

11-7-2014

# 3D Navigation with Six Degrees-of-Freedom using a Multi-Touch Display

Francisco Raul Ortega

*Florida International University, forte007@fiu.edu*

**DOI:** 10.25148/etd.FI14110721

Follow this and additional works at: <http://digitalcommons.fiu.edu/etd>

 Part of the [Graphics and Human Computer Interfaces Commons](#)

---

## Recommended Citation

Ortega, Francisco Raul, "3D Navigation with Six Degrees-of-Freedom using a Multi-Touch Display" (2014). *FIU Electronic Theses and Dissertations*. 1594.

<http://digitalcommons.fiu.edu/etd/1594>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact [dcc@fiu.edu](mailto:dcc@fiu.edu).

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

3D NAVIGATION WITH SIX DEGREES-OF-FREEDOM USING A MULTI-TOUCH  
DISPLAY

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Francisco R. Ortega

2014

To: Dean Amir Mirmiran  
College of Engineering and Computing

This dissertation, written by Francisco R. Ortega, and entitled 3D Navigation with Six Degrees-of-Freedom using a Multi-Touch Display, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

---

Peter Clarke

---

Raju Rangaswami

---

Wei Zeng

---

Naphtali Rische, Co-Major Professor

---

Armando Barreto, Co-Major Professor

Date of Defense: November 7, 2014

The dissertation of Francisco R. Ortega is approved.

---

Dean Amir Mirmiran  
College of Engineering and Computing

---

Dean Lakshmi N. Reddi  
University Graduate School

Florida International University, 2014

© Copyright 2014 by Francisco R. Ortega

All rights reserved.

## DEDICATION

This is dedicated to my parents, sisters, nephews, nieces, brothers-in-law, and my entire Family. This is dedicated to my beloved wife, the one and only. I dedicate this to the Dreamers, those who came to the US as kids, who feels american, but yet, they are not recognized as one. This is also dedicated to those who believed in me and helped me along the way, to my Atari 65XE Computer for the love of coding, to my Atari 1050 disk drive, for making the wait much shorter. This is dedicated to Marcelo Bielsa — Esto se lo dedico a toda mi familia, Luz Adriana, Francisco M., Patricia E., Aida Guerrero, Pedro Melo, Tata Custodio, Tia Marta, Tia Kena, Tia Teresita, Marcela, Cecilia, Jimena, Roberto, Kenneth, Eduardo, Fernanda, Francisca, Maria Sofia, Roberto Ignacio, Felipe, Sebastian, Nicolas, Felipito, y todo los que vendran. This is dedicated to my future kids, the one coming and the ones to come. — PARA TI MAMA AIDA y PARA TI MAMA PATTY.

## ACKNOWLEDGMENTS

The dissertation would not have been possible without the support and guidance from my two Advisors: Dr. Armando Barreto and Dr. Naphtali Rishe. It is with my most sincere gratitude that I will never forget all the help provided by both. I have to emphasize how Dr. Barreto took a chance on a student coming from a different major. The guidance, the time, and the knowledge given to me by Dr. Barreto has been more than what a student could have ever asked for. It has been invaluable. Dr. Rishe always provided amazing support and help during my studies. Most important, Dr. Rishe, has believe on my potential as a researcher. My learning experience without the support of both advisors would not have been possible.

My committee has also been very supportive. Dr. Peter Clarke, Dr. Raju Rangaswami, and Dr. Wei Zeng, provided guidance and help along the way. I have to thank Dr. Adjouadi, the unofficial committee member. With his inspiring classes and research, he always gave me the motivation to push the envelope further. Other faculty and staff know that your support helped me to get this far. I need to thank Martha Gutierrez, Pat Brammer, Ana Saenz, Olga Carbonell, Professor Jill Weiss, Dr. Weiss, and Dr. Pelin. I need to thank the School of Computer Science and the Department of Electrical and Computer Engineering. I will also like to thanks Levent Erborra for his help during the GRE studies. Finally, I have to thank the FIU writing center and in particular, the best editor one can find, Corey Ginsberg.

I also need to thanks the people who made it possible to research with their funding. First, Dr. Milani, providing an amazing fellowship (GAANN), granted by the Department of Education of the United States, which provided more than 3 years of funding. I also have to thank the Florida Education Fund, in particular, Dr. Lawrence and Mr. Jackson, for their McKnight Dissertation Year Fellowship, which provided 4 semesters and a lot of help. The graduate school of FIU also have been very helpful for the entire Ph.D. program.

I have to thank Mr. Dudley and Dr. Montas-Hunter for their amazing help. The HPDRC lab at FIU, led by Dr. Rishe, provided incredible help and funding to continue the research. Finally, the National Science Foundation (NSF) helped the research with the following grants: CNS-0821345, CNS-1126619, HRD-0833093, IIP-0829576, CNS-1057661, IIS-1052625, CNS-0959985, OISE-1157372, IIP-1237818, IIP-1330943, IIP-1230661, IIP-1026265, IIP-1058606, IIS-1213026.

There are many friends who come to mind. There are too many to mention. However, I need to mention the ones that made the most difference in my graduate studies and life in general: Dan (Daisy) Lu, Frank Hernandez, Tessa Verhoef, Jose Ignacio (Nacho) Mora, Xabriel Colloso-Mojica, Jaime Ballesterro, Miguel Erazo, Aaron Lebos, Alex Montorro, Eddie Garcia, Fidelia Rubio, and those old friends, who started the dream, with the band ROOTS. I also have to thank my DSP Lab mates, Jian (Raymond) Huang, Fatemeh Sara Abyarjoo, Peng Ren, Cindy Gao, Jonathan Cofino, Nonnarit (Ong) O-Larnnithipong.

I'm grateful for people who from the distance helped me. With their encouragement and help of Dr. MacKenzie and Dr. Eberly, I was able to pursue my dream of writing an upcoming book with CRC Press. I also need to thank the community of Ogre 3D, for all their help during development.

Without my family none of this would matter. Their support and understanding has been priceless. Their love has proven to be the best motivation to continue the research when there were difficult times. My dad Francisco, mom Patricia, my grandparents: Aida and Pedro, my sisters: Marcela, Cecilia, Jimena, for their continue love and care for my entire life. For my brother-in-laws: Roberto, Kenneth, Eddie, for the love to my sisters and my nephews. For all my nieces and nephews: Fernanda, Francisca, Felipe A., Sebastian, Roberto, Felipe, Nicholas. I would like to thank my wife's family as well. For all the family that has yet to come and the family that has past, I thank you.

I will always be in debt to my father and mother, for their love and support. For the Atari 65XE and disk drive 1050, that was given to me, and gave me the gift of coding. Because they let me dream, they let me be, they supported me in good and bad times. Because they believed in me, when I couldn't do it myself. I'm here because of them and they are here because of me. I love you.

However, there is one person, whom I must thank for her patience, support and love. My wife Luz Adriana. She understood best when it was late at night or early in the morning, and I had an incredible amount of work. For her time helping me to enter all the subject data from paper to Excel, I'm forever grateful. For you, the family that starts, and the one that is coming. It is for you and our new family, that I continue to move along this path. I love you, always and forever — Francisco Raul Ortega, 2014.



## ABSTRACT OF THE DISSERTATION

### 3D NAVIGATION WITH SIX DEGREES-OF-FREEDOM USING A MULTI-TOUCH DISPLAY

by

Francisco R. Ortega

Florida International University, 2014

Miami, Florida

Professor Naphtali Rishe, Co-Major Professor

Professor Armando Barreto, Co-Major Professor

With the introduction of new input devices, such as multi-touch surface displays, the Nintendo WiiMote, the Microsoft Kinect, and the Leap Motion sensor, among others, the field of Human-Computer Interaction (HCI) finds itself at an important crossroads that requires solving new challenges. Given the amount of three-dimensional (3D) data available today, 3D navigation plays an important role in 3D User Interfaces (3DUI). This dissertation deals with multi-touch, 3D navigation, and how users can explore 3D virtual worlds using a multi-touch, non-stereo, desktop display.

