

Navigation in Mobile Virtual Environments

The future of mobile devices is aLIVE!

by

Justin Carr

A MASTERS PROJECT

submitted to

Oregon State University

In partial fulfillment of
the requirements for the
degree of

Master of Science

Presented May 21, 2013

Commencement June 15, 2013

Contents

Acknowledgements.....	3
Abstract.....	4
Introduction.....	5
Related Work.....	6
Magic Lenses.....	11
Design and Implementation.....	12
Design Summary.....	13
Design Goals.....	13
Discussion.....	17
Future Work.....	19
Conclusion.....	20
Appendix.....	21
Geometry.....	21
Sensor Sampling Configurations.....	24
Clamping and Filtering.....	27
Test 1 Data.....	28
Test 2 Data.....	30
Bibliography.....	33

Acknowledgements

Thank you to the HCI writing group for your edits and camaraderie in the writing process of this project. It made the process easier to know that others respected my efforts and wanted to help me improve. Especially since that process involved reminding me how commas work.

Thanks also to my friends who supported me and indulged me when I asked them to play with the demo versions of this project and listen to my 45-minute “elevator talks” on this project. I sure hope I haven't scared them off because I will miss them if I have.

Ron Metoyer, thank you for giving a silly philosophy major a chance to do some real research. Your support got my whole graduate experience started and your CS 550 class convinced me that I should be a graphics student.

Carlos Jensen, thank you for your support and encouragement through my time in the HCI group. The HCI group helped me to improve the way I talk about my own work and I think it shows in this project.

Mike Bailey thank you for everything. Your guidance and patience over the course of this project made me feel like it was okay to keep trying to be a graduate student even when I felt convinced that I was not smart enough or driven enough. You helped me to see that sometimes asking dumb questions and exploring what other people might think is a silly idea is probably the most enjoyable part about being a graduate student.

Anjali Menon, thank you for being you! You figured out ways to make me laugh out loud even when this project had me so stressed out I forgot to eat. You also fed me afterward and listened to me read aloud from the first drafts of this paper. I love you so much and I am so thankful for your support on this project and all of my other crazy ideas.

The biggest thanks of all go to my mom and my dad: they supported me in every way they knew how and even some ways that I don't think they know about. They helped me to finance the pursuit of my second degree, they helped me acquire hardware that runs the majority of this project's source code, they loved and supported me while I was frustrated and they never doubted my ability to get it done. I don't think I'll ever really pay them back for all the things they've done for me here and everywhere else in my life. So I hope this project can convey at least a trillionth of my thanks. I am probably the luckiest son alive.

Abstract

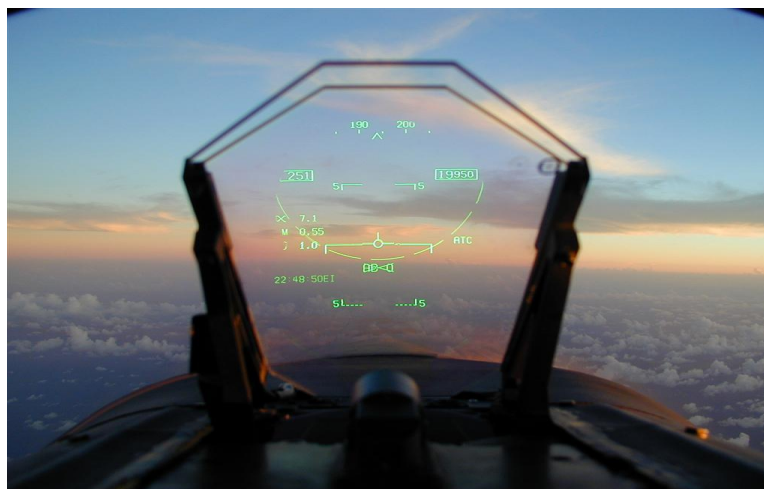
The dream of ubiquitous virtual reality (VR) tools has been a powerful presence in pop culture since before the turn of the century. But for all of its charm, VR applications are few and far between due to the high cost of hardware setups and cumbersome form factor. This project investigates the possibility of bringing VR concepts to mobile devices, which have become relatively inexpensive, travel-savvy computing platforms that are capable of displaying 3D graphics at interactive speeds. The investigation examines related work in the context of the VR community's evolving ontology and taxonomy, builds and tests an example VR interface that leverages the user's kinesthetic sense modality, and discusses a plan for future study.

Introduction

What happened to the dream of virtual reality (VR)? Sometime in the last 20-30 years, popular media turned its eye on VR technology and spectacular ideas about what the future might hold for it began to emerge. These ideas are visible everywhere from novels like *Snow Crash* and *Daemon* to movies such as *TRON*, *Lawnmower Man* and *The Matrix* and they paint dramatic pictures, both hopeful and scary, about how VR could shape the way we experience life. And yet, as inspired as popular culture was by these ideas, as much as we have learned about the benefits of VR technology from existing work, VR applications are relatively inaccessible and uncommon. [2], [3]

Unlike VR applications, mobile devices are quite accessible and common¹, but recent advances in mobile device design have introduced some similarities too. The current generation of smart phones and tablet computers is capable of rendering complex virtual environments (VE) at interactive speeds [4–8] and using built-in micro-electromechanical systems (MEMS) sensors to detect how users manipulate these devices in space. These two features make smart devices appear desirable as potential platforms for VR applications as the combination of these features resembles existing VR applications that use motion-tracked wands and other artifacts. [9–11] If robust, usable VR applications could be implemented on mobile devices, it would allow developers to bring the advantages and inspiration applied VR interfaces offer to a much wider audience.

The differences between traditional VR interfaces and the new interfaces that could be built on mobile devices still represent a challenge in forwarding this idea. To begin with, how does this kind of interface compare to previous work? Milgram's taxonomic framework of mixed reality (MR) systems gives some direction: an application built on a device with a small screen that uses real world interaction would ostensibly have a high level of world knowledge, a low level of reproduction fidelity and a low level of presence. Such an application would either be an augmented reality² (AR) interface or an augmented virtuality³ (AV) interface depending on how the application represents its VE. [1]



This aircraft HUD is an AR interface. Information such as the attitude, altitude, and heading of this plane are superimposed onto the pilot's view of the outside airspace [picture reference: <http://htka.hu/wp-content/uploads/2010/04/HUD.jpg>].

- 1 Gartner, Inc. estimates that over 821 million smart phones and tablet computers were sold world-wide in 2012 and predicts that sales will exceed 1.2 billion in 2013. [Gartner]
- 2 Augmented reality interfaces complement real world visuals, such as camera feeds or objects seen directly without intermediate tools, with virtual, digital artifacts, such as waypoints. [1]
- 3 Augmented virtuality interfaces complement VE visuals, such as digital terrain maps, with real world visuals or object interaction, such as camera feeds or real world objects. [1]

However, this taxonomy only describes the visual aspect of the interface, what about the haptic aspect? Secondly, do the sensors and graphics hardware shipped with these devices perform well in this kind of interface? While graphics benchmarking tests indicate that they do [4–8], the author could not find mobile device research that evaluated the performance of the onboard sensor hardware in VR or AR interfaces.

This project explores the challenges in developing VR interfaces on mobile devices by addressing the question: “Are current-generation mobile devices suitable platforms for VE navigation applications?” To do this, this paper surveys the history and language of VR research, visualizing a more open-ended understanding of immersive technology based on suggestions by Bowman and McMahan in *Virtual Reality: How Much Immersion Is Enough?* [3]. The paper then describes the implementation and testing of a proof-of-concept interface, named “a Low-Immersion Virtual Environment” (aLIVE) on an example mobile device. The paper closes with suggestions for future work to realize a more robust version of aLIVE.

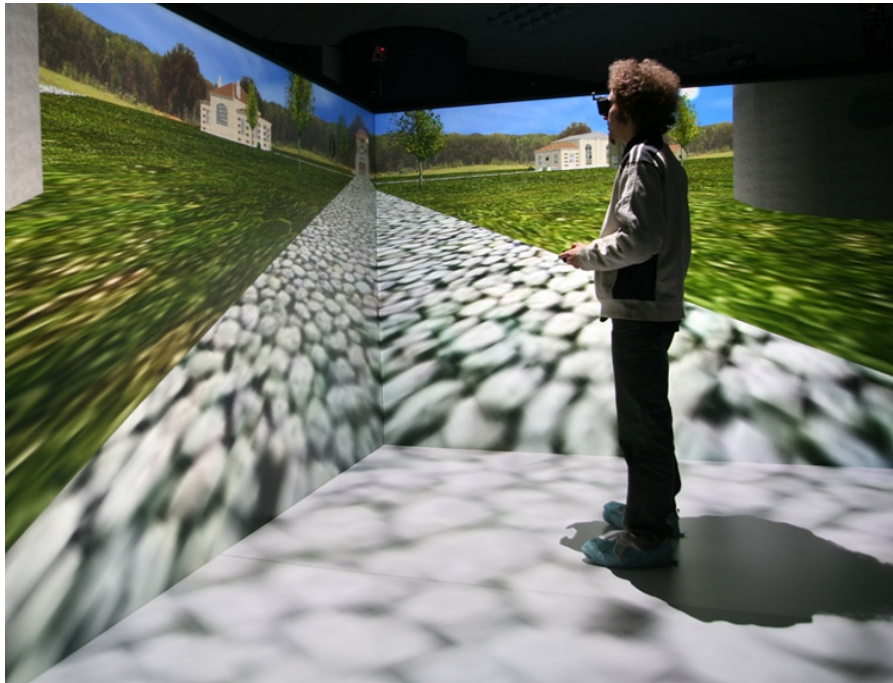
Related Work

There are many examples of successful traditional⁴ VR interfaces in the wild: flight simulators [12], video games [13], amusement park rides [14]. Yet they are far from ubiquitous. The academic community and popular culture's fashion sensibilities tend to agree this is largely due to the costs of preparing and using such an environment and the equipment's burdensome form factor. Jesse Fox et. al. summarize this sentiment in “Virtual Reality for Social Scientists”:

Because of the claims of many futurists in the early 1990s, when people hear the words “virtual reality,” it is often with a dose of skepticism and technological trepidation: What happened to that bizarre world where everyone sits at home and experiences life in a funky helmet? The fact is that much of the high-end virtual reality technology featured in these futuristic fantasies has not diffused as quickly as other emergent technologies (e.g., cellular phones) because it remains too costly and cumbersome for everyday use. [2]

To illustrate this, consider the requirements for a CAVE Automatic Virtual Environment (CAVE). Since a CAVE is essentially a closed room in which the walls have been converted into a single, continuous display surrounding the user, this UI would require a dedicated usage space, a set of wall-mounted screens or projectors and some means of providing input to the application.

