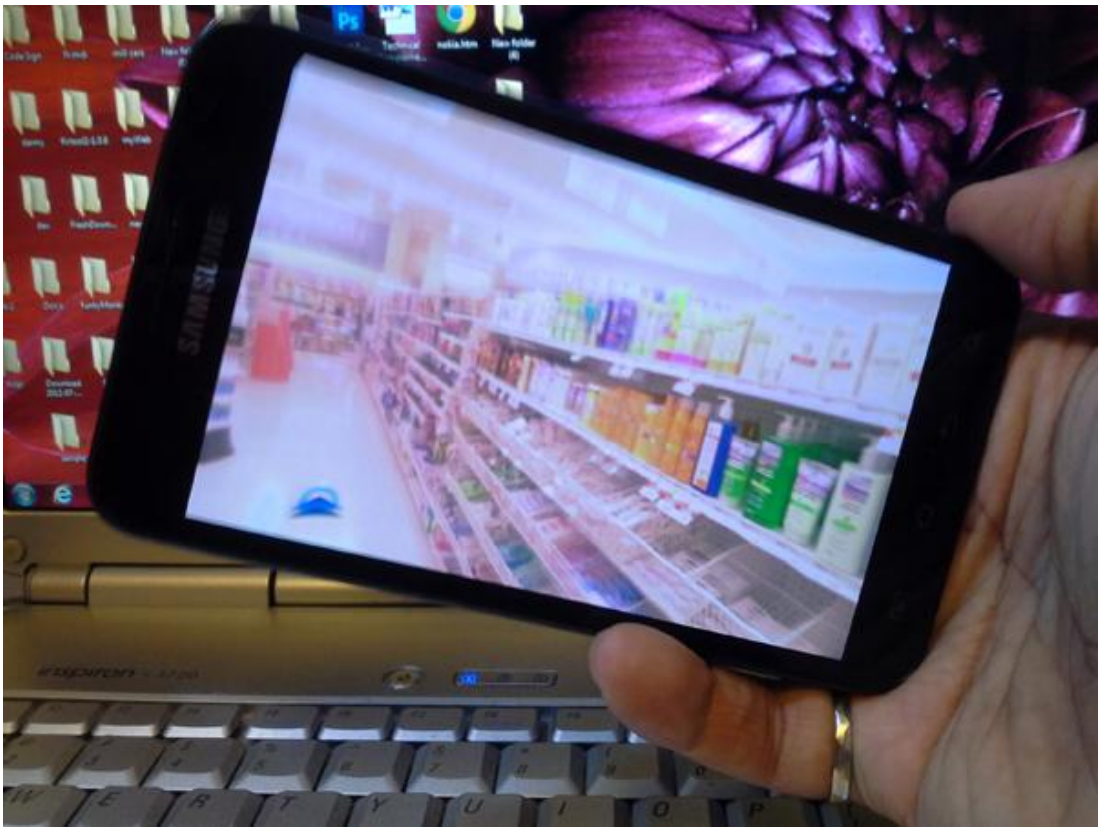

Figure 32: Tilting an actual device triggering a change in accelerometer values causing the panorama to move right.