The contributions of this dissertation include a feature-extraction algorithm for multi-touch displays (FETOUCH), a multi-touch and gyroscope interaction technique (Gyro-Touch), a theoretical model for multi-touch interaction using high-level Petri Nets (PeNTa), an algorithm to resolve ambiguities in the multi-touch gesture classification process (Yield), a proposed technique for navigational experiments (FaNS), a proposed gesture (Hold-and-Roll), and an experiment prototype for 3D navigation (3DNav). The verification experiment for 3DNav was conducted with 30 human-subjects of both genders. The experiment used the 3DNav prototype to present a pseudo-universe, where each user was required to find five objects using the multi-touch display and five objects using a game controller (GamePad). For the multi-touch display, 3DNav used a commercial library called Ges-

tureWorks in conjunction with Yield to resolve the ambiguity posed by the multiplicity of gestures reported by the initial classification. The experiment compared both devices. The task completion time with multi-touch was slightly shorter, but the difference was not statistically significant. The design of experiment also included an equation that determined the level of video game console expertise of the subjects, which was used to break down users into two groups: casual users and experienced users. The study found that experienced gamers performed significantly faster with the GamePad than casual users. When looking at the groups separately, casual gamers performed significantly better using the multi-touch display, compared to the GamePad. Additional results are found in this dissertation.

## TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION . . . . .	1
1.1 Problem Statement . . . . .	1
1.2 Objective of Study . . . . .	2
1.3 Motivation . . . . .	2
1.4 Research Questions . . . . .	3
1.5 Significance of Study . . . . .	3
1.6 Literature Review . . . . .	3
1.6.1 Input Considerations . . . . .	5
1.6.2 Toward 3D Navigation . . . . .	6
1.6.3 3D Navigation . . . . .	9
1.6.4 Gesture Modeling and Petri Nets . . . . .	15
1.7 Dissertation Structure . . . . .	18
2. BACKGROUND . . . . .	20
2.1 Computer Graphics . . . . .	20
2.1.1 Camera Space . . . . .	20
2.1.2 3D Translation And Rotations . . . . .	23
2.1.3 Geometric Modeling . . . . .	26
2.1.4 Scene Managers . . . . .	27
2.1.5 Collision Detection . . . . .	29
2.2 Human-Computer Interaction . . . . .	29
2.2.1 Usability . . . . .	30
2.2.2 The Light Pen and the Computer Mouse . . . . .	31
2.2.3 Graphical User Interfaces and WIMP . . . . .	32
2.2.4 Input Technologies . . . . .	34
2.2.5 Input Device States . . . . .	37
2.2.6 Bi-Manual Interaction . . . . .	38
2.3 Multi-Touch Displays . . . . .	42
2.3.1 Projective Capacitive Technology . . . . .	44
2.3.2 Optical Touch Surfaces . . . . .	46
2.4 3D User Interfaces . . . . .	49
2.5 3D Output Interfaces . . . . .	50
2.5.1 Visual Displays Characteristics . . . . .	50
2.5.2 Understanding Depth . . . . .	51
2.5.3 Displays . . . . .	54
2.6 3D Input Interfaces . . . . .	55
2.7 3D Navigation . . . . .	56
2.7.1 3D Travel . . . . .	57
2.7.2 3D Travel Tasks . . . . .	57
2.8 Petri Nets . . . . .	60

2.8.1	Graphical Representation . . . . .	62
2.8.2	Formal Definition . . . . .	62
3.	TOWARD 3D NAVIGATION WITH MULTI-TOUCH INTERACTIONS . . . . .	65
3.1	Multi-Touch Feature Extraction . . . . .	65
3.1.1	FETOUCH . . . . .	65
3.1.2	FETOUCH+ . . . . .	71
3.1.3	Implementation: FETOUCH++ . . . . .	74
3.2	GyroTouch . . . . .	78
3.2.1	Implementation . . . . .	81
3.3	PeNTa: Petri Nets . . . . .	83
3.3.1	Motivation and Differences . . . . .	84
3.3.2	HLPN: High-Level Petri Nets and IRML . . . . .	86
3.3.3	PeNTa and Multi-Touch . . . . .	88
3.3.4	Arc Expressions . . . . .	89
3.3.5	A Tour of PeNTa . . . . .	93
3.3.6	Simulation and Execution . . . . .	95
3.3.7	Overview . . . . .	96
3.4	Yield: Removing ambiguity . . . . .	96
3.4.1	Yield: How it Works . . . . .	97
3.4.2	Yield: The Algorithm . . . . .	102
3.4.3	Yield: The Implementation . . . . .	109
3.5	FaNS: Navigational System - A Fair Approach . . . . .	112
3.5.1	FaNS: The Implementation . . . . .	112
3.6	Hold-and-Roll: Finding a Gesture for the Z Axis . . . . .	116
4.	3DNAV: MULTI-TOUCH SYSTEM PROTOTYPE . . . . .	119
4.1	Preliminary Device Testing . . . . .	119
4.1.1	Device Listeners and Common Interfaces . . . . .	119
4.1.2	3D Mouse . . . . .	122
4.1.3	Inertial Navigation System . . . . .	128
4.1.4	Microsoft Kinect . . . . .	130
4.1.5	Keyboard and Mouse . . . . .	136
4.1.6	GamePad . . . . .	137
4.1.7	Multi-Touch . . . . .	142
4.2	OGRE . . . . .	152
4.3	ECHoSS: Experiment Module . . . . .	161
4.4	Overview . . . . .	166
5.	DESIGN OF EXPERIMENT: 3D NAVIGATION . . . . .	167
5.1	Experiment Objective . . . . .	167
5.2	Pre-Trials . . . . .	168
5.3	Device Selection . . . . .	168

5.4	Experimental Subjects	169
5.5	Experiment Apparatus	170
5.5.1	Hardware Setup	170
5.5.2	Software Setup	172
5.6	Multi-Touch Gesture Design	175
5.6.1	Gesture Definition	175
5.6.2	Gesture Selection	176
5.6.3	Gesture Mapping	178
5.7	GamePad Design	179
5.8	Additional Controller Design	180
5.9	Techniques	181
5.9.1	Primed Search	181
5.9.2	Visual Cues	182
5.9.3	Device Switching	184
5.10	Questionnaires	185
5.11	Gamers' Experience	186
5.12	Objective Measurements	189
5.13	Experiment Procedure	191
5.14	3D Navigation Experiment Tour	191
6.	EXPERIMENT ANALYSIS	199
6.1	Data Outliers	199
6.2	The Dataset	200
6.3	Quantitative Data	200
6.3.1	Time: GamePad and Multi-Touch	201
6.3.2	Homing: Switching Devices	210
6.4	Qualitative Data	212
6.4.1	Paired Questions	212
6.4.2	Additional Pairs of Questions	214
6.4.3	GamePad or Multi-Touch	216
6.4.4	Rotation and Translations Questions	218
6.4.5	Other Questions	219
6.4.6	Hold-And-Roll Questions	219
7.	EXPERIMENT DISCUSSION	221
7.1	Assimilating Experimental Results	221
7.2	3D Navigation	222
7.2.1	Open Questions	224
7.2.2	Hold-and-Roll	225
7.3	Lessons learned	225
7.4	Limitations of the Study	226
7.4.1	Internal Validity	227
7.4.2	External Validity	228

8. CONCLUSIONS & FUTURE WORK . . . . .	232
8.1 Concluding Remarks . . . . .	232
8.2 Future Work . . . . .	234
BIBLIOGRAPHY . . . . .	236
APPENDICES . . . . .	257
VITA . . . . .	288