4 There are two ways the word “immersion” has been used to talk about VR applications. The first is a continuum describing the extent to which the users senses are incorporated into a VR application's control loop. The second is a set of hardware and software configurations that an interface must have before it can be considered a VR interface. This set of configurations includes head-mounted displays, CAVEs, and other techniques for helping a user feel immersed in the VE. For clarity, this paper will refer to this second definition as “traditional VR”.



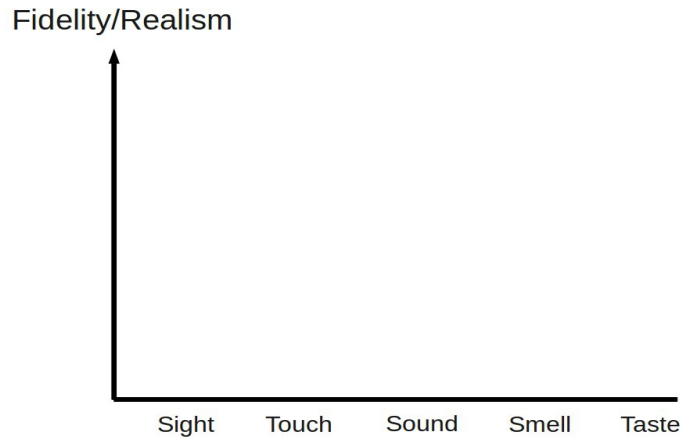
CAVE display by Antycip Simulation [http://www.antycipsimulation.com/files/uploads/UTBM_Cave_Yview.jpg]

For private consumers, acquiring a CAVE would require obtaining these parts, then assembling and calibrating the environment by hand or contracting with an organization such as Mechdyne (www.mechdyne.com) that specializes in implementing CAVEs. In addition to these resource costs, traditional VR interfaces may also create health and safety costs such as motion-sickness, loss of visual acuity, and disorientation. [15] When all of these costs are taken into consideration, traditional VR interfaces do not seem practical for most everyday applications.

If the cost of these traditional VR interfaces is so high, what is it we are paying for exactly? In the common case, immersion and presence. In his article, “A Note On Presence Terminology” Mel Slater defines “immersion” as the objective, measurable outputs that a computer system uses to create the feeling of being in a VE. These outputs are connected to our sensory modalities in such a way that they create what Slater calls “presence”, the subjective, immeasurable, psychological response to the VE. [16] In an ideal system, a user's presence will allow them to make intelligent decisions about how to interact with the VE, closing the control loop. Leveraging the combination of immersion and presence in these interfaces is a powerful technique because it allows the user to intuit ways to interact with the VE.

For anyone interested in using immersion, presence or other concepts that have come from VR but don't have the resources to use traditional VR interfaces, there are two options: find a way to lower the cost or re-think what VR offers. In “Virtual Reality: How Much Immersion Is Enough?”, Bowman and McMahan discuss the VR community's narrow focus on the combination of immersion and presence in virtual environments, pointing out that by defining immersion as a “multidimensional continuum” the VR community may find other advantages, besides presence, to leverage in VR applications. Some of the benefits hypothesized in their work are: improved spatial understanding, reduced information clutter, and increased peripheral awareness. [3] Bowman and McMahan do not indicate what the dimensionality of this immersion continuum might look like, but even without much detail we can see how this idea changes the VR design space. To illustrate this, let the immersion

continuum be defined by a plane with the human senses along the X-axis and relative fidelity⁵ along the Y-axis:



This version of the immersion continuum, created in this paper to visualize Bowman and McMahan's point, is a qualitative representation of the design space that only takes into account two of the many possible dimensions we could use to describe VR.

In this representation of the immersion continuum, if Application A is more immersive than Application B, A will have higher-level fidelity scores in more sense modalities. Thus, A will have a “footprint” with a larger area than B. Additionally, A and B will likely have footprints with different shapes depending on the techniques they use to interact with a user's senses. Note that fidelity must be taken in a relative sense as the qualia a user experiences in a VR application can be difficult to quantify: How do we define high-fidelity smell? Which gives better visual fidelity: a head-mounted display or a CAVE? Below are some footprints for existing VR applications.

Relative Fidelity Scale⁶:

- 0 :: Not Used
- 1 :: Low
- 2 :: Medium
- 3 :: High

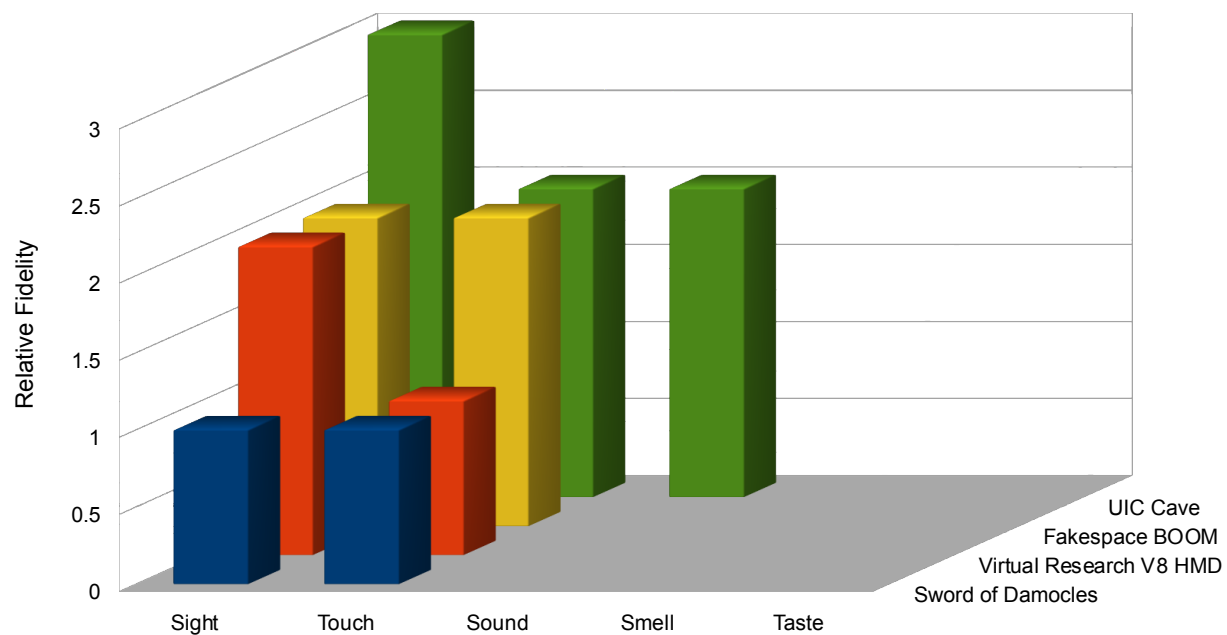
Project	Sight	Touch	Sound	Smell	Taste
Sword of Damocles [17] ■■■■	1: HMD with wireframe graphics	1: head-tracked perspective	0	0	0
Virtual Research V8	2: HMD with 640x480	1: head-tracked perspective	0	0	0

⁵ For this discussion, fidelity refers the level of detail that an application can represent in a given human sense modality. For certain applications this is synonymous with realism.

⁶ These fidelity values are not determined by research. They were created from performance descriptions and screenshots of each project for the purpose of illustrating the state of the design space.



Project	Sight	Touch	Sound	Smell	Taste
HMD [18] ■ ■ ■ ■ ■	resolution				
UIC CAVE [18] ■ ■ ■ ■ ■	3: 1280x1024 resolution	variable depending upon input device	variable depending upon application	0	0
Fakespace BOOM [19] ■ ■ ■ ■ ■	variable depending upon “display head”	2: head-tracked HMD with wide range of motion	0	0	0

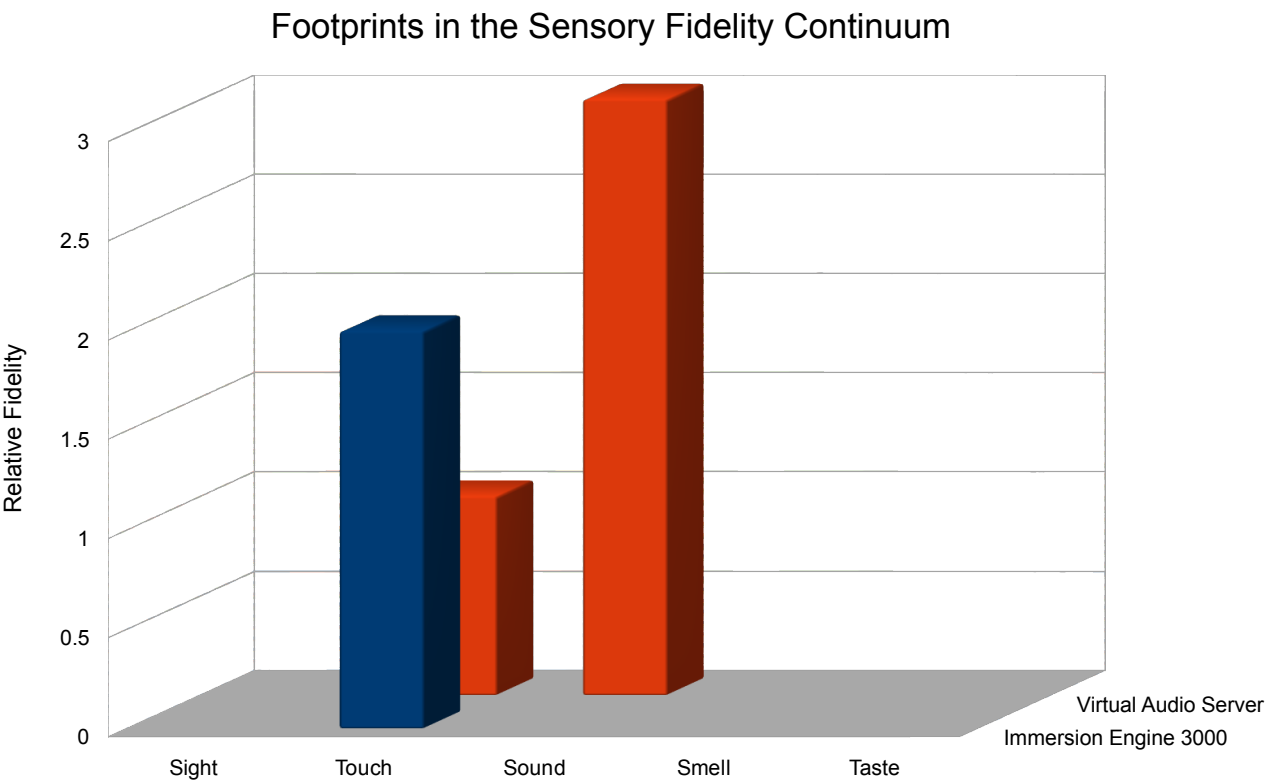
Footprints the Sensory Fidelity Continuum