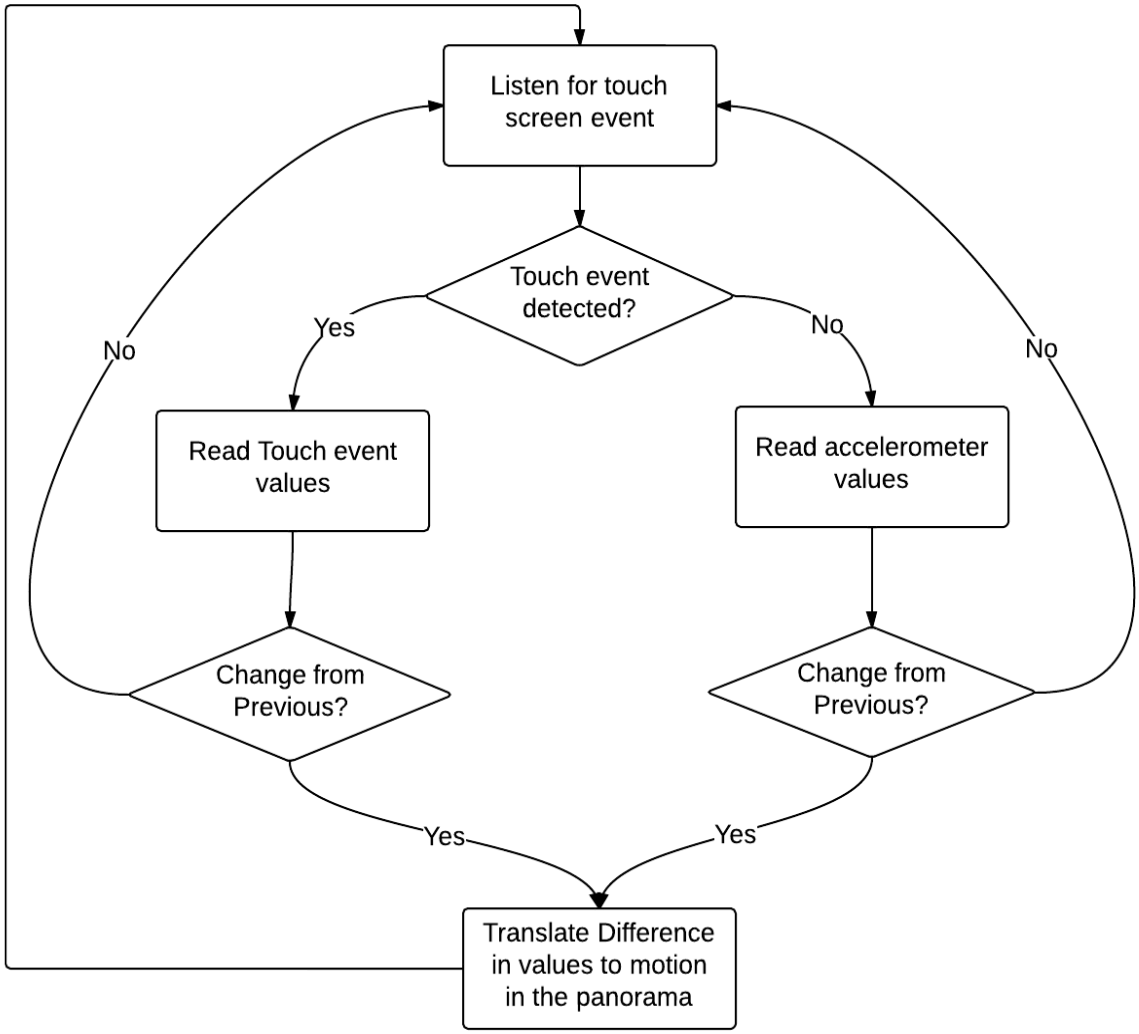

Figure 33: The flow of sensor detection and switch between accelerometer and touch interaction.

Other details in the mobile implementation define the transition from the sensor method to the touch method, and that is simple by touching the screen. When the user touches the screen of the mobile device, reading the values from the accelerometer sensor will be halted, and the algorithm to interpret the pan will start, as illustrated in figure 33 above.

4.3 Augmentation

Augmentation in this thesis involves adding any object to the virtual scene which is not part of the original environment. In this project, augmenting the scene is very flexible and can host a variety of static or dynamic media contents, such as images, audio, video or 3d primitives, as long as they respect the following criteria:

- The augmented object shall not be larger than the scene itself.
- The augmented object will be positioned within the logical bounds of the panorama, i.e. not behind walls or below the floor.

As seen in chapter 3, section 4, the environment map is the blueprint of the virtual scene and its relation to other scenes. It also holds the description of the augmented objects in the environment.

One of the most basic objects that augment the scene is the marker that points to other scenes. It is an interactive 2d button that exists in the scene; its direction and position is set by values in the “neighbor” node that exists as a child of the “cube” node which is the current scene, as seen in figure 34.

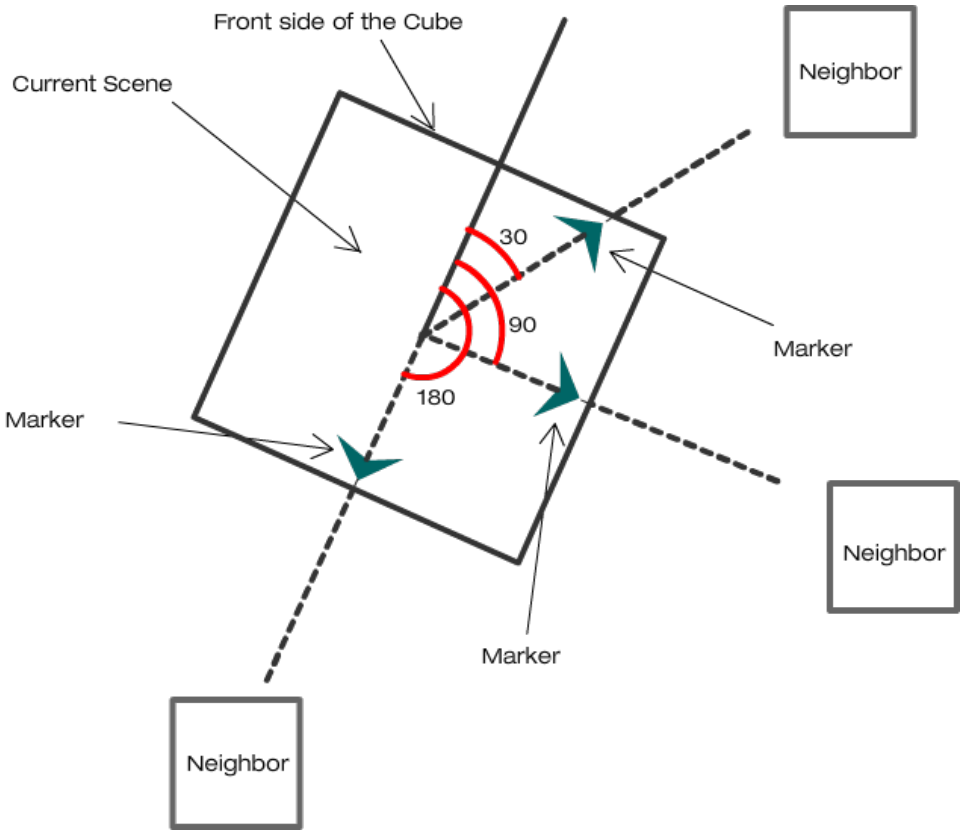

Figure 34: Top view of marker placement based on the position of neighboring panoramas.

For example, the following xml describes a scene with three neighbors:

<cube ... >

<neighbor name="pano0" direction="180"distance="meters"></neighbor>
<neighbor name="pano2"
direction="90"distance="meters"></neighbor>
<neighbor name="pano3" direction="30"distance="meters"></neighbor>

</cube>

In essence, this means that the current scene has three neighbors that that are situated at angles 180, 90 and 30 degrees from the normal to the front side of the cube panorama and turning clockwise. The markers pointing to these neighbors will be placed at 180, 90 and 30 degrees respectively, thus indicating the true direction where the neighbor is located. Figure 34 is a top view that illustrates how markers are placed in a scene based on the previous example.

There are three values that describe the location of an augmented object, as in figure 35, : position, which is distance from the camera horizontally, height which is the distance from the camera vertically and angle, which is the angle at which the object exists with respect to the global north based on the x-z plane (parallel to the camera horizon). Within the child node that describes the location of the augmented object, there exist a “mediaType”, “mediaSource” and “mediaAction” attributes that will specify the content and the actions of the augmented object as per the following:

- mediaType: refers to the content of the augmented object such as image, video, audio or a url referencing other media objects such as other panoramas in remote locations.

- mediaSource: refers to the location where the media is stored.