## LIST OF TABLES

TABLE	PAGE
1.1 Questions and Hypotheses . . . . .	4
2.1 Partial History of Multi-Touch Until 1984 . . . . .	41
2.2 Partial History of Multi-Touch from 1985 to 1995 . . . . .	43
2.3 Partial History of Multi-Touch from 1997 to 2014. . . . .	44
3.1 Multi-Touch Data Structure . . . . .	89
3.2 Transitions . . . . .	93
5.1 Pre-Trial Users. . . . .	168
5.2 Gesture Definitions. . . . .	177
5.3 Gesture Mappings. . . . .	178
5.4 Controller Mappings. . . . .	179
5.5 Multiple Choice Legend. . . . .	186
5.6 Entry Questionnaire. . . . .	187
5.7 Additional Entry Questionnaire. . . . .	188
5.8 Additional Exit Questionnaire. . . . .	188
5.9 Game Classification. . . . .	190
5.10 Game Classification Description . . . . .	190
5.11 Sentences. . . . .	194
5.12 Exit Questionnaire. . . . .	196
6.1 Descriptive Statistics for $T_d$ . . . . .	202
6.2 Normality Test for $T_d$ . . . . .	203
6.3 Normality Test for $T_{dx}$ . . . . .	203
6.4 Descriptive Statistics for $T_{dx} = \text{Log}(T_d)$ . . . . .	204
6.5 Co-Factor Means $T_{dx}$ . . . . .	208
6.6 Co-Factor Analysis $T_{dx}$ . . . . .	209
6.7 Normality Test for $T_{dx}$ by Group . . . . .	210

6.8	Wilcoxon-signed-rank test: GP_Keyboard - MT_Keyboard . . . . .	212
6.9	Wilcoxon-signed-rank statistics: GP_Keyboard - MT_Keyboard . . . . .	212
6.10	Wilcoxon-signed-rank test: $Q_2 - Q_1$ . . . . .	214
6.11	Wilcoxon-signed-rank statistics: $Q_2 - Q_1$ . . . . .	214
6.12	Normality Test for $Q_4$ to $Q_{11}$ . . . . .	216
6.13	Descriptive Statistics for $Q_4$ to $Q_{11}$ . . . . .	217
6.14	Wilcoxon-signed-rank tests: Experienced gamer . . . . .	217
6.15	Wilcoxon-signed-rank statistics: Experienced gamer . . . . .	218
6.16	Question 14: Video GamePad Controller (GamePad) or multi-touch . . . . .	219
6.17	Questions 16 and 17 . . . . .	220
6.18	Question 21 . . . . .	220
7.1	Hypothesis Results . . . . .	221
7.2	Open Question Results <sup>¶</sup> . . . . .	230
7.3	Hold-and-Roll Comments <sup>¶</sup> . . . . .	231



## LIST OF FIGURES

FIGURE	PAGE
2.1 Camera Space [77] . . . . .	21
2.2 View Frustum [77] . . . . .	23
2.3 Principal Axes. Drawn by Auawise <sup>1</sup> . . . . .	25
2.4 Mouse Study: Time [46] . . . . .	33
2.5 Mouse Study: Error Rate [46] . . . . .	33
2.6 Images . . . . .	38
2.7 Bi-Manual Interaction [133] . . . . .	42
2.8 Multi-touch Capacitance Technology . . . . .	45
2.9 Multi-touch Capacitance Technology . . . . .	46
2.10 Multi-touch Capacitance Technology . . . . .	47
2.11 Vending Machine (Adapted from [174]). . . . .	61
3.1 State Machine . . . . .	66
3.2 Queue . . . . .	67
3.3 Rotation Gesture . . . . .	72
3.4 WiiMote with Motion Plus . . . . .	78
3.5 GyroTouch . . . . .	80
3.6 Parallel PN: State 1 . . . . .	91
3.7 Parallel PN: State 2 . . . . .	91
3.8 Cold transitions (Entry and Exit) . . . . .	91
3.9 Multiple Gestures in PeNTa . . . . .	94
3.10 Partial Petri Net for Scale . . . . .	96
4.1 3D Mouse Space Sensor <sup>†</sup> . . . . .	122
4.2 3D Mouse Functions <sup>†</sup> . . . . .	123
4.3 C# Skeleton Viewer (Possible Gestures) . . . . .	133

4.4	Ancient 3D ruins . . . . .	135
4.5	Screen Closeup . . . . .	135
4.6	From the user’s point of view . . . . .	136
4.7	Images . . . . .	138
4.8	Images . . . . .	139
4.9	OpenGL Cube of Spheres . . . . .	153
4.10	3DS Max Scene . . . . .	154
5.1	3M M2256PW multi-touch display . . . . .	170
5.2	XBox 360 GamePad . . . . .	171
5.3	Multi-Touch Reset Button . . . . .	180
5.4	Additional Keyboard Input . . . . .	181
5.5	Hyper Cube for Training . . . . .	182
5.6	Search Object Marker (Flag) . . . . .	183
5.7	Three-Ring Sphere . . . . .	184
5.8	Hyper sphere (Target Object) . . . . .	193
5.9	3D Navigation Experiment Display . . . . .	193
5.10	Space Ship (Target Object) . . . . .	193
5.11	Target Objects . . . . .	194
5.12	Non-Target Objects . . . . .	194
5.13	Keyboard Pop Up . . . . .	195
5.14	Subject . . . . .	195
5.15	Hold-and-Roll . . . . .	195
6.1	Outliers . . . . .	200
6.2	BB Plot . . . . .	204
6.3	BB Plot (Transform Data) . . . . .	205
6.4	QQ Plot . . . . .	205

6.5	QQ Plot (Transform Data)	206
6.6	Histograms	206
6.7	Histograms (Transformed Data)	207
6.8	Experiment Table	219
B.1	Handout with Search Objects	269
D.1	Xbox 360 Legend	274
D.2	Xbox 360 Legend	275

## ABBREVIATIONS

2D	Two-Dimensional.
3D	Three-Dimensional.
3DNav	3D Navigation System.
3DUI	Three-Dimensional User Interface.
AI	Artificial Intelligence.
ANOVA	Analysis of variance.
API	Application Programming Interface.
BSP	Binary Space Partitioning.
CG	Computer Graphics.
CPN	Coloured Petri Nets.
CPU	Central Processing Unit.
CRT	cathode ray tube.
DI	Diffuse Illumination.
DLL	Dynamic Link Library.
DOF	Degrees of Freedom.
DPI	Dots per Inch.
DSL	Domain-Specific Language.
DSP	Digital Signal Processing.
ECHoSS	Experiment Controller Human Subject System.
FaNS	Fair Navigation System.
FETOUCH	Feature Extraction Multi-Touch System.
FIU	Florida International University.
FOR	Field of Regard.
FOV	Field of View.
FPS	First-Person Shooter.

FSM	Finite-State Machine.
FTIR	Frustrated Total Internal Reflection.
GamePad	Video GamePad Controller.
GB	Gigabyte.
GHz	Giga Hertz.
GML	Gesture Markup Language.
GOMS	Goals, Operators, Methods, and Selection.
GPU	Graphics Processing Unit.
GUI	Graphical User Interface.
GWC	GestureWorks Core.
GyroTouch	Gyroscope Multi-Touch System.
HCI	Human-Computer Interaction.
HLPN	High-Level Petri Net.
HMD	Head-mounted Display.
HMM	Hidden Markov Models.
Hz	Hertz.
INI	Initialization.
INS	Inertial Navigation System.
IR	Infrared.
IRB	institutional review board.
IRML	Input Recognition Modeling Language.
KLM	Keystroke-Level Model.
LCD	Liquid-crystal Display.
LLP	Laser Light Plane.
Max	Maximum.
MB	Megabyte.