While this coarse visualization oversimplifies years of hard work in pushing the boundaries of what we mean by fidelity, we can see that these projects are in the traditional VR style and do not occupy much space in this continuum of immersion; focusing heavily on visual fidelity. This visualization implies that there is still a range of senses to explore and a plethora of undiscovered techniques for creating different stimuli for these senses. Below is a list of other projects that have begun to explore other senses and techniques, breaking away from traditional VR:

Project	Sight	Touch	Sound	Smell	Taste
Virtual Audio	0	1: head-tracked	3: Headphone-	0	0

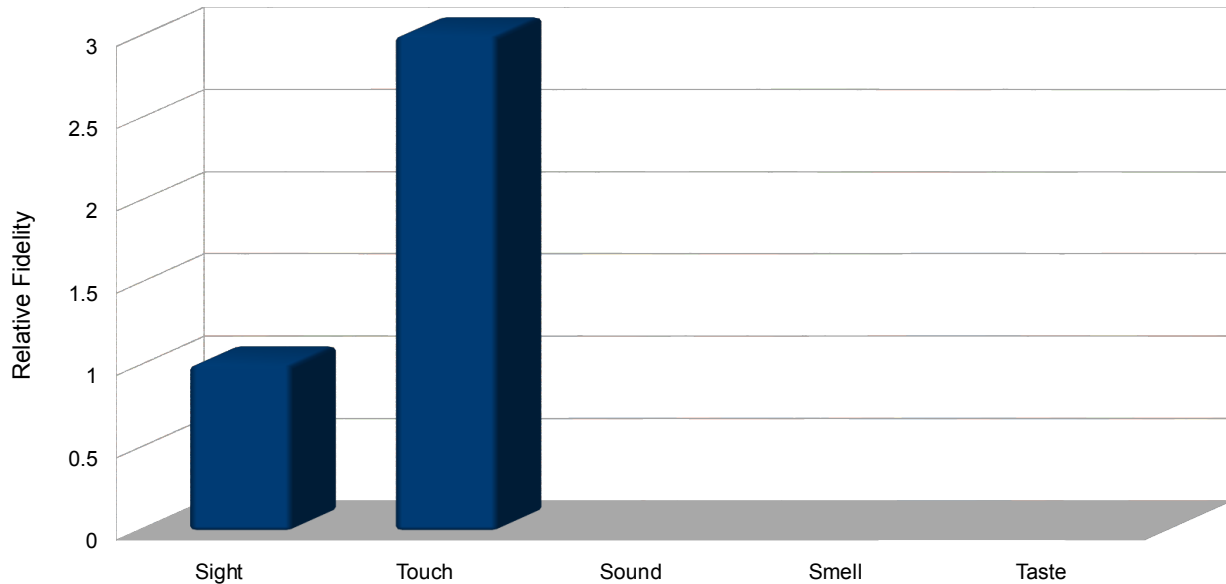
Project	Sight	Touch	Sound	Smell	Taste
Server [20] 		perspective	based stereophonic output		
Immersion Engine 3000 [21] 	0	2: force- feedback stylus	0	0	0



Bowman and McMahan go on to suggest that, while immersion has been shown to increase a user's ability to complete a wide variety of tasks, a given application may require different levels of immersion or presence, providing examples of existing applications that do not require presence. The project featured here, a Low-Immersion Virtual Environment (aLIVE), seizes upon Bowman and McMahan's idea by creating a VE-exploration application that occupies a small part of the user's visual field and incorporates the control device with the screen as is necessary with a mobile device. The result is an application that must be used while the space surrounding the user is clearly visible and the entire apparatus can move freely through space:

Project	Sight	Touch	Sound	Smell	Taste
aLIVE 	1: tiny screen	3: 3-axis accelerometer and gyroscope	0	0	0

Footprints in the Sensory Fidelity Continuum

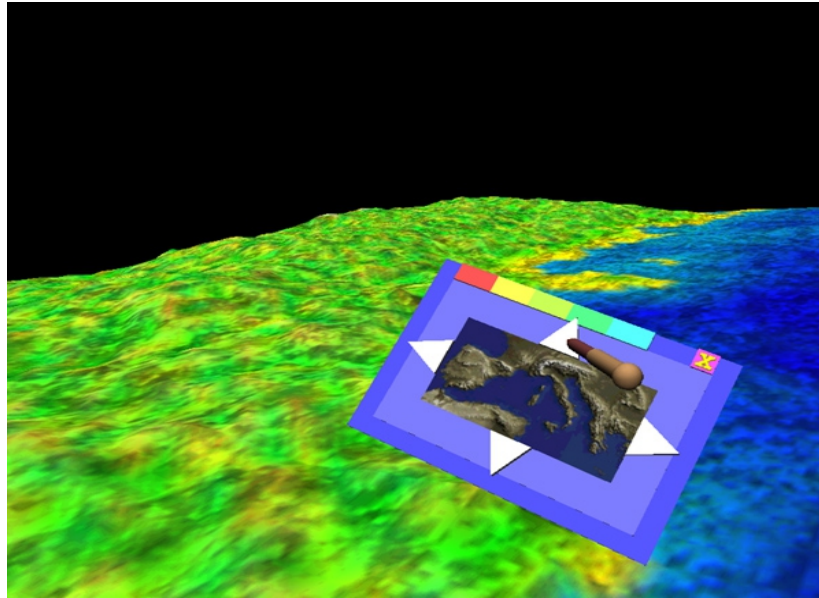


aLIVE has a unique profile in the immersion continuum as it targets a platform that cannot easily represent a VE with a high degree of fidelity/immersion. However, as Bowman and McMahan predicted, this offers the application some unique advantages: relatively inexpensive hardware, the ability to use the application in a mobile setting and the ability to incorporate real world context into the usage of the application.

Magic Lenses

aLIVE is a variation on magic lens interfaces⁷. Historically, magic lenses have been implemented as physical objects or software entities that represent an optional, secondary view that may be nested inside an application's primary viewing mode. [24–30] Of these, the prominent VR applications that implement magic lenses do so in the traditional VR style with the addition of a flat, motion-tracked artifact.

⁷ A type of UI that simulates a lens with “magical” properties. Objects seen through the lens are perceived differently by the user than objects not perceived through the lens. Ex: a 3D model-viewing magic lens that allows the user to view the inside contours of the 3D model.



Personal Interaction Panel [<http://www.cg.tuwien.ac.at/research/vr/pip/images/sheet04.jpg>]

The user sees the magic lens behavior in the virtual space that the motion-tracked artifact occupies in the VE [27–30]. aLIVE differs in that the object describing the magic lens (the mobile device's screen), the input device and the machine running the VE are all the same physical object. The magic lens' place in the application is secondary only to the user's view of their real world surroundings. Magic lens interfaces are well-studied for applications ranging from desktop CAD [24] to AR [26]. However, it has been difficult to find studies with performance data for magic lens interfaces comparing navigation and search tasks in VR magic lens applications to related technologies. Studies comparing navigation tasks in more visually-immersive environments have been performed and these suggest that visual immersion improves user completion time for search and navigation tasks [31]. However, the devices tested in these studies occupied vastly different areas on the immersion spectrum (desktop application vs. HMD). The studies I have been able to find using magic lens interfaces in VR applications discuss novel implementation techniques that allow the user access to a greater degree of object metadata in a single view or window [24], [26], [28–30]. As these studies do not seem relevant to navigation and search in VR applications, I have been unable to find a significant amount of related work in this capacity.

Design and Implementation

The application created for this project, aLIVE, is designed to explore the research question: “Are current-generation smart devices suitable platforms for VR applications?” This section will discuss the implementation of aLIVE by summarizing the tools and techniques used in the creation process, then discussing the design goals for the application and how they are addressed. Complications that arose in the development of the aLIVE will also be addressed at the end of this section for the benefit of those who might wish to replicate this project's design.

Design Summary



The development machine (left) and the mobile device (right) running aLIVE in debug mode.

Development environment:

- iMac and Macbook Pro
- Mac OS 10.8.3
 - required for Xcode 4.6
- Xcode 4.6.1 IDE
 - required for iOS 6.0 SDK

Mobile device:

- 3rd generation iPad
 - required for on-board gyroscope and accelerometer
- iOS 6.0.1

Application design:

- Based on Xcode's "OpenGL ES Game" application template
- Renders scene from VE to a single view that occupies the entire screen. Some debugging functionality is included via on-screen buttons.
- Navigation through 3D environment provided by the gyroscope and accelerometer

Design Goals

As this is an exploratory project, there are three design goals:

1. Render a VE on a current-generation mobile device at interactive speeds with useful

presentation

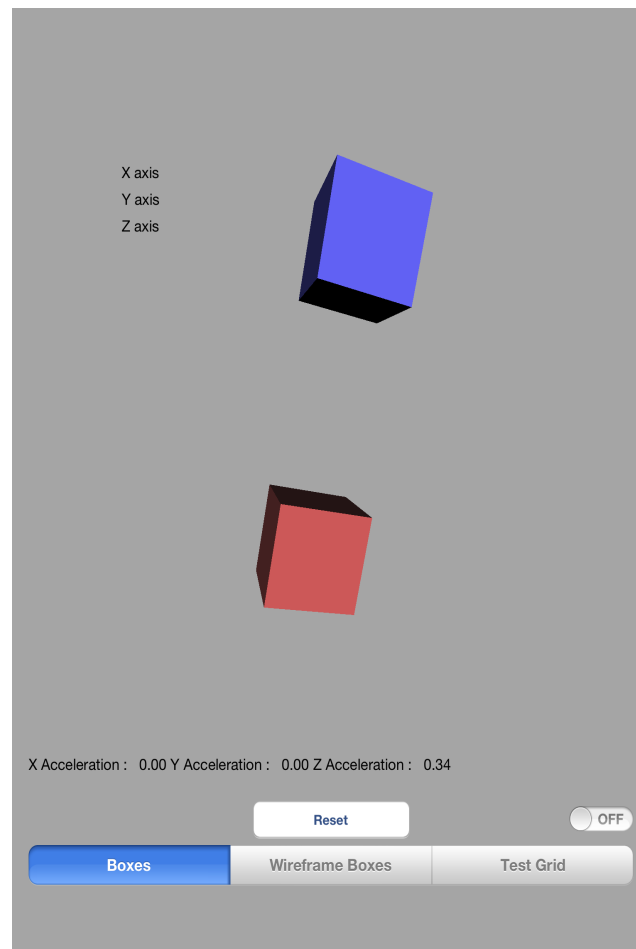
2. Allow the user to manipulate their vantage point of the scene by manipulating the mobile device
3. Establish the performance capabilities of this kind of interface in a basic implementation