- mediaAction: refers to the interaction available to the user such as clickable, static, or play_on_load if it is a video or an audio type media.

Unless interactivity of the augmented object dictates otherwise, the augmented object is always facing the camera. Figure 36 shows a snapshot of an actual session of the viewer in browser mode and we can see two augmented objects, the marker and an ad sample for a random brand.

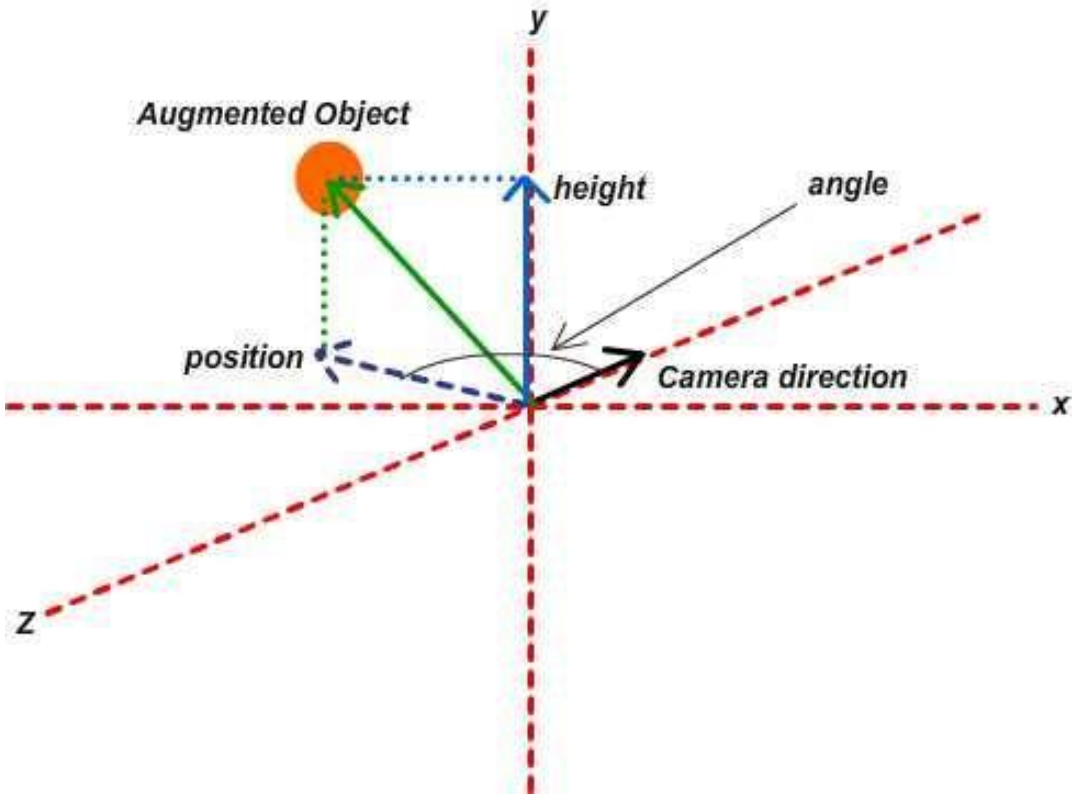

Figure 35: Augmented object detailed coordinate system.

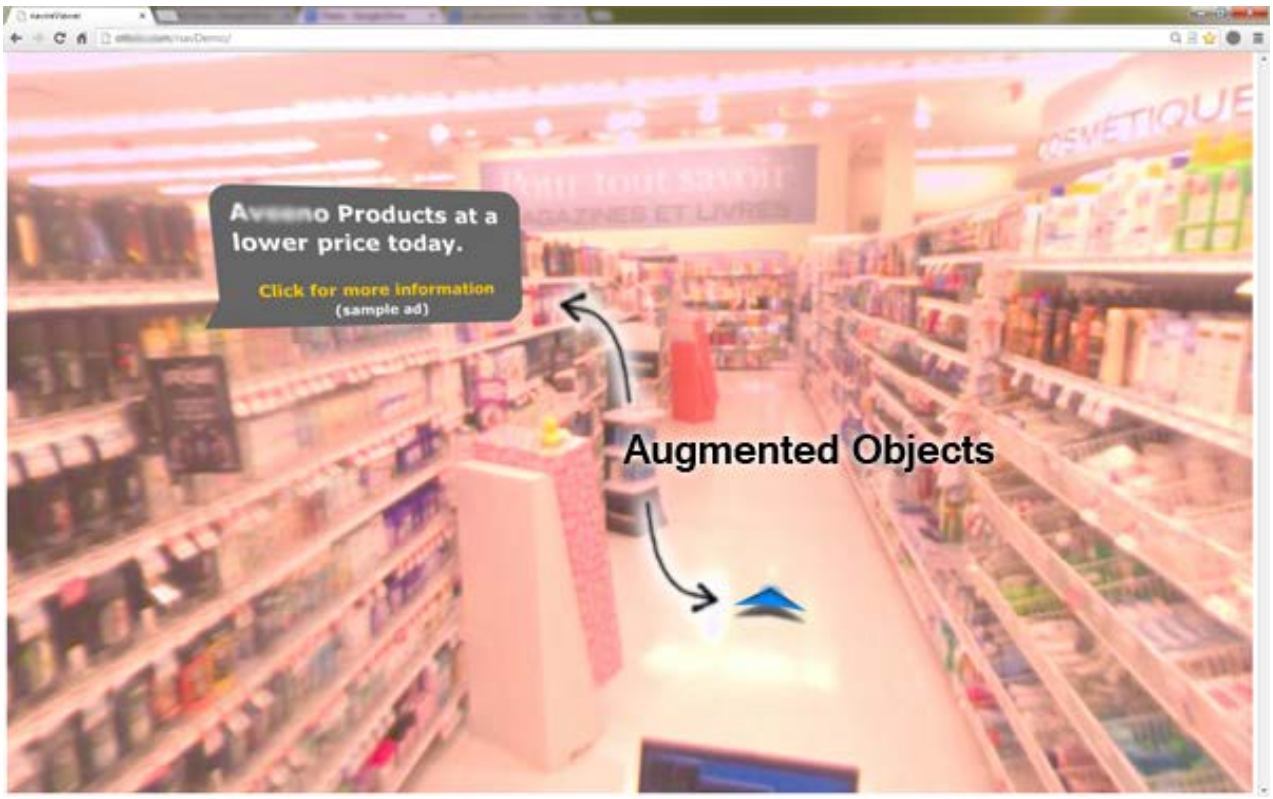

Figure 36: Top view of marker placement based on the position of neighboring panoramas