MEMS	Micro-electro-mechanical systems.
MHz	Mega Hertz.
Min	Minimum.
MO	Mode.
MRI	magnetic resonance imaging.
MSDN	Microsoft Software Developer Network.
N	Population Size.
NES	Nintendo Entertainment System.
OGRE	Object-Oriented Graphics Rendering Engine.
OIS	Object-Oriented Input System.
p	Significance Value.
PARC	Palo Alto Research Center.
PC	Personal Computer.
PCT	Projective Capacitive Technology.
PeNTa	Petri Net Touch.
PN	Petri Net.
Prt Net	Predicate Transition Net.
RAM	Random-Access Memory.
RBI	Reality Based Interactions.
RegEx	Regular Expression.
SD	Standard Deviation.
SDK	Software Development Kit.
SEM	Standard Error of Mean.
TUI	Tangible User Interface.
TUIO	Tangible User Interface Object.
UI	User Interface.

USB	Universal Serial Bus.
Var	Variance.
VR	Virtual Reality.
WiiMote	Nintendo Wii Controller.
WIM	World-in-Miniature.
WIMP	Windows-Icon-Menu-Pointer.
WINAPI	Windows API.
WinRT	Windows run-time.
$\bar{x}$	Mean.
$\tilde{x}$	Median.
XML	Extensible Markup Language.
Yield	Yanked Ambiguous Gestures.

# CHAPTER 1

## INTRODUCTION

The seminal work known as SketchPad [202], by Ivan Sutherland has inspired many researchers in the field of Human-Computer Interaction (HCI) and Three-Dimensional User Interface (3DUI). Sutherland created an elegant and sophisticated system, which is considered by some as the birth of computer user interface studies. The invention of the mouse by Douglas Engelbart in 1963 [46] and the invention of the Graphical User Interface (GUI) in the Palo Alto Research Center (PARC) gave way to one of the most successful paradigms in HCI: Windows-Icon-Menu-Pointer (WIMP), which has allowed users to interact easily with computers.

Today, with the introduction of new input devices, such as multi-touch surface displays, the Nintendo Wii Controller (WiiMote), the Microsoft Kinect, the Leap Motion sensor, SixSense Stem System, and Inertial Navigation Systems (INS), the field of HCI finds itself at an important crossroads that requires solving new challenges.

Humans interact with computers, relying on different input-output channels. This may include vision (and visual perception), auditory, tactile (touch), movement, speech, and others [37]. In addition, humans use their (long and short-term) memory, cognition, and problem-solving skills to interact with computer systems [37]. This computer interaction has a set of challenges that needs to be addressed. This dissertation addressed the problem of three-dimensional (3D) navigation using a multi-touch, non-stereo, desktop display. This includes the modeling of multi-touch interaction, and the understanding of how users can benefit by using multi-touch desktop displays.

### **1.1 Problem Statement**

With the amount of 3D data available today, 3D navigation plays an important role in 3DUI. This dissertation deals with multi-touch and 3D navigation. In specific, it deals with how



users can explore 3D virtual worlds with multi-touch, non-stereo, desktop displays. What type of techniques can be applied for a better multi-touch interaction with 3D worlds? Can users benefit from such interaction? This dissertation answers the questions using novel techniques and human-subject testing. In particular, the questions addressed by the dissertation are listed in 1.4.

## **1.2 Objective of Study**

The objective of this research is the development of models, algorithms, techniques, and an experimental environment for 3D navigation. This is with the purpose to advanced the state of the art of 3D navigation using multi-touch desktop displays. This study seeks to improve 3D navigation with 6-degrees of freedom (DOF) using multi-touch, and study the performance of this device in this type of environment.

## **1.3 Motivation**

The motivation for this research is to find novel ways for 3D navigation using multi-touch in generic virtual worlds. The keyword is generic, which can be expanded to fit different domains, such as medical, architectural, and scientific data, among others (see 1.6.3). This is where the motivation for utilizing 6-DOF comes about. Some domains may required six or more DOF. In general, this author is motivated to move the body of knowledge in the field of 3DUI. This, in the author's opinion, helps to create building blocks towards the vision of Mark Weiser [219]:

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

## **1.4 Research Questions**

The research aims to address questions about 3D navigation using multi-touch. In specific, how does multi-touch compare to a game controller (gamepad) and what are the implications for users employing 6-DOF for navigation? Table 1.1 shows the questions (*Q*) and hypotheses (*H*) for this dissertation. At the end of this dissertation, all of the questions postulated in Table 1.1 are answered.

## **1.5 Significance of Study**

3D data allows for the exploration of real-world environments (e.g., New York City) and scientific data visualization (e.g., Complex 3D Magnetic resonance imaging (MRI) data). How this data is navigated is crucial to understanding data and finding objects in the virtual world. Finding ways to navigate with 6-DOF using a standard multi-touch display will allow users to interact more intuitively with the data. Given the pervasive nature of multi-touch surfaces today, the study will help to understand how to improve users' 3D navigation when using multi-touch displays.

## **1.6 Literature Review**

The objective of this section, as the name indicates, is to review the state of the art that directly impacts the contributions of this dissertation. This section is divided into four sub-parts. The first part covers input and design guidelines literature, which helped the design of experiment. The second part covers generic topics that deal with 3D techniques. The third part deals with different approaches to 3D navigation. The final part deals with multi-touch recognition and modeling. While the attempt in this chapter is to be as detailed as

$Q_s$	Is it possible to use a set of 2D Gestures in a multi-touch display to perform 3D navigation with 6-DOF?
$H_s$	The proposed set will allow 3D navigation with 6-DOF.
$Q_t$	Will the real-time 3D navigation for a primed search be improved with multi-touch desktop display or with the GamePad, using time elapsed as an objective measure?
$H_t$	The proposed real-time multi-touch interaction for 3D navigation will take less time to find the objects in a primed search in comparison to the GamePad.
$Q_u$	Given the multi-touch input, would there be a significant difference between casual and experienced video gamers?
$H_u$	Given the multi-touch input, there will be a significant difference between both groups.
$Q_v$	Given the GamePad input, would there be a significant difference between the casual and experienced video gamers?
$H_v$	Given the GamePad input, there will be a significance difference between both groups.
$Q_w$	Would it take less time to complete the primed search for expert video gamers when using the GamePad controller?
$H_w$	Expert video gamers will take less time to complete the primed search when using the GamePad controller.
$Q_x$	Would it take less time to complete the primed search for casual gamers when using the multi-touch device?
$H_x$	Casual gamers will take less time to complete the primed search when using the multi-touch device.
$Q_y$	Would there be a difference in time for sentence completion when using the multi-touch compare to the GamePad?
$H_y$	It will take less time to complete the sentences when switching from the multi-touch than when switching from the GamePad .
$Q_z$	Would there be higher rate of error for sentence completion when using the multi-touch compare to the GamePad?
$H_z$	There will be a higher error rate when switching from the GamePad than when switching from the multi-touch.

Table 1.1: Questions and Hypotheses

possible, it cannot be totally exhaustive. The reader is suggested to look into the actual cited work for more information, as well as *3D User Interfaces: Theory and Practice* (Bowman et al. [17]), *Human-Computer Interaction: An Empirical Research Perspective* (MacKenzie [133]), and *Human-Computer Interaction* (Dix et al. [37]).

### **1.6.1 Input Considerations**

Six-DOF for input devices have been studied in full detail by Zhai, in his doctoral dissertation [234], in 1995. The dissertation goes in-depth into how subjects deal with 6-DOF. The study [234] provided great insight for 3D navigation: The muscle groups involved in a 6-DOF vary depending on the device used. This in itself helps to design better interfaces. The study also talks about the transfer function (see Chapter 2), which must be compatible with the characteristics of the actual device. This was also stated by Bowman et al. [17, Chapter 5], in their guidelines: “match the interaction technique to the device” [17, p. 179]. Hinckley also advanced the knowledge about input technologies in [88]. General aspects of input devices are covered in Chapter 2. The recommendations by Zhai [234] help to emphasize the 3D interaction design guidelines offered by Bowman et al. [17]:

1. Use existing manipulation techniques unless an application will benefit greatly from creating new ones.
2. Match the interaction to the device.
3. Reduce wasted motion (clutching).
4. Use non-isomorphic techniques whenever possible.
5. Reduce the number of degrees of freedom (DOF) whenever possible.