1) Render a VE on a current-generation mobile device at interactive speeds with useful presentation

This first design goal must be met in order to establish that current-generation mobile devices are suitable for 3D applications of any kind, VR or otherwise. The recent advances in the mobile computing industry have demonstrated that there are many affordable devices that exceed the hardware requirements for this goal to be met. [4], [6], [7] As more and more different devices become available that meet these requirements, it becomes difficult to design a proof-of-concept project that is useful to the myriad developers working on the devices that may be able to use this kind of interface. To make sure that this project could meet this goal such that the results are applicable to as many platforms as possible, aLIVE could have been implemented either as a device-agnostic application or it could be built natively for a single platform that would represent a reasonable standard of performance across the market. Since this project is intended to rely heavily on built-in motion sensors, it was decided that the application should be built natively so as to avoid performance issues that might arise in trying to use this hardware in a device-agnostic manner and leverage native API support for motion sensing. As there is currently no agreed-upon standard platform in the mobile device market, the authors felt that the iPad 3 would be the most appropriate platform for this project to target because it is known that the iPad 3 has the graphics and motion sensing features aLIVE requires [5], [32] and because of its current position in the mobile device market⁸. Even if other mobile device developers are not very familiar with Apple devices, they need to at least have an idea of how their devices compare to Apple devices in order to be competitive in the current market.

Since it has already been shown that the iPad can render much more complex environments at interactive speeds[5], a simple VE and UI is sufficient for the purposes of this project. The VE used in the tests and demonstrations of aLIVE is composed of simple cubes as defined in the OpenGL ES Game template project available in Xcode. (Appendix, Geometry section). The aLIVE UI is as straightforward as possible: a single window or view that uses the entire screen space in order to maintain the affordances of a magic lens interface:

⁸ While there is substantial disagreement on current market trends, iPads are currently the dominating tablet computers with ~39% of worldwide tablet sales in the first quarter of 2013. [22], [23]



Screenshot taken from test device while running aLIVE

The decision to include debugging settings on the screen allows for faster configuration during testing, however they can be turned off to reduce screen clutter.

2) Allow the user to manipulate the vantage point displayed on screen by manipulating the mobile device

This goal represents aLIVE's strongest mode of immersion and the primary area of testing in this project. In order to meet this goal, the authors selected the 3rd generation iPad over previous versions of iPad as it was the most recently available Apple tablet with the hardware to support motion sensing⁹. Additionally, Xcode, the recommended Apple IDE for app development, comes with a variety of frameworks that provide tools for quickly developing apps that efficiently use the target device's resources. The frameworks this project uses to achieve this goal are the Core Motion framework and the GLKit framework. The Core Motion framework allows access to the tablet's motion sensor data while the GLKit framework provides a collection of tools for rendering 3D scenes using OpenGL ES. These two frameworks together allow the application to gather motion data to guess the device's position in space and render the VE to reflect the device's position as though it were in the VE.

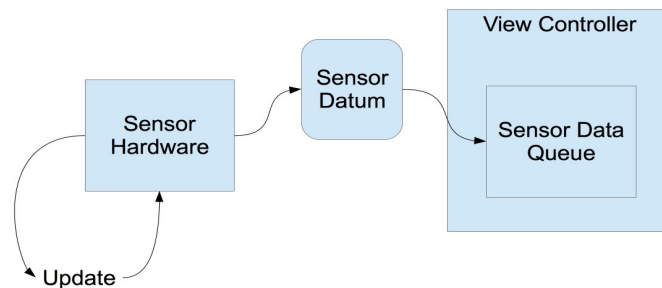
The Core Motion framework defines three objects for using motion data: an accelerometer data object, a gyroscope data object, and a device motion object. The aLIVE application uses the device

⁹ This device ships with 3-axis MEMS accelerometer and gyroscope. Previous generations of iPad shipped with only the accelerometer which was insufficient to detect rotational translation of the device.

motion object because it provides the sensor fusion logic needed to intelligently use the iPad's accelerometer and gyroscope together to guess the device's position in space. The Core Motion framework also allows for a variety of different configurations for gathering and using data via the device motion object including sampling speed and device calibration attitude¹⁰. This project requires that aLIVE can use different combinations of these configurations:

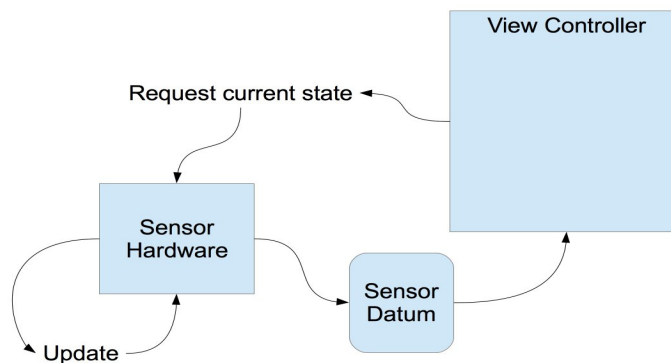
- Adjustable sensor sampling rate (Appendix, Sensor Sampling Configurations section)
- “Push” or “Pull” sensor interaction (Appendix, Sensor Sampling Configurations section)

“Push” and “Pull” are Apple Inc's names for their Core Motion framework implementation of sensor interruption and sensor polling respectively. Push uses an Objective-C block handler to collect sensor data at the specified sensor sampling rate by pushing block handler instances onto a queue which can be popped off as the system uses them.



Push: the sensor hardware updates its readings at the specified sampling rate. Every update, the new sensor datum is pushed onto a queue in the View Controller. The View Controller may then dequeue this data as it is used.

Pull reads a sensor datum whenever a higher-level application functions asks for one while the sensors' state is updated at the given sensor sampling rate.



Pull: the sensor hardware updates its readings at the specified sampling rate. The View Controller requests the current datum as it is needed to redraw the scene.

¹⁰ Attitude in this context refers to orientation in 3D space as defined by Euler angles or quaternions.

Preliminary testing of these two different methods in aLIVE's development process indicated that there was no difference between the two except that push method is more memory intensive as there is a queue of sensor data that must be maintained. For testing purposes, this paper only evaluated performance for the pull method.

Using fused data to derive a device's translation through space is still difficult in this context due to limitations in current sensor technology. A sensor's ability to accurately measure a phenomenon is a ratio of the device's sampling rate to the phenomenon's rate of change. Therefore, to capture sudden, rapidly changing mobile device movement, a high sampling rate is required. However, as the sampling rate increases, the amount of erroneous noise data increases and the system resources available for other computationally-heavy tasks, such as rendering a 3D scene, decrease. Further, deriving a position datum from a feed of accelerometer data is problematic because position is a double integral of acceleration – small errors in acceleration data can lead to enormous errors in position. [33] With this in mind, aLIVE tests a simple clamping technique and a simple filtering technique (Appendix, Clamping and Filtering section) for reducing sensor noise.

3) Establish the performance capabilities of this kind of interface in its most basic form

As more research is performed in the areas of VR, user motion sensing and mobile devices, it will be important to understand how that research impacts the capabilities of the techniques we have already developed. For example, the intended impact on aLIVE in the VR community is to demonstrate that it is possible to create applications that use immersion and presence in a mobile environment. This third design goal is intended to help create a sort of benchmark for future work in this area; any technologies that wish to expand or improve aLIVE's capabilities will need to meet or beat its current abilities. This goal is addressed by focusing on VE navigation using dead-reckoning¹¹ as the author felt it would be the simplest technique for creating kinesthetic immersion on a mobile device. By focusing aLIVE's development on this technique, the testing process is simplified and the possibility of future work is much more open-ended.

While the iPad 3 may have been the best platform to target from the perspective of this study it is important to note that this decision had a substantial impact on the implementation process. In particular, Apple's developer program hindered development speed and certain Xcode features improved the implementation experience. However, the benefits of these features was somewhat mitigated by Apple's documentation practices. Participation in Apple's developer program is required in order to be able to run custom code on Apple devices. While the details of the registration process are beyond the scope of this paper, anyone wishing to replicate this work should know that it will be necessary to research hardware and SDK requirements in advance as these will have dependencies that must be supported by the Apple-registered development environment. Additionally, the documentation of the tools available in the Core Motion framework did not facilitate detailed understanding of the limitations and requirements of the device's sensors. The effects this had on the project as a whole are evaluated in the Discussion section of this paper.

Discussion

In order for aLIVE to show conclusively that the current generation of mobile computers is

¹¹ Dead-reckoning is a navigation technique that does not use calibration or correction information from outside of the device. This technique calculates the device's current location based upon a starting point or “fix” and a log of motion data gathered after the fix was established. The term comes from maritime navigation, when a navigator has only the tools onboard the ship to determine the vessel's whereabouts.

suitable for VE navigation applications, it must first be shown that the application can correctly map its position in real space to its position in the VE within a certain acceptable amount of error over time. Errors in this kind of mapping due to calibration problems or sensor errors will be experienced by the user as drift¹². If the application drifts to the extent that the user has very little control over their position in the VE, it will indicate that motion-based dead-reckoning on the iPad 3 is not sufficient for a VE navigation application, similar implementations on other platforms will likely have the same issues with drift and other resources must be brought to bear in order to make this kind of application possible. If the application drifts to the extent that the user has a moderate amount of control over their position in the VE, then we can say that the application might be suitable for navigation applications that only require a coarse degree of control. If the application does not appear to drift at all, then previously impossible applications such as indoor GPS will become trivial as the user's location indoors will be the sum of their last GPS coordinate and the vector sum of their mobile device's movements since entering the building. This last case is so unlikely it is absurd, however, if it turns out that the application can maintain a within-tolerances coupling between its real space location and its VE location for long enough, applications such as indoor GPS might not be as unlikely a possibility as originally thought.