Chapter 5: Usability Study

The previous chapter discussed in details how the various components of this project interact. This chapter will attempt to evaluate the viewer portion of this project from a usability aspect via the usability heuristics established by J. Nielsen [18], and apply a usability experiment on some of its aspect. It was determined to perform the two evaluations in order to tackle the usability from both a qualitative (heuristics) and quantitative (experiment) measures.

Usability evaluation, in the context of this document, measures how usable a technology is by the audience. The viewer described in this document is not a unique product, on the contrary, it is a one amongst a multitude of applications that have the same purpose, of navigating a remote environment; therefore it's very critical to gain the upper hand in the usability aspect of the offered solution. Usually, amongst competitors, the variations in the basic service are very small, and the product that will win the consumer acceptance, will compensate for its lack of features and leverage on its user-friendly aspect to gain time in focusing the development strategy in the proper direction; all in all, the product that is more user friendly than the others will have its first try at a positive spiral situation, and that alone is a crucial element for new technologies.

5.1 Heuristic Evaluation

Heuristic evaluation as suggested by Nielsen [18], is a “systematic inspection of a user interface design for usability”. Given a set of principles, table 1, the evaluators will walk through the interface trying to find areas where it violates the principles. Nielsen suggests 10 principles that are broad enough to cover the usability of any user interface. Based on a study of the evaluation of 6 projects, Nielsen also suggests that the evaluation be done by 15 or more expert evaluators in order to achieve a 90% or more level of accuracy, or 5 expert evaluators to gain a 75% accuracy [18]; however, due to availability restrictions, this evaluation was conducted by one expert evaluator, which in turn placed the accuracy of finding the problems in the interface in space of 33%.

#	*Heuristic*	*Description*
1	Simple and natural dialogue	Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
2	Speak the user language	Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
3	Minimize the user memory load	Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

4	Consistency	Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
5	Feedback	The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
6	Clearly marked exits	Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
7	Shortcuts	Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
8	Good error messaging	Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
9	Prevent errors	Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
10	Help and documentation	Even though it is better if the system can be used without documentation,

		it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Table 1: Usability heuristics as set by Nielsen [18].

5.1.1 Task selection

The evaluation will cover both the mobile and desktop interfaces. The tasks evaluated are going to be:

1. Navigating within a scene.
2. Interacting with an object in the scene.
3. Moving from one scene to another.

The tasks were selected as such to bridge the usability of the different platforms with the most common use cases, thus achieving higher accuracy of evaluation since there are better chances of finding problems on one platform that was missed in the other. We recognize, and took into consideration, that some variation exists and different platforms can exhibit unique problems.

5.1.2 Task analysis

I. Top goals for the users in this evaluation are:

   a. Perceive the scene naturally.
   b. Able to move freely and naturally within the scene.

c. Able to perceive the interactivity of an interactive object in the scene.
d. Able to perform a stable interaction with an object.
e. Transition smoothly from one scene to the next after interaction.

II. The recursive task division into sub tasks is as follows:

a. Navigating within a scene:

- Launch the application.
- Use available input devices to move within the scene in all directions.

b. Interacting with an object in the scene:

- Launch the application.
- Use available input devices to move within the scene in all directions.
- Find an interactive object.
- Interact with the object with the available input devices.

c. Moving from one scene to another:

- Launch the application.
- Use available input devices to move within the scene in all directions.
- Find the navigation interactive object that allows for the transition.
- Interact with the object with the available input devices to move to the next scene.

III. For all the tasks and subtasks, for the sake of simplicity, we will assume the following:
    a. The user knows the system is populated with more than one scene.

    b. The user has clear indications that previous tasks have been completed throughout the task analysis.

    c. The user forms proper approach to execute the tasks.

    d. User can interpret results to an acceptable level.

5.1.3 Heuristic evaluation of the desktop system:

For each task, the evaluator will use the applicable heuristics to detect problems in the interface, describe the problem, explain it and assign a degree of severity to it. The following are the problems found with the system:

- **Problem 1**

    - *Task*: Navigating within a scene.

    - *Violated heuristic*: # 5; provide feedback.

    - *Problem details*: when the user launches the application, there is a pause before anything is displayed on the screen. This pause duration can vary depending on the size of the loaded assets and the speed/availability of the network connection. Worst case scenario, the user will be left with a blank screen for a long duration.

    - *Severity*: Moderate severity. Although the user might eventually learn that the system is loading through continuous use, the system

should provide feedback about the state and duration of this process.

- **Problem 2**
    - *Task:* Navigating within a scene.
    - *Violated heuristic:* #8; good error messaging, and #9; prevent errors.
    - *Problem details:* when the user launches the application, if the application fails to load the initial assets, either because of a network interruption or a server-down state, there are no indications of what the error is or how to handle the current state of the application.
    - *Severity:* Moderate severity. The user will eventually reach the conclusion that the something is amiss and will react either by reloading the application or checking other application for connection validity; however, this problem is borderline severe since it can leave novice users with the impression that they did something wrong. Therefore, it is highly recommended that good error messaging be implemented.
- **Problem 3**
    - *Task:* Navigating within a scene.
    - *Violated heuristic:* #10; Help and documentation.

- *Problem details:* when the user launches the application, there is no access to a help mechanism that will aid novice/first-time users in using the application.
- *Severity:* low severity. The user will grasp the intended functionality of the system with very few steps since it is not as complex from a usability standpoint; however, help should always be provided to clarify any confusion that might arise from interacting with the various component of the system.