From the previous list, item 4 reminds the designer that it is difficult to model isomorphic rotation techniques, as shown in [172]. As it will become apparent in Chapter

5, some of these design guidelines were considered for the experiment developed for this dissertation.

Additional guidelines have been proposed. For example, in [96], Jacob et al. proposed interfaces within a post-WIMP framework. In this framework, they tried to find a balance from Reality Based Interactions (RBI) and artificial features. RBI includes naïve physics, body awareness and skills, environment awareness and skills, and social awareness and skills. Also, Hancock et al, in [78], proposed some specific guidelines when dealing with multi-touch rotations and translation. These included the ability<sup>1</sup> to provide more DOF than WIMP. Also, Hancock et al. [78] suggested that a constant connection between the visual feedback and the interaction would prevent cognitive disconnect by avoiding actions that the user may not be expecting. In other words, The system needed to provide a realistic 3D visual feedback to match the interaction. For additional discussion, see [14, 15, 17, 88].

## **1.6.2 Toward 3D Navigation**

3D navigation has benefited from previous work in related areas. The most closely related areas are 3D interaction and virtual devices. The pioneer and seminal work by Ivan Sutherland [202] with Sketch Pad provided a way forward for User Interfaces (UIs). Also, early studies by Nielson and Olsen [155], which provided direct “Manipulation Techniques for 3D Objects,” and Chen et al. [29], who studied 3D rotations, provided the groundwork for more recent developments. Touch interactions, virtual devices, and multi-touch techniques are described next. For a brief look at 3D interactions (up to 1996), see [79].

---

<sup>1</sup>The ability to do rotation and scale independently or together.

## Understanding Touch Interactions

To achieve a more natural interaction between the screen and the user, studies like [9, 24, 215, 216, 223] provided a different take on how touch information can be used. One example is, [215] studied finger orientation for oblique touches. In another example, Benko and Wilson [9] studied dual-finger interactions (e.g, dual-finger selection, dual-finger slider, etc.). Additional work dealing with contact shape and physics can be found in [24, 223]. In a very comprehensive review of finger input properties for multi-touch displays, [216] provides suggestions that have been used already in [215].

Other studies looked to see when rotations, translations, and scaling actions are best if kept separate or combined [78, 148]. If combined actions are chosen, the user's ability to perform the operations separately may be limited [148]. Additional studies have looked at when to use the uni-manual or bi-manual type of interactions [111, 142]. Some studies have concluded that one-hand techniques are better suited for integral tasks (e.g., rotations), while two-hand techniques are better suited for separable tasks [111, 142].

## Virtual Devices

Nielsen and Olsen used a triad mouse to emulate a 3D mouse [155]. In this study, they performed 3D rotations, translations and scaling (independently of each other). For 3D rotations, they used point  $P_1$  as the axis reference, and points  $P_2$  and  $P_3$  to define the line forming the rotation angle to be applied to the object. In more recent work [78], one can find subtle similarities with [155], in the proposition of defining a point of reference to allow seamless rotation and translation.

The Virtual Sphere [29] was an important development for 3D rotation methods previously proposed. This study tested the Virtual Sphere against other virtual devices. It was found that the Virtual Sphere and the continuous XY+Z device behaved best for complex rotations (both devices behaved similar with the exception that the XY+Z device does not

allow for continuous rotations about all X, Y, Z axes). The Virtual Sphere simulated a real 3D trackball with the user moving left-right (X axis), top-down (Y axis) and in a circular fashion (Z axis) to control the rotation of the device. Related work included the Rolling Ball [61], The Virtual Trackball [6], and the ARCBALL [83]. The ARCBALL “is based on the observation that there is a close connection between 3D rotations and spherical geometry” [17].

### **Multi-Touch Techniques**

Hancock et al. provided algorithms for one, two and three-touches [78]. This allowed the user to have direct simultaneous rotation and translation. The values that are obtained from initial touches  $T_1$ ,  $T_2$  and  $T_3$  and final touches  $T'_1$ ,  $T'_2$  and  $T'_3$  are  $\Delta\text{yaw}$ ,  $\Delta\text{roll}$  and  $\Delta\text{pitch}$ , which are enough to perform the rotation on all three axes, and  $\Delta x$ ,  $\Delta y$ ,  $\Delta z$  to perform the translation. A key part of their study showed that users preferred gestures that involve more simultaneous touches (except for translations). Using gestures involving three touches was always better for planar and spatial rotations [78].

A different approach is presented in RNT (Rotate 'N Translate) [119], which allows planar objects to be rotated and translated, using the concept of friction. This meant that the touch was performed opposing the objects' movement or direction, causing a pseudo-physics friction. This concept was also referred to as “current”. This particular algorithm is useful for planar objects and it has been used in other studies (e.g., [175]).

The spatial separability problem when dealing with 3D interactions (for multi-touch displays) was studied in [148]. The authors proposed different techniques that helped the user perform the correct combination of transformations [148]. The combination of transformation included scaling, translation + rotation, and scaling + rotation + translation, among others. From the proposed techniques in [148], two methods yielded the best results. They were Magnitude Filtering and First Touch Gesture Matching [148]. Mag-

nitude Filtering works similarly to snap-and-go [148], but it does not snap to pre-selected values or objects. It also introduced the concept of catch-up. This concept allowed “continuous transition between the snap zone and the unconstrained zone.” [148]. The other technique, First Touch Gesture Matching, worked by minimizing “the mean root square difference between the actual motion and a motion generated with a manipulation subset of each model” [148]. The algorithm selects the correct technique using best-fit error and the magnitude of the transformation.

### 1.6.3 3D Navigation

Since the days of the animated film, “A Computer Animated Hand,” the development of the Sensomora Simulator by Hellig [19, 84], and the contributions by Ivan Sutherland [202, 203], the field of Computer Graphics (CG)<sup>2</sup> led practitioners and researchers to look for ways to push the envelope further. One of these challenges has been to push the state-of-the-art for 3D navigation. This is the primary objective of this dissertation, and the relevant literature on the topic is described next.

3D navigation<sup>3</sup> has been used in a variety of domains, including medicine [55, 72, 114, 132, 138], large-scale virtual environments [181], geographical and geological environments [8, 23, 30, 38, 196, 201], other scientific visualizations [31, 54, 227, 230, 231], astronomy [53], dynamic 3D worlds [183], city models [179], energy management systems [5, 149], video games<sup>4</sup> [222], Tangible User Interfaces (TUI) [228] , and others [2, 33, 177, 208]. Note that not all the 3D navigation studies include 6-DOF. In some

---

<sup>2</sup>See <http://design.osu.edu/carlson/history/lessons.html>.

<sup>3</sup>Some work has overlap with 3D interactions.

<sup>4</sup>See games such as Doom, Quake, and Microsoft Flight Simulator.



domains, having 4-DOF may be enough. For example, Sultanum et al. studied 3D navigation for geological outcrops with only 4-DOF [201].

One of the common tasks in 3D navigation is to search for objects in the virtual world. As is described in Chapter 2, there are two types of search: naïve and primed. This was first studied for large virtual worlds by Darken and Sibert [34]. The study looked at way-finding (see [17, Chapter 7]) for search and exploration. They found that if no visual cues or directional guides are given, users will tend to become disoriented [34]. Another important finding was that users tend to follow natural paths (e.g., coast line). This worked was followed up by Bowman et al. [15].