To assess the application's drift and to better understand how much drift is tolerable, aLIVE's motion sensing capabilities are assessed in in two different tests:

1. To assess application's drift in translational motion, the device is carried in a vertical position, screen facing the user, and walked steadily forward in a straight line. Every 10 feet (3.05 meters) the user stops walking and the device's calculated position in user space is read. The configuration for this test was a sampling rate of 50 Hz and an accelerometer clamping threshold of .3 Gs (or 2.941995 m/s²). (Appendix Test 1 Data section)
2. To assess application's drift in rotational motion, the device is placed upon a flat, level surface and rotated about each of its axes a specified amount. The configuration for this test was a sampling rate of 50 Hz and an accelerometer clamping threshold of .3 Gs. (Appendix Test 2 Data section)

If test 1 were performed on an ideal system, the ratio of distance traveled to distance reported by the application would be 1:1 for every measurement. The data from this test run shows that it approximates a 1:1 ratio, but with some odd behavior, particularly at the 15-foot mark. Prior to this point, the distance the application reported was close to the actual the device had traveled. However, once the device had traveled 15 feet, there was a large discrepancy between the reported and actual position. After this point, the reported position becomes increasingly accurate. While these results are somewhat promising, the drift was quite noticeable and without further knowledge on how the MEMS sensors work and how the Core Motion framework uses them, it is impossible to say whether the odd behavior at the 15-foot mark was a fluke or the apparent stability up to that point was a fluke.

As indicated by test 2, rotation drift is almost a non-issue. Deriving a device's attitude through sensor fusion may be done more reliably than deriving position as there is more data available in the derivation process; the current attitude is derived from the reference orientation, the x-axis, y-axis, and z-axis rotation values, and a gravity vector which is calculated from the accelerometer while current position can only be derived by integrating the changes in acceleration over time.

The linchpin of this proof-of-concept is the extent to which we are able to use the iPad's sensors to obtain accurate readings of device position as mapping from real space is the application's only

¹² As the difference between what the user expects to experience and what they actually experience increases, the 3D scene will appear to float or “drift” free of the user's input.

immersion technique. Since position is a double integral of acceleration, noise will be exacerbated and, if any filtering/clamping is present, low or constant acceleration will not be detected. Further, depending on how acceleration filtering or clamping has been implemented, the iPad's physical properties when speeding up or slowing down will feel odd, hampering the user's presence in the application. If we were able to obtain exact position data more directly as other, non-mobile applications do, there would be little to no drift at all. However, this is nearly impossible to do in a mobile setting without resorting to computationally-expensive techniques, such as terrain feature recognition via camera, or non-standard sensor hardware, such as LIDAR, which could also be computationally-expensive to use.

Though was straightforward to accomplish project design goal number 2 as it was written, “allow the user to manipulate the vantage point displayed on screen by manipulating the mobile device”, the drift behavior encountered in test 1 indicates that an application like aLIVE is not suited for handling tasks that require precise translational motion. This finding is disappointing because it mitigates one of the key advantages to the aLIVE implementation: the application platform is mobile but it cannot accurately replicate, in a VE, the kind of long-distance motion mobile devices see in everyday use. Based upon the findings from test 3, it would take only a couple minutes of active use for the drift to become so severe that it would not be possible to see the user's expected position within the VE from the device's actual position in the VE. However, there are still a wide range of application where this kind of performance would suffice. For example: if a museum had a persistent VE that patrons could experience using aLIVE, so long as the application could be re-calibrated at every exhibit, ~10(?) feet via QR code or other marker, the experience could still feel cohesive.

Future Work

The possibilities for further investigation on aLIVE and the design space it resides in are too many to be enumerated here but there are phases that future work in this vein will need to pass through in order to become acceptable for everyday use. The first phase of future work will include work that answers the original research question of this paper: “Are current-generation mobile devices suitable for VE navigation applications?” The second phase will explore additional methods and consequences of human interaction with this interface.

The first phase of future work will have two sub-parts. The first part will focus on tools and techniques for addressing drift and the second part will focus on human subjects testing. Before human subjects testing will yield useful data, it will be important to make aLIVE's most important immersion technique as dependable as possible and this cannot happen without reducing the adverse effects of drift. Techniques that could be used toward this end (and have seen success in other applications) include sophisticated accelerometer filtering techniques, such as Kalman filtering, or relying on other onboard tools, such as the GPS and compass, or placing markers in the environment for updating the device's fix or combinations of the above. Another approach to reducing the negative effects of drift would be to expand the application's magic lens metaphor to include drift-tolerant behavior. This could be done by simulating friction, spring action, or other behaviors in the virtual representation of the tablet. Once techniques have been found that can correct drift to within a certain tolerance, the second part of this first phase will use human subjects testing to answer questions like: Does the small screen size prevent users from feeling present in the VE? Is there an optimal scale size for VE applications in this context? How does this kind of interface compare to older immersive interfaces for search and navigation tasks? Are users able to split their attention between an abstract VE's terrain features and real world terrain features to safely avoid collisions? Once these questions have been answered, we can determine whether or not the current-generation of mobile devices is suitable for VE navigation

applications.

The second phase of future work will examine the effectiveness of other tools and technologies integrated into a more fully-fledged version of aLIVE. For example, this phase could explore functionality based on other mobile platform features such as multitouch screen actions or the inclusion of other users and their mobile devices in the VE. Of particular interest in this area is 3D object selection. 3D object selection techniques have been well-studied on stationary computing platforms but not for mobile devices and will be an important feature for future VE navigation applications.

Conclusion

This paper has begun to answer the question, “Are current-generation mobile devices suitable for VE navigation applications?” by examining the evolution of VR taxonomy and implementation, taking a small step in exploring the dimensionality of Bowman and McMahan's immersion continuum, building an experimental interface and testing its performance. The current trend in the VR community is pushing the boundaries of what immersion means by designing alternative means of engaging non-traditional human sense modalities to create the experience of being in a VE. The project featured in this paper, aLIVE, combines the idea of engaging non-traditional sense modalities with mobile devices to create a user interface that can explore a VE without being tied to a stationary location or preventing the user from interacting with their surroundings. The application was built and tested on the iPad 3 and attempts to use dead-reckoning navigation to turn basic kinesthetic motion into an intuitive means of navigating a VE. Though the application is found to be excellent at capturing and modeling rotational motion, it is unable to accurately model translational motion and suffers from drift. As translational motion is a key part of the interface's design and mobile device usage in general, the application indicates that this design is suitable for applications that only require the user to provide very coarse translation data or none at all. This finding is somewhat disappointing as it represents a major barrier preventing this kind of application from being used effectively in a mobile setting.

Although this project has found a major barrier to the usefulness of this style of interface, future work in addressing drift is promising. There are numerous other fields from which navigation techniques can be leveraged to this end as well as changes that could be made to the device's control loop to tighten translational motion in the application. The results of this project and the successes of other, similar technologies indicate that while a robust, satisfying, mobile VE navigation application is not possible on this version of aLIVE, it is likely that it will be possible on a device that is very similar to it.

Appendix

Geometry

```

GLfloat gCubeVertexData[216] =
{
    // Data layout for each line below is:
    // positionX, positionY, positionZ,   normalX, normalY, normalZ,
    0.5f, -0.5f, -0.5f,    1.0f, 0.0f, 0.0f,
    0.5f, 0.5f, -0.5f,    1.0f, 0.0f, 0.0f,
    0.5f, -0.5f, 0.5f,    1.0f, 0.0f, 0.0f,
    0.5f, -0.5f, 0.5f,    1.0f, 0.0f, 0.0f,
    0.5f, 0.5f, 0.5f,    1.0f, 0.0f, 0.0f,
    0.5f, 0.5f, -0.5f,    1.0f, 0.0f, 0.0f,

    0.5f, 0.5f, -0.5f,    0.0f, 1.0f, 0.0f,
    -0.5f, 0.5f, -0.5f,    0.0f, 1.0f, 0.0f,
    0.5f, 0.5f, 0.5f,    0.0f, 1.0f, 0.0f,
    0.5f, 0.5f, 0.5f,    0.0f, 1.0f, 0.0f,
    -0.5f, 0.5f, -0.5f,    0.0f, 1.0f, 0.0f,
    -0.5f, 0.5f, 0.5f,    0.0f, 1.0f, 0.0f,

    -0.5f, 0.5f, -0.5f,    -1.0f, 0.0f, 0.0f,
    -0.5f, -0.5f, -0.5f,    -1.0f, 0.0f, 0.0f,
    -0.5f, 0.5f, 0.5f,    -1.0f, 0.0f, 0.0f,
    -0.5f, 0.5f, 0.5f,    -1.0f, 0.0f, 0.0f,
    -0.5f, -0.5f, -0.5f,    -1.0f, 0.0f, 0.0f,
    -0.5f, -0.5f, 0.5f,    -1.0f, 0.0f, 0.0f,

    -0.5f, -0.5f, -0.5f,    0.0f, -1.0f, 0.0f,
    0.5f, -0.5f, -0.5f,    0.0f, -1.0f, 0.0f,
    -0.5f, -0.5f, 0.5f,    0.0f, -1.0f, 0.0f,
    -0.5f, -0.5f, 0.5f,    0.0f, -1.0f, 0.0f,
    0.5f, -0.5f, -0.5f,    0.0f, -1.0f, 0.0f,
    0.5f, -0.5f, 0.5f,    0.0f, -1.0f, 0.0f,

    0.5f, 0.5f, 0.5f,    0.0f, 0.0f, 1.0f,
    -0.5f, 0.5f, 0.5f,    0.0f, 0.0f, 1.0f,
    0.5f, -0.5f, 0.5f,    0.0f, 0.0f, 1.0f,
    0.5f, -0.5f, 0.5f,    0.0f, 0.0f, 1.0f,
    -0.5f, 0.5f, 0.5f,    0.0f, 0.0f, 1.0f,
    -0.5f, -0.5f, 0.5f,    0.0f, 0.0f, 1.0f,