- **Problem 4**

  - *Task:* Interacting with an object in the scene.

  - *Violated heuristic:* #3; Minimize the user memory load.

  - *Problem details:* when the user is presented with an object in the scene, there are no indication prior to interaction to the interactivity of the object, i.e. certain color code or mouse hover interaction.

  - *Severity:* Moderate severity. The user will be able to expend the extra click to investigate whether the object is interactive or not, but this could become tiresome for a scene busy with added objects that vary between static and interactive. The system should follow one of the conventions of designating interactive elements from static ones for ease of use and a better interface experience.

- **Problem 5**

  - *Task:* Moving from one scene to another.

  - *Violated heuristic:* # 5; provide feedback.

- *Problem details:* when the user launches the application and the loading of the initial assets is successful and complete, the interface proceeds to load the neighboring scenes silently. The navigation icons indicate the loading feedback by a gradual change in color; however, if the loading was paused or slowed down, the icon will exhibit the same color code for a while without any information about the status of the loading.

- *Severity:* Low severity. After few uses the user will get accustomed to the speed of the loading in his/her environment, and will adjust the expectation; but it remains beneficial to provide feedback about the loading status.

- **Problem 6**

  - *Task:* Moving from one scene to another.

  - *Violated heuristic:* #8; good error messaging, and #9; prevent errors.

  - *Problem details: similar to problem 4,* when the user launches the application and the loading of the initial assets is successful and complete, the interface proceeds to load the neighboring scenes silently. The navigation icons indicate the loading feedback by a gradual change in color; however, if the loading was interrupted due to network or server fault, the icon will remain in the same state without any indication of what is the current state.

  - *Severity:* Moderate severity. Although the user might react by re-launching the system or by later verifying the internet, it is important to relay proper error messages in order for the user to rapidly acknowledge the situation.

5.1.4 Heuristic evaluation of the mobile system:

Similar to the previous evaluation, the evaluator will use the applicable heuristics to detect problems in the interface, describe the problem, explain it and assign a degree of severity to it. The following are the problems found with the system:

- **Problem 1**
    - *Task*: Navigating within a scene.
    - *Violated heuristic*: # 3; Minimize the user memory load.
    - *Problem details*: when the user launches the application, navigating the scene with side tilts is not easy to perceive. The front and back tilts marry will to navigating the scene up and down, but the side tilt does not do the same thing for the lateral horizontal navigation
    - *Severity*: Moderate severity. The user will get accustomed to the functionality with trial and errors, but it is recommended that other means be explored to express horizontal navigation.

- **Problem 2**
    - *Task:* Interacting with an object in the scene.
    - *Violated heuristic:* #4; Consistency.
    - *Problem details:* The user can navigate the scene with two modes, the touch and pan and the device tilt; however, interacting with

and object in the scene is the same with both mechanisms, which is touching the object. The sensor method does not give the user ease of interacting with the object since the scene is always moving depending on the tilt of the device. The user has to race to interact with the object or adjust the tilt to stop the movement of the scene to facilitate the same task.

- *Severity:* Moderate severity. The user will get the rhythm of the motion of the scene and will adopt his/her favorite mechanism of navigating the screen without transitioning to the other. It is still recommended that other mechanism be explored to lower the complexity and maintain consistency between the two navigation mechanisms.

- **Problem 3**

  - *Task:* Navigating within a scene.

  - *Violated heuristic:* #6; clearly marked exits.

  - *Problem details:* When the user desires to leave the application to a previous interaction that is not directly in sequence, there exist no top level view that the user can retreat to it in order to perform an alternate action.

  - *Severity:* Moderate severity. Although sequential exploration of the scene is the intended functionality. The user should be provided with a global map that will allow him/her to navigate to non-sequential scenes throughout the environment.

- **Problem 4**

    - *Task:* Navigating within a scene.

    - *Violated heuristic:* #7; Shortcuts.

    - *Problem details:* when the user revisits the system, he/she will have to start from the beginning rather than from where they have left the environment.

    - *Severity:* High severity. In a simple environment this might not be of an issue, but in a complex scene the user will be forced to reiterate his/her previous navigation to reach the same point. This will cause high level of inconvenience that might lead to the user leaving the system permanently.

5.1.5 Summary of heuristic evaluation

Since it was done by only one expert, the evaluation of the system done in 5.1.3 and 5.1.4 probably covers only about 33% of the actual number of usability defects, as discussed in [19]. Nonetheless, we observed the capture of multiple problems with varying degrees of causes and severity.

Table 2, represents the number of heuristic violations discovered for each system. We must note that since both systems are an extension of each other for different platforms, violation of core tasks is mutual to both.

System	*Heuristic violations*
Desktop	6
Mobile	10

Table 2: Number of violations per system

Table 3, gives us an overview of the severity of the violations committed by the applications. This is a reflection of the design process, and it is also affected by the amount of features present in the application.

	Severity of violations		
system	*Low*	*Moderate*	*High*
Desktop	2	4	0
Mobile	2	7	1

Table 3: Severity of the violations per system.

Table 4, sheds the light on the heuristics mostly violated amongst all the application. We notice that the most violated heuristics are “Good error messaging”, “Prevent errors” and “provide feedback”.

Heuristic	***Desktop***	***Mobile***	***total***
Simple and natural dialogue	0	0	0
Speak the user language	0	0	0
Minimize the user memory load	1	2	3
Consistency		1	1
Feedback	2	2	4
Clearly marked exits		1	1
Shortcuts		1	1

Good error messaging	2	2	4
Prevent errors	2	2	4
Help and documentation	1	1	1

Table 4: A comparison of the violated heuristic per application.

5.2 Usability Experiment

This section tests the usability of two aspects of the usability of the desktop viewer. There exist two possible representations for each aspect and we wish to measure which is best.