Bowman et al. described a testbed evaluation for virtual environments [15]. This included selection, manipulation, and travel experiments. In the travel experiment, which is of interest to this dissertation, they performed naïve search and primed search (see Chapter 2), as described in [17, 34]. In this study [15], users were provided with flags, with numbers 1-4. In addition, the target was marked with a painted circle consisting of a 10-meter radius (large) or a 5-meter radius (small). For the naïve search, the targets were in numerical order, with a low accuracy (large circle) required. The targets were not all visible during the naïve search. In the primed search, all the objects were visible and they were not sorted in numerical order. The required accuracy was changed to a 5-meter radius (small circle). Seven different travel techniques were used in this between-subjects experiment. For the naïve search, the gaze-directed technique [16] was the faster out of the seven techniques tested. Right after the gaze-directed technique, pointing [16] and Go-Go [171] techniques came second. For the primed search, gaze-directed and pointing techniques were significantly faster than the HOMER [17] approach. In both cases, the map technique was found to be the slowest one [15]. At the end of the experiment, the authors concluded that pointing gave the best results for navigation [15].

When working in large-scale environments, with multiple displays, Ruddle et al. [181] looked at the effect between head-mounted and desktop displays. Each participant, a total of 12 subjects, traveled long distances (1.5 km) [181]. Two findings from the study are relevant to this dissertation. First, participants “developed a significantly more accurate sense of relative straight-line distance” when using the head-mounted display (HMD) [181]. Second, when subjects used the desktop display, they tended to develop a “tunnel vision,” which led to missing targets [181].

Santos et al. studied the difference between 3D navigation with a non-stereo, desktop display versus a HMD<sup>5</sup> [195]. In this comparison study (42 subjects), the subjects were divided, into non-experienced and experienced gamers, as well as three levels for their stereoscopic usage (none, moderate, experienced). The experienced gamers group performed differently, with respect to how many objects were caught in the game when using the desktop display [195]. This indicated that their previous skills helped this group of subjects to perform the task. This is due to their familiarity with similar environments. No statistical difference for the groups were found when they used the HMD. Santos et al. found that users preferred the desktop display [195].

A usability study conducted by Fu et al. [53] looked at large-scale 3D astrophysical simulations. Their navigation approach used different gestures and touch widgets to allow different actions in a multi-dimensional world, to study astrophysics. Users found tasks using multi-touch very intuitive and useful for the specific-domain tested. This study did not include a quantitative analysis, but did show that users (16 participants) found the use of multi-touch intuitive. The rotations were performed with one finger, with a movement that was horizontal for the rotation about the Y axis, vertical for the rotation about the X axis, and diagonal pan to rotate about an arbitrary axis on the XY-plane. Rotations were

---

<sup>5</sup>HMD are stereo.

also performed with different gestures, with five (or four) fingers in the same direction. Translations and scale gestures were also provided.

The WiiMote has been a popular device for 3D navigation. A study using Google Earth compared two different configurations for the WiiMote [196]. To move front/back, left/right, the first configuration used the accelerometer and the second configuration used the Infrared (IR) sensor. The rest of the movements in Google Earth were shared by both configurations. The study found that the accelerometer configuration showed a statistically significant improvement, with respect to the other configuration [196]. A similar study used the WiiMote to compare three different techniques for video game navigation [222]. The objective was to navigate while avoiding some objects. The first technique used the accelerometer sensor, the second technique used the WiiMote for head-tracking using the IR sensor, and the third technique combined the prior techniques, adding a Kalman filter [232]. Techniques two and three were also modified to provide alternate versions, which used the Nintendo WiiMote MotionPlus (gyroscope sensor). The third method (hybrid) was preferred when performing the maneuver and aversion tasks. When comparing the techniques with or without the MotionPlus, users preferred the MotionPlus for the evasion task but not for the maneuver tasks [222].

In a comparative study, Lapointe et al. [123] looked at the interactions of 3D navigation with 4-DOF. The devices compared for this study were a keyboard, a mouse, a joystick, and a GamePad. Their quantitative study showed that the mouse significantly outperformed the other devices [123]. In another comparison study, Beheshti et al. [8] studied the difference between a mouse using a desktop display and a multi-touch tabletop<sup>6</sup>. In their study, the tabletop (multi-touch) outperformed the desktop (mouse); however, this difference was not statistically significant [8]. Furthermore, there was no spatial difference between genders

---

<sup>6</sup>Microsoft Surface, first generation.

[8]. This result is in contrast with other studies that have showed a significant difference between genders in similar environments [7, 27, 32, 124].

In a game study by Kulsheresht and LaViola Jr. [120], they evaluated performance benefits when using a head-tracking device. From a total of 40 subjects, half of them used the head-tracking device to assist them in playing four games given by the experimenters. The other half did not use the head-tracking device. The games tested were Arma II, Dirt2, Microsoft Flight, and Wings of Prey [120] using a Personal Computer (PC) and an Xbox 360 controller. The subjects were divided into two groups: Casual and experienced gamers. For the casual gamers, they reported a significant preference for the head-tracking device when playing Dirt2, providing a more engaging user experience. For the experienced gamers, they reported a significant preference for the head-tracking device when playing Microsoft Flight, providing a more engaging user experience. The study also showed that experienced gamers showed a significant improvement when using the head-tracking device for Arma II and Wings of Prey compared to a traditional game controller. Their analysis found that experienced gamers may benefit by using a head-tracking device in certain scenarios, such as First-Person Shooter (FPS) and air combat games [120].

In the experiment conducted by Yu et al., titled “FI3D: Direct-Touch Interaction For the Exploration of 3D scientific Visualization Spaces” [231], they studied 3D navigation using touch. The technique used in [231] allowed users to navigate in 3D. The virtual world was a representation of scientific data. Users navigated using single-touch gestures or the mouse. The objective was to test a 7-DOF that included X, Y, Z translations; yaw, pitch, and roll rotations; and scaling (the 7th degree) to zoom in or out of the screen. Their approach [231] limited the touch to one finger interaction in most instances and provided the use for an additional touch to create specific constraints to aid the movement. Their study showed that the mouse had a faster time for translation and rotation, but only the improvement in rotations was statistically significant for the mouse. The case of the scale

(zoom in/out), showed a significant difference between both input devices, with the mouse having a faster action time [231]. Their method concentrated on visualization of data, consisting of visualization spaces where (most) data had pre-determined spatial meaning [231]. This required the user to support the mental model of a dataset [114].

Some contributions provided interesting techniques for 3D navigation, which are worth mentioning in this literature survey. For example, in 1997, Hanson and Wernert developed methods for constrained 3D navigation using two-dimensional (2D) controllers [80]. Another technique was to use TUIs, as shown by Wu et al. [228]<sup>7</sup>. The NaviRadar, a pedestrian feedback system for navigation, provided tactile feedback to the users [182]. A very interesting approach for World-in-Miniature (WIM)<sup>8</sup> is the work by Coffey et al. [31]. Their approach used WIM slices to interact with sections of the world [31]. Real-world metaphors have been also used. For example, the Segway PT, two-wheel ride<sup>9</sup>, was used as inspiration in [210] for 3D traveling. There have also been techniques to explore cities [179]. A different approach was to use sketching for 3D navigation. The study by Hagedorn and Döllner used a sketch-based approach to accomplish navigation tasks [71]. A novel approach by McCrae et al. for multi-scale 3D navigation was studied in [136]. Different camera techniques have been used for 3D navigation, such as the two-handed Through-The-Lens technique [200], HoverCam [110], controlled camera animation [185], and Navidget [70]. Navigation for time-scientific data was studied by Wolter et al. [227]. Visual memory for 3D navigation was also explored [176]. Many other studies have proposed techniques for 3D navigation tasks [16, 18, 29, 44, 49, 69, 95, 155, 183, 186, 193, 204, 211, 230].