```

Carr 22

```
    0.5f, -0.5f, -0.5f,    0.0f, 0.0f, -1.0f,
    -0.5f, -0.5f, -0.5f,    0.0f, 0.0f, -1.0f,
    0.5f, 0.5f, -0.5f,    0.0f, 0.0f, -1.0f,
    0.5f, 0.5f, -0.5f,    0.0f, 0.0f, -1.0f,
    -0.5f, -0.5f, -0.5f,    0.0f, 0.0f, -1.0f,
    -0.5f, 0.5f, -0.5f,    0.0f, 0.0f, -1.0f
};

- (void)setupGL
{
    [EAGLContext setCurrentContext:self.context];

    [self loadShaders];

    self.effect = [[GLKBaseEffect alloc] init];
    self.effect.light0.enabled = GL_TRUE;
    self.effect.light0.diffuseColor = GLKVector4Make(1.0f, 0.4f, 0.4f, 1.0f);

    glEnable(GL_DEPTH_TEST);

    glGenVertexArraysOES(1, &_vertexArray);
    glBindVertexArrayOES(_vertexArray);

    glGenBuffers(1, &_vertexBuffer);
    glBindBuffer(GL_ARRAY_BUFFER, _vertexBuffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(gCubeVertexData), gCubeVertexData,
GL_STATIC_DRAW);

    glEnableVertexAttribArray(GLKVertexAttribPosition);
    glVertexAttribPointer(GLKVertexAttribPosition, 3, GL_FLOAT, GL_FALSE, 24,
BUFFER_OFFSET(0));
    glEnableVertexAttribArray(GLKVertexAttribNormal);
    glVertexAttribPointer(GLKVertexAttribNormal, 3, GL_FLOAT, GL_FALSE, 24,
BUFFER_OFFSET(12));

    // Creating and binding marker VAO/VBO

    glGenVertexArraysOES(1, &_markerVertexArray);
    glBindVertexArrayOES(_markerVertexArray);

    glGenBuffers(1, &_markerVertexBuffer);
    glBindBuffer(GL_ARRAY_BUFFER, _markerVertexBuffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(gMarkerVertexData), gMarkerVertexData,
GL_STATIC_DRAW);
```

```

    glEnableVertexAttribArray(GLKVertexAttribPosition);
    glVertexAttribPointer(GLKVertexAttribPosition, 3, GL_FLOAT, GL_FALSE, 0,
    BUFFER_OFFSET(0));

    glBindVertexArrayOES(0);
}

- (void)glkView:(GLKView *)view drawInRect:(CGRect)rect
{
    glClearColor(0.65f, 0.65f, 0.65f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    switch (modeSwitch.selectedSegmentIndex) {
        case WIREFRAME_TEST_ENV:

            glBindVertexArrayOES(_markerVertexArray);
            [self.effect prepareToDraw];
            glDrawArrays(GL_LINES, 0, 126);

            break;

        case WIREFRAME_BOXES:

            glBindVertexArrayOES(_vertexArray);

            // Render the object with GLKit
            [self.effect prepareToDraw];

            glDrawArrays(GL_LINE_LOOP, 0, 36);

            // Render the object again with ES2
            glUseProgram(_program);

            glUniformMatrix4fv(uniforms[UNIFORM_MODELVIEWPROJECTION_MATRIX], 1, 0,
            _modelViewProjectionMatrix.m);
            glUniformMatrix3fv(uniforms[UNIFORM_NORMAL_MATRIX], 1, 0, _normalMatrix.m);

            glDrawArrays(GL_LINE_LOOP, 0, 36);

            break;

        default:

```

Carr 24

```
glBindVertexArrayOES(_vertexArray);

// Render the object with GLKit
[self.effect prepareToDraw];

glDrawArrays(GL_TRIANGLES, 0, 36);

// Render the object again with ES2
glUseProgram(_program);

glUniformMatrix4fv(uniforms[UNIFORM_MODELVIEWPROJECTION_MATRIX], 1, 0,
_modelViewProjectionMatrix.m);
glUniformMatrix3fv(uniforms[UNIFORM_NORMAL_MATRIX], 1, 0, _normalMatrix.m);

glDrawArrays(GL_TRIANGLES, 0, 36);

break;
}
}
```

Sensor Sampling Configurations

```
// ----- DEBUG ENV OPTIONS ----- //
```

#define DeviceMotionWithQueue	0
#define DeviceMotionNoQueue	!DeviceMotionWithQueue

#define ApplyFilters	1
//#define FilterThreshold	.3 // Moved to UI

#define DebugEulerAngles	0
#define DebugQuat	0
#define DebugUserAccel	1


```
- (void) enableDeviceMotionSensors
{
    if ( ![sensorManager isDeviceMotionActive] ) {

        [sensorManager setDeviceMotionUpdateInterval: 1.0f/sensorFrequencyStepper.value ];
        [sensorFrequencyLabel setText:[ NSString stringWithFormat:@"%f",
sensorFrequencyStepper.value]];
    }
}
```



```
#if DeviceMotionWithQueue
```

```
// CMAttitudeReferenceFrame is a predefined enum:
```

```
// CMAttitudeReferenceFrameXArbitraryZVertical - Z axis oriented vertically, as on a table.
```

```
// CMAttitudeReferenceFrameXArbitraryCorrectedZVertical - Uses magnetometer to correct yaw  
and results in increased CPU usage
```

```
// Apple says this uses "Push" method to retrieve sensor data
```

```
[sensorManager
```

```
startDeviceMotionUpdatesUsingReferenceFrame:CMAttitudeReferenceFrameXArbitraryCorrectedZVe  
rtical toQueue: [[NSOperationQueue alloc] init] withHandler: ^(CMDeviceMotion *dmReceived,  
NSError *error)  
{
```

```
    CMQuaternion currentAttitude_CM = dmReceived.attitude.quaternion;
```

```
    double x_posNext, y_posNext, z_posNext;
```

```
    double x_velNext, y_velNext, z_velNext;
```

```
// multiplying out Gs to get m/s^2
```

```
    double currentAccelerationX = (dmReceived.userAcceleration.x) * Gs;
```

```
    double currentAccelerationY = (dmReceived.userAcceleration.y) * Gs;
```

```
    double currentAccelerationZ = (dmReceived.userAcceleration.z) * Gs;
```

```
// ROTATION
```

```
// We negate the Z rotation in order to simulate a lens
```

```
    GLKQuaternion currentAttitude_raw = GLKQuaternionMake( -currentAttitude_CM.x,  
-currentAttitude_CM.y, -currentAttitude_CM.z, currentAttitude_CM.w);
```

```
    GLKQuaternionNormalize( currentAttitude_raw );
```

```
    cmRotate_modelViewMatrix = GLKMatrix4MakeWithQuaternion(currentAttitude_raw);
```

```
// TRANSLATION
```

```
// We assume constant acceleration over the sampling interval
```

```
//
```

```
//      time_n          time_n+1          time_n+2
```

```
//  -----|-----|-----|-----
```

```
//      x_pos          x_posNext
```

```
//      x_vel          x_velNext
```

```
#if ApplyFilters
```

```
    if ( fabs(currentAccelerationX) < filterValStepper.value) currentAccelerationX = 0.0f;
```

```
    if ( fabs(currentAccelerationY) < filterValStepper.value) currentAccelerationY = 0.0f;
```

```
    if ( fabs(currentAccelerationZ) < filterValStepper.value) currentAccelerationZ = 0.0f;
```

Carr 26

#endif

```
x_posNext = x_pos + (x_vel * (1.0/sensorFrequencyStepper.value)) + (.5 *  
currentAccelerationX * pow(1.0/sensorFrequencyStepper.value,2));  
y_posNext = y_pos + (y_vel * (1.0/sensorFrequencyStepper.value)) + (.5 *  
currentAccelerationY * pow(1.0/sensorFrequencyStepper.value,2));  
z_posNext = z_pos + (z_vel * (1.0/sensorFrequencyStepper.value)) + (.5 *  
currentAccelerationZ * pow(1.0/sensorFrequencyStepper.value,2));
```

```
x_velNext = x_vel + (currentAccelerationX * (1.0/sensorFrequencyStepper.value));  
y_velNext = y_vel + (currentAccelerationY * (1.0/sensorFrequencyStepper.value));  
z_velNext = z_vel + (currentAccelerationZ * (1.0/sensorFrequencyStepper.value));
```

// We negate the direction of the vector to simulate a lens

```
cmTranslate_modelViewMatrix = GLKMatrix4MakeTranslation( x_posNext, y_posNext,  
z_posNext );
```

```
x_pos = x_posNext;  
y_pos = y_posNext;  
z_pos = z_posNext;
```

```
x_vel = x_velNext;  
y_vel = y_velNext;  
z_vel = z_velNext;
```

// ----- LOGGING DEVICE DATA TO SCREEN ----- //

// VE Position Data

```
x.text = [NSString stringWithFormat:@"X pos: %f", x_pos];  
y.text = [NSString stringWithFormat:@"Y pos: %f", y_pos];  
z.text = [NSString stringWithFormat:@"Z pos: %f", z_pos];
```

#if !SensorStats

// Euler Angles

#if DebugEulerAngles

```
NSString * pitch = [[NSString alloc] initWithFormat: @"Pitch : %6.2f ",  
dmReceived.attitude.pitch ];  
NSString * roll = [[NSString alloc] initWithFormat: @"Roll : %6.2f ", dmReceived.attitude.roll ];  
NSString * yaw = [[NSString alloc] initWithFormat: @"Yaw : %6.2f ",  
dmReceived.attitude.yaw ];
```

```
NSString * combineAll = [pitch stringByAppendingString: [roll stringByAppendingString: yaw] ];
```

Carr 27

```
[self logToScreenAndConsole:combineAll];
#endif

// Quaternion
#if DebugQuat
    NSString * quaternionScalar = [[NSString alloc] initWithFormat: @"Scalar : %6.2f ",
dmReceived.attitude.quaternion.w ];
    NSString * quaternionVX = [[NSString alloc] initWithFormat: @"VX : %6.2f ",
dmReceived.attitude.quaternion.x ];
    NSString * quaternionVY = [[NSString alloc] initWithFormat: @"VY : %6.2f ",
dmReceived.attitude.quaternion.y ];
    NSString * quaternionVZ = [[NSString alloc] initWithFormat: @"VZ : %6.2f ",
dmReceived.attitude.quaternion.z ];

    NSString * combineQuat = [quaternionScalar stringByAppendingString: [quaternionVX
stringByAppendingString:[ quaternionVY stringByAppendingString:quaternionVZ]]];
    [self logToScreenAndConsole:combineQuat];
#endif

// User Acceleration
#if DebugUserAccel
    NSString * xAccel = [[NSString alloc] initWithFormat: @"X Acceleration : %6.2f ",
dmReceived.userAcceleration.x ];
    NSString * yAccel = [[NSString alloc] initWithFormat: @"Y Acceleration : %6.2f ",
dmReceived.userAcceleration.y ];
    NSString * zAccel = [[NSString alloc] initWithFormat: @"Z Acceleration : %6.2f ",
dmReceived.userAcceleration.z ];

    NSString * combineUserAccel = [xAccel stringByAppendingString: [yAccel
stringByAppendingString: zAccel] ];
    [self logToScreenAndConsole:combineUserAccel];
#endif

#endif

// Sensor Statistics
#if SensorStats
    [xAccelStats dupdate:dmReceived.userAcceleration.x];
    [yAccelStats dupdate:dmReceived.userAcceleration.y];
    [zAccelStats dupdate:dmReceived.userAcceleration.z];

    [xVelStats fupdate:x_velNext];
    [yVelStats fupdate:y_velNext];
    [zVelStats fupdate:z_velNext];
```