The first aspect is exhibited by the action of the system when the user is navigating the virtual scene and wants to interact with an object in the scene, the scene will either stop moving when the mouse pointer touches the vicinity of the interactive object, or the scene shall remain in motion all the time and the user has to center the mouse pointer and position the interactive object in a no-motion situation. This issue was brought to the design discussion based on a split opinion between users as to which representation interaction is more effective, and this issue was of high importance since this feature defines many aspects of the remote environment exploration.

The opinions were as follows:

- Stop the motion of the scene when the mouse pointer is within proximity of the interactive object.
- Continue the motion of the scene, and allow the user to center the object in the middle of the screen along with the mouse pointer where the scene will be static.

The second aspect is whether to limit the number of navigation methods on mobile to touch and pan method rather than accelerometer sensor method.

5.2.1 Motion and interaction in a scene experiment

The following are the steps taken in preparing for the experiment to test the most accepted interaction with interactive objects in the scene:

1. Developing the hypothesis: The hypothesis developed was: “Users are more comfortable and can perform the interaction task faster when the motion of the scene stopped when in the vicinity of the interactive object”.

2. Picking a set of participants: Five users were picked to aid in performing the experiment. The users were picked from within the class of users most anticipated to use the system. Each user was given a consent form to insure the conformance to the ethical aspect of the experiment.

3. Selecting the variables to test: The independent variable of this experiment was the use of the system with both mechanisms. The dependent variables of this experiment where the ease of recognition and the speed of performing the task.

4. Designing the experiment: The null hypothesis was set to be: “Users are less comfortable and perform the interaction task slower when the motion of the scene stopped when in the vicinity of the interactive object”.

5. Conducting the experiment: Two identical systems were provided with a different interactivity aspect for each. One that stops the motion of the scene when the mouse is in the vicinity of the interactive object and the other is the one where the motion does not stop and the user has to

center the object and the mouse pointer in the middle of the screen. Users were subjected to the two modules randomly and were asked perform an interaction with the first dynamic object they see in the scene. The speed of the response was recorded.

6. Performing the statistical analysis: The measured data is displayed in table 5 and table 6. A t-test [20] was used to statistically measure the results of the experiment.

	Interaction completion in seconds	
User	*Stop motion*	*Continue motion*
User 1	2	3
User 2	1.5	3.5
User 3	2	3
User 4	2	2.5
User 5	1.5	3

Table 5: Measurement of speed of interactions.

Computing the t-test is done through taking the difference between the mean of the first group and the mean of the second group and dividing the result by the square root of (the variance of the first group divided by the number in the group added to the variance of the second group divided by the number in the group).

We have found the following:

- The mean of the first group = 1.8
- The mean of the second group = 3
- The variance of the first group = 0.075
- The variance of the second group = 0.125
- Therefore we conclude that the t-test = 12

Although the results of 5 participants cannot affirm the finding, but for the sake of this document we can affirm that the value of the t in the t-test is large enough to point that the difference between the two groups did not happen by chance.

7. The action: Based on the findings of the statistical analysis, we can conclude that our hypothesis is correct and users are more comfortable and can perform the interaction task faster when the motion of the scene stopped when in the vicinity of the interactive object.

5.2.2 Only the touch and pan mechanism of navigation instead two mechanism experiment

The following are the steps taken in preparing for the experiment to test the most accepted option to explore a scene on a mobile device:

1. Developing the hypothesis: The hypothesis developed was: “Users are more comfortable and can perform scene exploration faster when only one mechanism of exploration is present”.

2. Picking a set of participants: Five users were picked to aid in performing the experiment. The users were picked from within the class of users most anticipated to use the system. Each user was given a consent form to insure the conformance to the ethical aspect of the experiment.

3. Selecting the variables to test: The independent variable of this experiment was the use of the system with both mechanisms. The dependent variables of this experiment where the ease of use and the speed of performing the task.

4. Designing the experiment: The null hypothesis was set to be: “Users are less comfortable and perform scene exploration slower when only one mechanism of exploration is present”.

5. Conducting the experiment: Two identical systems were provided with a different interactivity aspect for each. One that only provides one way of navigating and exploring a virtual scene, which is the touch and pan approach, while the other offered two mechanisms, the touch and pan as well as the tilt approach.
   Users were subjected to the two modules randomly and were asked perform a 360 degree exploration of the scene in both the horizontal and vertical scopes scene. The speed of the performing the action was recorded.

6. Performing the statistical analysis: The measured data is displayed in table 5 and table 6. A t-test [20] was used to statistically measure the results of the experiment.

	Interaction completion in seconds	
User	*Touch-pan*	*Touch-pan and tilt*
User 1	6	7
User 2	6.5	6.5

User 3	6	9
User 4	5.5	7
User 5	6	6.5

Table 6: Measurement of speed of exploration.

Computing the t-test is done through taking the difference between the mean of the first group and the mean of the second group and dividing the result by the square root of (the variance of the first group divided by the number in the group added to the variance of the second group divided by the number in the group).

We have found the following:

- The mean of the first group = 6
- The mean of the second group = 7.2
- The variance of the first group = 0.125
- The variance of the second group = 1.075
- Therefore we conclude that the t-test (approx.) = 2.753

Although the results of 5 participants cannot affirm the finding, but for the sake of this document we can affirm that the value of the t in the t-test is large enough to point that the difference between the two groups did not happen by chance.

7. The action: Based on the findings of the statistical analysis, we can conclude that our hypothesis is correct and users are more comfortable and can perform the interaction task faster when there is only one method of exploration at a time.

5.2.3 Conclusion

The usability study performed on the system has exposed usability gaps through two directions, the heuristic evaluation and the experimental evaluation. The main flaw the usability study suffered from was the low number of evaluators in both instances. Although the results have pointed out clearly the gaps in usability, it remains a fact that a larger audience is instrumental in creating better, clearer and accurate understanding of the result of the evaluation. For example, the heuristic evaluation was performed by one expert leading to finding only 33% of violation in the system as per Nielsen [18]. As for the usability experiment, both experiments were performed with 5 users, which is rather a low sample number. In both approaches it is recommended that a 15 experts evaluate the heuristics in order achieve a rate of discovered violations closer to 99%.