---

<sup>7</sup>See also [66].

<sup>8</sup>See [17, Chapter 5]. For additional references about related work about this topic, see [31].

<sup>9</sup>See <http://www.segway.com>.

## 1.6.4 Gesture Modeling and Petri Nets

### Recognition

There have been various methods proposed to achieve touch gesture recognition, including the use of finite state machines [90, 91], Hidden Markov Models (HMM) [189], neural networks [169], dynamic programming [134], featured-based classifiers [180], and template matching [4, 107, 127, 225]. Thorough reviews can be found in [99, 170, 205].

Some of the methods used in handwriting recognition [205] serve as a foundation for gesture recognition techniques [12]. While handwriting recognition efforts date as far back as the 1950s [205], it has been the work of Rubine [180] that has been used as a foundation by some in the gesture recognition area [191, 225]. The Rubine algorithm used a simple training technique with specific features [180].

The “\$ algorithms”<sup>10</sup> is a partial list of techniques derived or inspired by work from the \$1 algorithm [225], such as [4, 115, 116, 127, 213]. The \$1 algorithm [225] provided a simple way to develop a basic gesture-detection method. In contrast, algorithms based on Hidden Markov Models or neural networks [169] involved a high level of complexity for the developer and the system as well. The \$1 algorithm provided a very fast solution to interactive gesture recognition with less than 100 lines of code [225] and required a simple training set. However, it is not meant for the recognition of multi-touch gestures. This does not diminish in any way the importance of the \$1 algorithm, because there are several features that make it important. Some examples are the obvious resampling of the gesture, the indicative angle (“the angle formed between the centroid of the gesture and [the] gesture’s first point” [225]), and the re-scaling and translation to a reference point to keep the centroid at (0,0). The \$1 algorithm was in part inspired by the publications titled “SHARK2: A Large Vocabulary Shorthand Writing System for Pen-based Computers”

---

<sup>10</sup>Referred to as the dollar family by their authors.

[117] and “Cursive script recognition by elastic matching” [188]. The goals of the \$ family of algorithms are described below:

- Present an easy-to-implement stroke recognizer algorithm.
- Compare \$1 with other sophisticated algorithms to show that it can compare certain types of symbols.
- Understand gesture recognition performance for users, recognizer performance, and human subjective preferences.

The \$N algorithm [4], with double the amount of code (240 lines), improved the \$1 algorithm [225] to allow single strokes and rotation invariance discrimination. For example, to make a distinction between A and  $\forall$ , rotation must be bounded by less than  $\pm 90^\circ$  [4]. The \$N algorithm [4] was extended primarily to allow single strokes to be recognized. This algorithm provided support for automatic recognition between 1D and 2D gestures by using “the ratio of the sides of a gesture’s oriented bounding box (MIN-SIDE vs MAX-SIDE)” [4]. In addition, the algorithm provided recognition for a sub-set of templates, to optimize the code. This reduction of templates was done by determining if the start directions are similar and by computing the angle formed from the start point through the eighth point. A common feature of the \$1 and \$N algorithms [4, 225] is the utilization of the Golden Section Search [173].

Other methods similar to the \$1 [225] and \$N [4] algorithms have been implemented. For example, the *Protractor Gesture Recognizer* algorithm [127] works by applying a nearest neighbor approach. This algorithm is very close to the \$1 algorithm [225] but attempts to remove different drawing speeds, different gesture locations on the screen and noise in gesture orientation. The study by Vatavu et al. (referred to as \$P), used the concept of cloud of points to recognize strokes [213]. Additional algorithms provide great resources for future work. Dean Rubine provided an excellent set of features to be tested with multi-

touch data. In addition to the Rubine algorithm [180], the work of Wang et al. [215] can be used to find whether or not the gesture was created with fingers in an oblique position. Finally, SP [213] provided a great direction for multi-touch recognition.

## **Modeling**

The importance in Human-Computer Interaction (HCI) of descriptive models (e.g., “Three-state model for graphical input” [21]) and predictive models (e.g., Fitts’ law) can be seen in the seminal works by English et al. [46] and Gray et al. [62]. A descriptive model is a “loose verbal analogy and methaphore” [167], which describes a phenomenon [133]. A predictive model is expressed by “closed-form mathematical equations” [167], which predict a phenomenon [133].

Input systems formalism is not recent. The pioneer work by Newman (1968) used a state diagram to represent a graphical system [153]. The seminal work by Bill Buxton in “A Three-State Model of Graphical Input” [21] demonstrated that input devices, such as mouse, pen, single-touch, and similar ones, needed only three states to be described. Around the same time as Buxton’s work, a well-rounded model for input interactions was published by Myers [147].

Multi-touch gesture detection, or detection of touch events, has been explored. In 2013, Proton and Proton++ showed the use of Regular Expressions (RegEx) to accomplish gesture detection [112]. Lao et al. [122] used state-diagrams for the detection of multi-touch events. Context-free grammar was used by Kammer et al. to describe multi-touch gestures without taking implementation into consideration [106]. Gesture Coder [130] created Finite-State Machines (FSMs) by demonstration<sup>11</sup> to later use them to detect gestures. Gesture Works and Open Exhibits by Ideum used a high-level language description, using

---

<sup>11</sup>Training methods.



XML, called Gesture Markup Language (GML) A rule-based language (e.g., CLIPS) was used to define the Midas framework [187].

Petri Nets have also been used to detect gestures. Nam et al. showed how to use Coloured Petri Nets (CPN) to achieve hand (data glove) gesture modeling and recognition [151], using HMM to recognize gesture features that are then fed to a Petri Net (PN). PNs have been shown to be applicable in event-driven systems [75], which is another reason they are interesting to use for modeling modern input devices. Spano et al. [198, 199] showed how to use Non-Autonomous PNs [35] (low-level PNs), for multi-touch interaction. Also, Hamon et al. [75] expanded on Spano's work, providing more detail to the implementation of PNs for multi-touch interactions.

## **1.7 Dissertation Structure**

The structure of the dissertation is organized in a series of chapters, including background, methodologies (two chapters), experiment design, experiment analysis, experiment discussion, and conclusion.

Chapter 2 describes the background research, concepts, and terminology pertinent to this dissertation. Chapters 3 and 4 describe the contributions from this dissertation. Note that this is divided into two chapters. Chapter 3 describes all the contributions and Chapter 4 expands on 3D Navigation System (3DNav), which is the multi-touch system implemented and evaluated with human subjects. Then, the design of the experiment used to evaluate 3DNav is explained in Chapter 5, with its data analysis in Chapter 6. A discussion about the experiment data is found in Chapter 7. Finally, the conclusion from this research is presented in Chapter 8.

The appendices contain additional information that is complementary to this dissertation. For example, Appendix B provides the actual questionnaires given to the subjects,

Appendix C contains the approval memos from the institutional review board (IRB) office, and Appendix D describes the Xbox 360 controller in detail.

## CHAPTER 2

### BACKGROUND

This chapter provides the background information for this dissertation. It covers basic understanding of the concepts of CG, HCI, 3DUI, and PN. For additional details about these topics, see [37, 92, 133, 174].

## 2.1 Computer Graphics

CG is the primary reason that has allowed PCs and mobile devices to become pervasive in today's society. Furthermore, interactive computer graphics allows for non-static 2D and 3D images to be displayed on the computer with a redraw rate higher than humans can perceive. Interactive graphics provides one of the most "natural means of communicating with a computer" [51] and a user. The primary reason is the ability to recognize 2D and 3D patterns, making graphical representation a rapid form to understand knowledge [51].

The following section covers the essential parts of CG that relate to this dissertation. In specific, how CG relates to 3D navigation. There are additional details, such as global illumination, programmable shaders, and other advanced features, that are beyond the scope of this dissertation; the reader is suggested to see [92, 131, 190]. For an introduction to various topics about real-time rendering, see [3].