Carr 28

```
#endif

    };

#else // DeviceMotionNoQueue
    // This usage of startDeviceMotion is recommended for videogames which are only interested in
    // the current position data of the device. Since we're storing the previous device motion data in the
    // ViewController we don't really need to have the queue. However, this option does not seem to take
    // into account the update interval.
    [sensorManager startDeviceMotionUpdates];
#endif
}

}
```

Clamping and Filtering

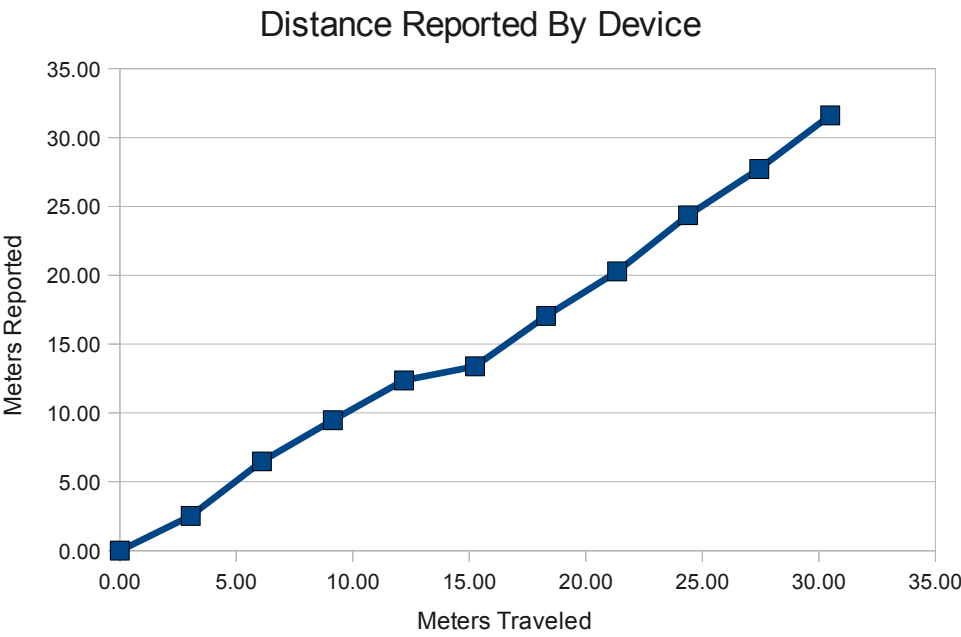
```
#define ApplyFilters            1
// #define FilterThreshold      .3 // Moved to UI
#define Clamping               1
#define Filtering              !Clamping

#if ApplyFilters
#if Clamping
    if ( fabs(currentAccelerationX) < filterValStepper.value) currentAccelerationX = 0.0;
    if ( fabs(currentAccelerationY) < filterValStepper.value) currentAccelerationY = 0.0;
    if ( fabs(currentAccelerationZ) < filterValStepper.value) currentAccelerationZ = 0.0;
#endif
#endif
#if Filtering
    currentAccelerationX = [self HiPassFilter2:currentAccelerationX prev:prevAccelX];
    currentAccelerationY = [self HiPassFilter2:currentAccelerationY prev:prevAccelY];
    currentAccelerationZ = [self HiPassFilter2:currentAccelerationZ prev:prevAccelZ];
    prevAccelX = currentAccelerationX;
    prevAccelY = currentAccelerationY;
    prevAccelZ = currentAccelerationZ;
#endif
#endif

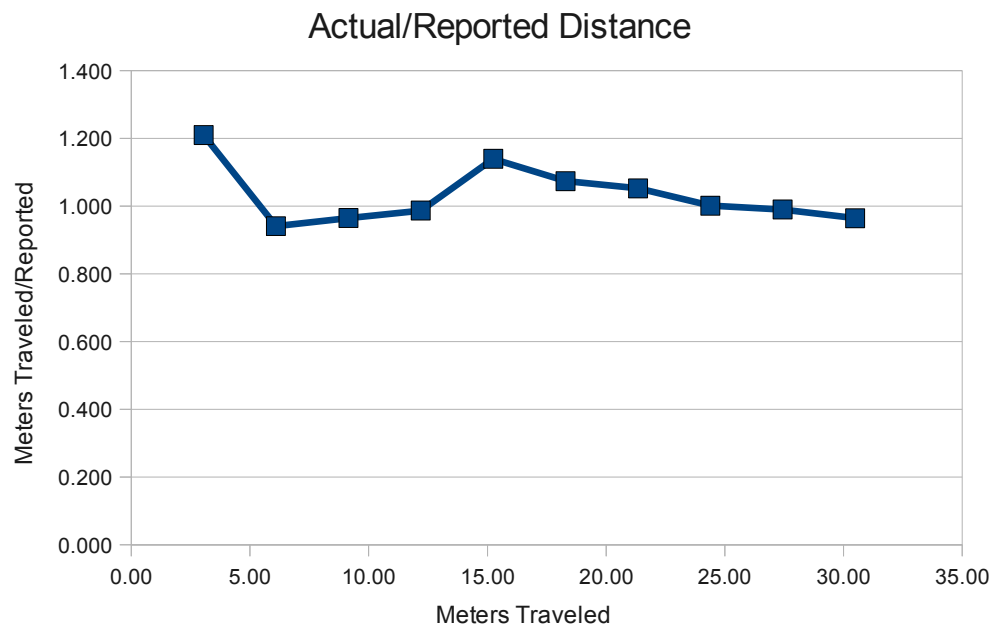
- (double) HiPassFilter2:(double)currentVal prev:(double)previousVal {
    return currentVal - ( (currentVal * filterValStepper.value) + (previousVal * (1.0 -
filterValStepper.value)) );
}
```

Test 1 Data

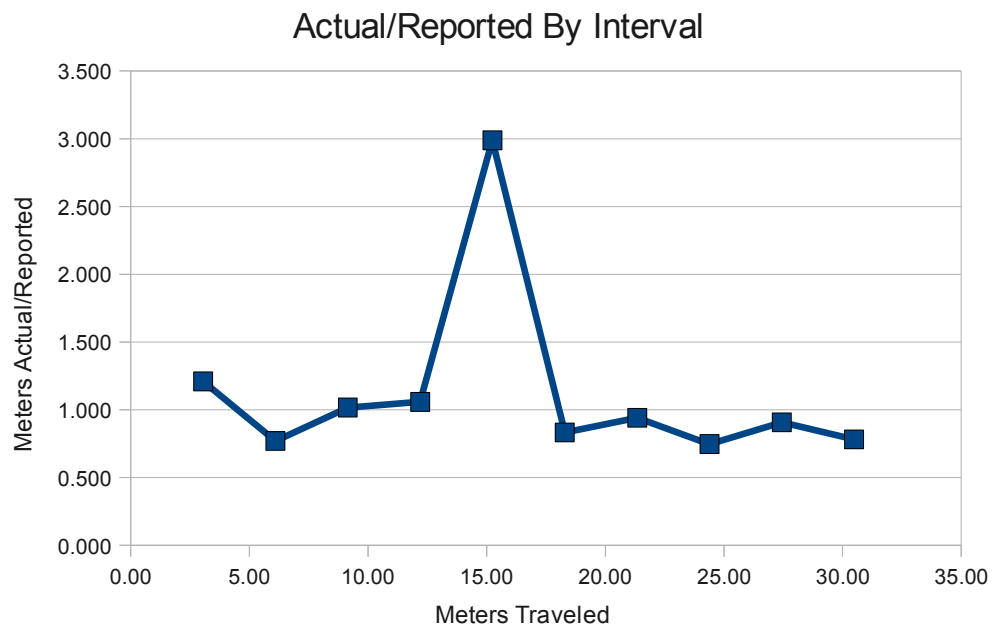
Meters (actual)	Meters (reported)
0.00	0.00
3.05	2.52
6.10	6.48
9.15	9.48
12.20	12.36
15.24	13.38
18.29	17.04
21.34	20.28
24.39	24.36
27.44	27.72
30.49	31.62



Meters (actual)	actual/reported
3.05	1.210
6.10	0.941
9.15	0.965
12.20	0.987
15.24	1.139
18.29	1.074
21.34	1.052
24.39	1.001
27.44	0.990
30.49	0.964



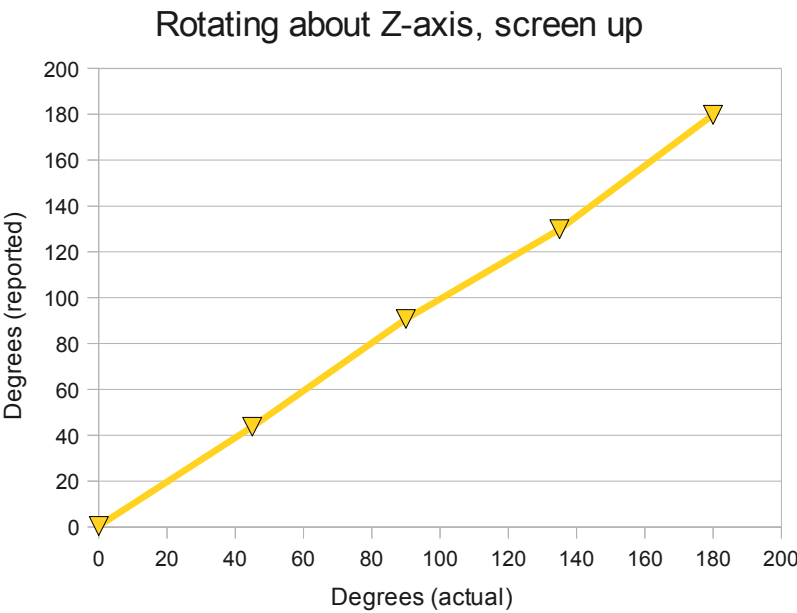
Meters (actual)	actual/reported by interval
3.05	1.210
6.10	0.770
9.15	1.016
12.20	1.059
15.24	2.989
18.29	0.833
21.34	0.941
24.39	0.747
27.44	0.907
30.49	0.782