Although the usability study suffered from a small number of evaluators or a small sample size of users, it remained clear the results took unmistakable directions, which can allow us to formulate a clear direction of repair, recovery or improvements to the system.

- Heuristic Evaluation Results:
  The evaluation found many violations in the system. Between the desktop and the mobile systems, there were a total of 16 violations, discovered by one expert. The violations count and severity were as follows:

  - Four low-severity violations
  - 11 moderate-severity violations
  - One high-severity violation

  The high moderate –severity violations number calls for concern and should be addressed.

- Usability Experiment:

  The usability experiments performed aimed at finding the most comfortable mean of using the system. One experiment pointed that the current implementation is the better approach, while the other pointed in the opposite direction.

  Between deciding if interacting with a virtual object in the scene should stop the motion of the scene or not, stopping the scene was the better choice and that is the current implementation.

  The other experiment evaluated the existence of a multi-mechanism exploration of the scene versus a single mechanism. The results came in against having a multi mechanism approach, which is the current implementation. The results of the usability study will be incorporated into future work on the system to enhance and enrich the user experience.

Chapter 6: Conclusion

The work presented in this document is focused on the viewer portion of a much larger system, the NAVIRE framework developed in the VIVA lab of the University of Ottawa. The NAVIRE project starts by using specific devices to capture images, mostly panoramic, of a series of real environment scenes, and then proceeds to pass the images through a series of algorithms such as image correction algorithm (chapter 2, section 2.2), matching the images (chapter 2, section 2.3) and correcting the orientation and pose of the panoramic images (chapter 2, section 2.4).

The objective of the work presented in this document is to make available for the end user a mean of a rich, interactive and immersive experience of exploring remote environments. Our system translates the geo-localized panoramic images, produced by the NAVIRE framework, into navigable virtual environment that is as close as possible to the natural exploration, enriched with virtual objects and can be used throughout desktop and mobile devices.

The challenges facing this work were met and dealt with as follows:

- The first and most noted challenge was to provide the user with the most intuitive way of exploring a scene and allow this intuition to remain intact as the user moved from using a mouse based device to a touch based mobile device with sensors. This challenge was addressed by providing the user with the mechanism of using the mouse on the desktop and the touch-pan mechanism on mobile devices, which is the most relevant to a mouse in touch based devices, as

well as adding the use of accelerometer sensor for an additional mechanism of navigation through tilting the mobile device.

- Create a map that will describe the scene programmatically and have flexibility to describe objects that do not belong to the scene, such as virtual objects. This map will also hold information about the relationship between panoramas, orientation of the panoramas as well as location, distance and other descriptive attributes of the scene. This challenge was addressed by using an XML map that described the scene as a set of panoramas with neighbors and augmented objects as well as attributes of each panorama that described its location, orientation and other relevant values.

- Address bandwidth management for devices where high bandwidth consumption might incur high costs. This challenge was not implemented in the current system, but future work will host a statistical analysis of the user navigation where only relevant neighbors in the direction of exploration will be loaded, rather than all of the neighboring panoramas.

- Handle the preloading of assets to create a smooth navigational flow. This was addressed by loading the neighboring panoramas in the order of their presence in the field of view. This allows for a more continuous navigational experience.

- Deduce optimal user interface design from blending known approaches with general guidelines for designing a cross-platform system. This was addressed by following general interface design guidelines and a heuristic pre-evaluation of the system.

- Explore the technology that will allow the best cross-platform integration with minimum development cycles, without affecting the performance on any of the targeted devices. This was achieved by adopting adobe AIR for developing a system that is multi-platform and PaperVision 3D AS3 libraries for a compatible framework with AIR.

- Design to accommodate the rapid change in mobile device specification and allow for stable performance across all targeted devices with minimum operating system dependency. This was addressed by allowing for a flexible screen management algorithm that allowed the view to fill, without distortion, the target screen. It was also taken into consideration the performance on low-end devices in order to establish a base where most devices can operate the system without restrictions.

We have also performed a usability study for the system in order to evaluate and measure the acceptance of the system from the point of view of an independent audience. Two evaluations were performed, a heuristic evaluation that tested two activities, and usability experiment that tested two mechanisms. The results were clear in all of the evaluations although we may attribute some inaccuracy to the low number of evaluators or participants, which we realize and will address in future work. The heuristic evaluation found many usability gaps that will be addressed in future implementation and modification of the system. The usability experiment resulted in on pro and one con in the implementation; this will also be addressed in future work.

Working on this project has enriched our knowledge in the matter of constructing a virtual exploration system of remote environments. We have experienced, firsthand, the logical structure that such system build upon, and how to standout in a market that hosts impressive solutions such as Google Street View. We have many clear directions to use in future work which allow for a better system, thus a richer, more immersive experience for the end-users.

References

[1] Fiala M.. Immersive panoramic imagery. In *Canadian Conference on Computer and Robot Vision*, pp. 386-391, 2005.

[2] Dubois E., Frequency domain methods for demosaicking of Bayer-sampled color images, *IEEE Signal Processing Letters,* vol. 12, pp. 847-850, Dec. 2005.

[3] Beermann M. and Dubois E., Improved demosaicking in the frequency domain by restoration filtering of the LCC bands, In *Proc. Visual Communications and Image Processing (VCIP 2008), San Jose CA*, SPIE vol. 6822, Jan. 2008, pp. 68221O: 1-10. doi: 10.1117/12.766517.

[4] TourWrist. *Types of Panroamic Images*. Available: http://www.panoguide.com/howto/panoramas/types.jsp, 2013.

[5] De Carufel J.-L., Laganière R., Matching cylindrical panorama sequences using planar reprojections, *ICCV OMNIVIS Workshop*, *Spain*, pp. 320-327, 2011.

[6] Kolhatkar S., Laganière R., Interactive Scene Navigation Using the GPU, *Computer graphics international*, *Singapore*, June 2010.

[7] Wojtaszek D., Laganière R., Peikari H., Peikari M., *Building Sparse 3D representations from a Set of Calibrated Panoramic Images, Symposium on Photogrammetry Computer Vision and Image Analysis* , vol. XXXVIII-3A, Paris, pp. 186-191, Sept. 2010.

[8] Speranza F., Bhatia A., Laganière R., Image Quality in Image-Based Representations of Real- World Environments: Perceived Smoothness of Viewpoint Transitions, *Int. Conf. on Computer Graphics Theory and Applications, Madeira, Portugal*, pp. 439-442, Jan. 2008.

[9] Yan M., Laganière R., Roth G., Estimation of Camera Positions over Long Image Sequences, *ICGST Graphics*, *Vision and Image Processing*, vol. 10, issue V, pp. 39-47, 2010.

[10] Kangni F., Laganière R., Orientation and Pose estimation of Panoramic Imagery, Machine Graphics & Vision, vol. 19, no 3, pp. 339-363, 2010.

[11] Uyttendaele M. Criminisi A., Kang S. B., Winder S., Hartley R., Szeliski R., Image-based interactive exploration of real-world environments. *IEEE Computer Graphics and Applications*, 24(3), May/June 2004.

[12] Rahman Z. –U., Jobson D. J., Woodell G. A., Hines G. D., Multi-sensor fusion and enhancement using the Retinex image enhancement algorithm*. Proc. SPIE*

, Visual Information Processing XI, Vol. 4736, pp. 36-44 July 2002.

[13] Guarnaccia M., Gambino O., Pirrone R., Ardizzone E., An Explorable Immersive Panorama, *CISIS, Sixth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, pp. 130-134, 2012.

[14] Point Grey Specification document, 2013.

[15] Brown M., Lowe D. G., Automatic panoramic image stitching using invariant Features. *International Journal of Computer Vision*, pp. 59–73, 2007.

[16] Roth G, Automatic correspondences for photogrammetric model building. In *20th Congress of ISPRS - International Society for Photogrammetry and Remote Sensing*, pp. 713–720, 2004.

[17] Skrypnyk I. , Lowe D. G., Scene modelling, recognition and tracking with invariant
image features. In *ISMAR*, pages 110–119, 2004.

[18] Nielsen J., Usability Engineering. San Francisco, CA: Morgan Kauffmann, 1993.

[19] Lethbridge T., CSI 5122 Course Notes, 2010, July. Available: http://www.site.uottawa.ca/~tcl/csi5122.

[20] "The T-Test", 2010, August. Available: http://www.socialresearchmethods.net/kb/stat_t.php.

[21] "UV Mapping", 2013. Available: http://en.wikipedia.org/wiki/UV_mapping.

[22] Hartley R. I. , Zisserman A., Multiple View Geometry in Computer Vision. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[23] Viva Lab. Virtual navigation in image-based representations of real world environments. 2006. Available: http://www.site.uottawa.ca/ research/viva/projects/ibr/.

[24] Chen S. E., QuickTime VR- An image Based Approach to Virtual Environment Navigation. In *Computer Graphics (SIGGRAPH'95*), pp. 29–38, August 1995.

[25] Dubois E., Shi F., Laganiere R., Labrosse F., On the use of ray tracing for viewpoint interpolation in panoramic imagery. *In Proceedings of the 2009 Canadian Conference on Computer and Robot Vision*, pp. 200-207, 2009.

[26] Greene N., Environment mapping and other applications of world projections. IEEE Comput. Graph. Appl., pp. 21–29, 1986.

[27] Shi F., Panorama interpolation for image-based navigation. Master's thesis, University of Ottawa (SITE), 2007.

[28] Winder J., Tondeur P., Papervision3D Essentials. Packt Publishing, ISBN 978-1-847195-72-2, first edition, 2009.

[29] Lively M., Professional Papervision3D. John Wily & Sons Limitid, ISBN 978-0-470-74266-2, first edition, 2010.

[30] Photo tourism, Available: http://phototour.cs.washington.edu, 2013.

[31] Google Earth, Available: http://www.google.com/earth/index.html, 2013.

[32] Catadioptric Cameras for 360 Degree Imaging, 2013. Available: http://www.cs.columbia.edu/CAVE/projects/cat_cam_360/

[33] Google Street View, 2013. Available: http://www.google.ca/intl/en-US/maps/about/behind-the-scenes/streetview/.

[34] Adobe AIR, 2013. Available: http://www.adobe.com/ca/products/air.html.

[35] Adobe Flash, 2013. Available: http://www.adobe.com/software/flash/about/.

Appendix A: a sample of the contents of config.xml:

```
<root>

        <paths
        images="http://www.site.uottawa.com/navDemo/"
        xmlmap="http://www. site.uottawa.com/navDemo/">
        </paths>

</root>
```

Appendix B: a sample of the contents of the environment map:

<root>

<cube name="pharm0" front="frame0000_front.jpg" back="frame0000_back.jpg" left="frame0000_left.jpg" right="frame0000_right.jpg" top="frame0000_top.jpg" bottom="frame0000_bottom.jpg" lat="latitude" long="longitude" orientation="0" >

<neighbor name="pharm1" direction="0"distance="meters"></neighbor>
</cube>

<cube name="pharm1" front="frame0001_front.jpg" back="frame0001_back.jpg" left="frame0001_left.jpg" right="frame0001_right.jpg" top="frame0001_top.jpg" bottom="frame0001_bottom.jpg" lat="latitude" long="longitude" orientation="0" >

<neighbor name="pharm0" direction="180"distance="meters"></neighbor>
<neighbor name="pharm2" direction="0"distance="meters"></neighbor>
</cube>

<cube name="pharm2" front="frame0002_front.jpg" back="frame0002_back.jpg" left="frame0002_left.jpg" right="frame0002_right.jpg" top="frame0002_top.jpg" bottom="frame0002_bottom.jpg" lat="latitude" long="longitude" orientation="0" >

<neighbor name="pharm1" direction="180"distance="meters"></neighbor>
<neighbor name="pharm3" direction="0"distance="meters"></neighbor>
</cube>

</root>