Test 2 Data

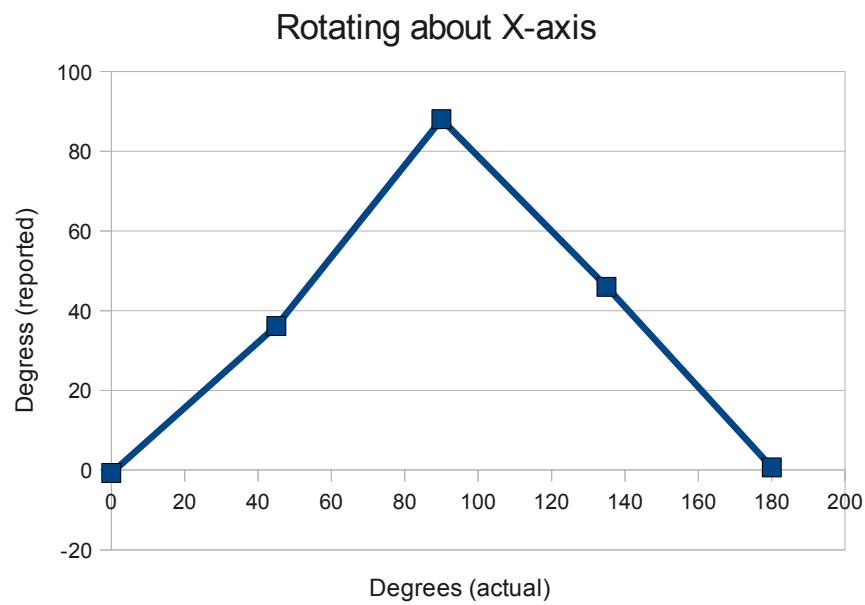
Rotation about Z-axis:

Degrees	yaw
0	0.45
45	43.75
90	90.75
135	129.86
180	179.78



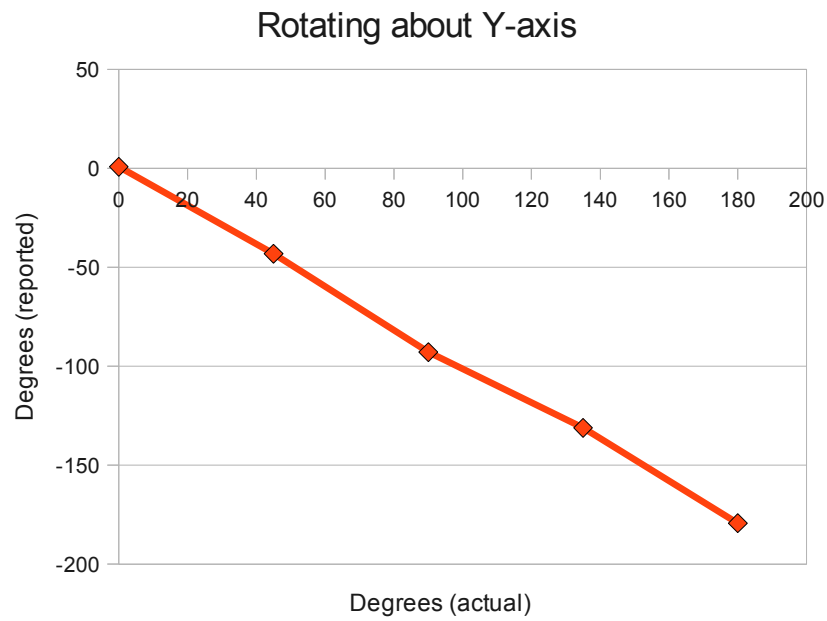
Rotation about X-axis:

Degrees	pitch
0	-0.75
45	36.15
90	88.03
135	45.97
180	0.68



Rotation about Y-axis:

Degrees	roll
0	0.71
45	-43.23
90	-92.9
135	-131.2
180	-179.32



Bibliography

- [1] P. Milgram, "Augmented reality: A class of displays on the reality-virtuality continuum," *Photonics for ...*, 1995.
- [2] J. Fox, D. Arena, and J. N. Bailenson, "Virtual Reality," *Journal of Media Psychology: Theories, Methods, and Applications*, vol. 21, no. 3, pp. 95–113, Jan. 2009.
- [3] D. A. Bowman and R. P. McMahan, "Virtual Reality: How Much Immersion Is Enough?," vol. 40, 2007.
- [4] Oode, "Epic Citadel Benchmark - Nexus 10 vs Nexus 7 vs Nexus 4 - HD." [Online]. Available: <https://www.youtube.com/watch?v=6-M4WdEQVns>.
- [5] K. Holland, "Linpack Benchmark: iPad vs iPod Touch 4G (both iOS 4+)." [Online]. Available: https://www.youtube.com/watch?v=J3A3Yibd_5w.
- [6] Arcatablet, "Epic Citadel benchmark ARCHOS 97 Titanium HD VS Onda V972." [Online]. Available: <https://www.youtube.com/watch?v=IVFopYzYTAw>.
- [7] PhoneArena, "Motorola DROID BIONIC benchmark tests." [Online]. Available: <https://www.youtube.com/watch?v=HdHCRDLu548>.
- [8] AjudandroidOficial, "LG Optimus 4X HD P880 Benchmark Epic Citadel." [Online]. Available: <https://www.youtube.com/watch?v=qyTwDRz7Wbk>.
- [9] K. Hinkley, R. Pausch, J. Goble, and N. Kassell, "A Survey of Design Issues in Spatial Input." [Online]. Available: <http://research.microsoft.com/en-us/um/people/kenh/papers/survey.htm>. [Accessed: 07-Oct-2012].
- [10] R. Kopper, D. A. Bowman, M. G. Silva, and R. P. McMahan, "A human motor behavior model for distal pointing tasks," *International Journal of Human-Computer Studies*, vol. 68, no. 10, pp. 603–615, 2010.
- [11] N. T. Dang, M. Tavanti, I. Rankin, and M. Cooper, "A comparison of different input devices for a 3D environment," *International Journal of Industrial Ergonomics*, vol. 39, no. 3, pp. 554–563, May 2009.
- [12] SimbolRides, "Virtual Reality Full-Motion Helicopter Flight Simulator on FSX." [Online]. Available: <https://www.youtube.com/watch?v=n534e-OL1bI>. [Accessed: 05-Oct-2013].
- [13] "Oculus Rift." [Online]. Available: <http://www.oculusvr.com/>. [Accessed: 05-Oct-2013].
- [14] R. Pausch, J. Snoddy, R. Taylor, S. Watson, and E. Haseltine, "Disney's Aladdin," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, 1996, pp. 193–203.
- [15] S. Nichols and H. Patel, "Health and safety implications of virtual reality: A review of empirical evidence.," *Applied Ergonomics*, 2002.
- [16] M. Slater, "A Note on Presence Terminology," *Presence-Connect*, pp. 1–5.
- [17] I. E. Sutherland, "A head-mounted three dimensional display," *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*, p. 757, 1968.
- [18] D. Manek, "Effects of visual displays on 3d interaction in virtual environments," 2004.

- [19] M. T. Bolas, E. R. Lorimer, I. E. McDowall, and R. X. Mead, "Proliferation of counterbalanced, CRT-based stereoscopic displays for virtual environment viewing and control," in *IS&T/SPIE 1994 International Symposium on Electronic Imaging: Science and Technology*, 1994, pp. 325–334.
- [20] H. Fouad, J. Ballas, and D. Brock, "An extensible toolkit for creating virtual sonic environments," *Proceedings of Intl. Conf. on Auditory Display*, 2000.
- [21] C. Colwell, H. Petrie, and D. Kornbrot, "Haptic virtual reality for blind computer users," *Proceedings of the third ...*, 1998.
- [22] IDC, "Worldwide Tablet Market Surges Ahead on Strong First Quarter Sales, Says IDC." .
- [23] K. C. Tofel, "Android vs. iPad: the tablet sales figures that matter," *GIGAOM*. [Online]. Available: <http://gigaom.com/2011/07/22/android-vs-ipad-the-tablet-sales-figures-that-matter/>. [Accessed: 05-Apr-2013].
- [24] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose, "Toolglass and magic lenses," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH '93*, 1993, pp. 73–80.
- [25] P. S. W. B. G. Chi, W. Fu, Q. Meng, and P. A. Heng, "WYSIWYF: Exploring and Annotating Volume Data with a Tangible Handheld Device," *mysites.ntu.edu.sg*.
- [26] J. Looser, R. Grasset, and M. Billinghurst, "A 3D Flexible and Tangible Magic Lens in Augmented Reality," *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–4, 2007.
- [27] J. S. Pierce, A. S. Forsberg, M. J. Conway, S. Hong, R. C. Zeleznik, and M. R. Mine, "Image plane interaction techniques in 3D immersive environments," in *Proceedings of the 1997 symposium on Interactive 3D graphics - SI3D '97*, 1997, p. 39–ff.
- [28] D. Schmalstieg, L. M. Encarnação, and Z. Szalavári, "Using transparent props for interaction with the virtual table," in *Proceedings of the 1999 symposium on Interactive 3D graphics - SI3D '99*, 1999, pp. 147–153.
- [29] S. Stoev, D. Schmalstieg, and W. Straßer, "Two-handed through-the-lens-techniques for navigation in virtual environments," *Immersive Projection Technology ...*, 2001.
- [30] Z. Szalavári and M. Gervautz, "The Personal Interaction Panel—a Two-Handed Interface for Augmented Reality," *Computer Graphics Forum*, 1997.
- [31] R. Pausch, D. Proffitt, and G. Williams, "Quantifying immersion in virtual reality," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, 1997, pp. 13–18.
- [32] Apple.com, "iPad 2." [Online]. Available: <http://www.apple.com/ipad/ipad-2/specs.html>. [Accessed: 05-Oct-2013].
- [33] D. Sachs, "Sensor Fusion on Android Devices: A Revolution in Motion Processing." [Online]. Available: <http://www.youtube.com/watch?v=C7JQ7Rpwn2k>.