### 2.1.1 Camera Space

There are various types of cameras. Two popular types of cameras are first-person and third-person. A first-person camera<sup>1</sup> is like viewing the outer space of the universe from

---

<sup>1</sup>Popularized by the game Doom and later Quake.

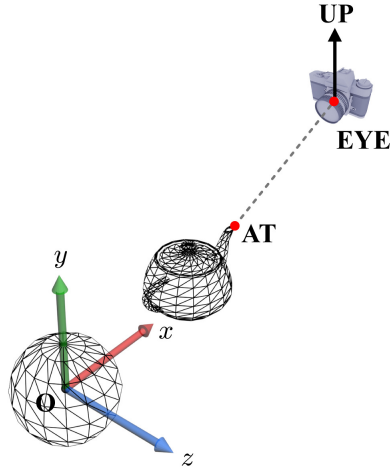


Figure 2.1: Camera Space [77]

the inside of a spaceship. This has been used in FPS games. A third-person camera<sup>2</sup> is like viewing a spaceship as it interacts with the universe. For the purpose of this dissertation, the camera plays a central role in 3D navigation. A first-person camera was used for the study.

In CG, a pipeline refers to a “sequence of data processing elements, where the output of one element is used as the input of the next one” [77]. In other words, is a sequential process that adds, modifies, and removes from the previous input until it creates the desired graphical output. The main steps in a pipeline include vertex processing, rasterization, fragment processing, and output merging. The pipeline has kept evolving over time. To read more about the current pipeline, see [131, 190].

The pipeline starts with the vertex process, which operates on “every input vertex stored in the vertex buffer” [77]. During this process, transformations such as rotations and scaling are applied. Later, the rasterization “assembles polygons from the vertices and converts each polygon to a set of fragments” [77]. Then, the fragment process works on each fragment and finds the color needed to be applied, as well as applying texture operations. A

---

<sup>2</sup>The famous Nintendo game, Mario Bros. used a third-person camera.

fragment is a pixel with its updated color. Finally, the output merging completes the process and outputs the graphical representation to the user. In general, the vertex and fragment processes are programmable<sup>3</sup> and the rasterization and output merging is fixed (hard-wired) into the software or hardware.

Before covering the space camera, it is important to have the local space and the world space in context. The local space is the coordinate system used to create a model, for example, a single mesh in 3DS Max<sup>4</sup>. The local space is also called object space or model space. The world space is the coordinate system used for the entire virtual scene. For the particular case of the study performed, the world remained static while the camera moved. However, there are many instances where local objects may need to have transformations applied that are not applied to the entire world. For additional information about transformations, see [77]. The local, world, and camera spaces are right-hand systems, as shown in Figure 2.1. The following list describes the camera components (adapted from [77]):

- **EYE** is the current position of the camera, denoted by a 3D point, in reference to the world space.
- **AT** is a 3D point, where the camera is looking towards.
- **UP** is a vector that describes where the top of the camera is pointing towards. A common usage is to have this vector set to  $[0, 1, 0]$ . This indicates that the top of the camera is pointing to the positive Y axis.

The camera has a view frustum that defines the viewable space. This also helps the rendering system to clip anything found outside the frustum (see Figure 2.2). The parameters found in this view volume are (adapted from [77]):

---

<sup>3</sup>In older OpenGL, the entire pipeline was fixed.

<sup>4</sup>Other popular 3D Modeling tools are Maya and Blender.

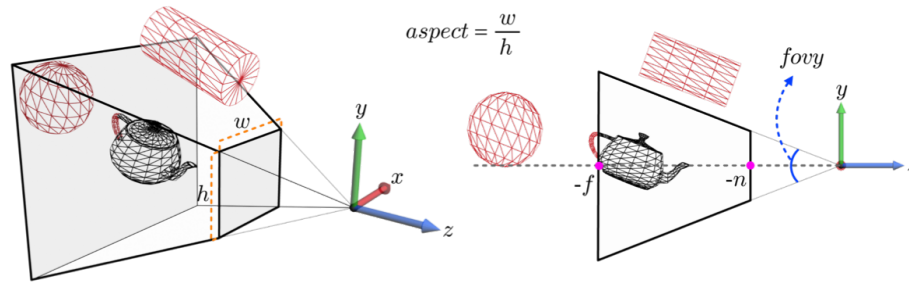


Figure 2.2: View Frustum [77]

- **fovy** defines the vertical field of view (FOV), which is the visual angle of the camera.
- **aspect** is the ratio between the height and width of the view volume.
- **n** is the near plane for the view volume.
- **f** is the far plane for the view volume.

### 2.1.2 3D Translation And Rotations

In an interactive computer graphics system, the movements of a camera are accomplished using transformations (e.g., affine transformation). There are additional transformations such as scale and reflection, among others (see [39]). For the study conducted in this dissertation, only translation and rotations are critical to understand.

Translations are the simplest to work with. Translation is the linear movement on X, Y, and Z axes. To perform a translation, simple addition and multiplication done to each individual axis is enough. For example, to move the object 30 units to the left, the translation vector will simply be  $T.x = T.x + 30$ . The origin is set to be at  $[0, 0, 0]$ .

Orientation is closely related to direction, angular displacement, and rotations. **Direction** usually denoted by a vector and indicates where the vector is pointing; however, the direction vector has no orientation. If you twist a vector along the arrow, there is no real

change. Orientation “cannot be described in absolute terms” [39]. An **orientation** describes a given rotation based on a reference frame. **Angular displacement** is the “amount of rotation” [39]. Orientation may be described as “standing upright and facing east” [39] or by adding angular displacement as “standing upright, facing north” [39] and then rotating “90° about the z-axis” [39]. There are several ways to describe orientation and angular displacement such as matrix form, Euler angles, and quaternions. For the purpose of this dissertation, only Euler angles and quaternions are covered.

### **Euler Angles**

Euler angles are defined as the angular displacements of a “sequence of three rotations about three perpendicular axes” [39]. The order of application of the rotations makes a difference in the final result. This means that if a rotation about the X axis is applied before the rotation about the Y axis, the result can be different than the one obtained if the rotation about the Y axis is applied first. The fact that this method to describe orientation uses three individual operations for each type makes it very easy to use by humans.

With Euler angles, the definition of pitch, roll, and yaw are quite intuitive. The easiest way to think about these types of rotation is to think about an airplane, as shown in Figure 2.3. **Yaw** is defined as the rotation about the Y axis (also called heading). **Pitch** is defined as the measurement of rotation about the X axis (also called elevation). **Roll** is defined as the rotation about the Z axis (also called bank). These are called the principle axes.

There are a few important consideration that must be taken into account when using Euler angles. First, rotations applied in a given sequence will yield a result that may be different if the rotations are applied in a different order. For example, a pitch of 45 degrees followed by a yaw of 135 degrees may yield a different result if the yaw is applied before the pitch. Second, the three rotations in Euler angles are not independent of each other. For example, the transformation applied for a pitch of 135° is equivalent to a yaw of 180°,

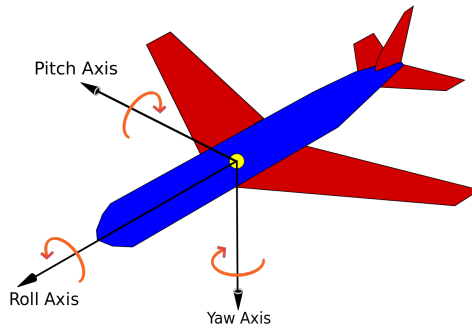


Figure 2.3: Principal Axes. Drawn by Auawise<sup>5</sup>

followed by pitch of  $45^\circ$ , and concluded by a roll of  $180^\circ$ . A common technique is to limit the roll to  $\pm 180^\circ$  and the pitch to a  $\pm 90^\circ$  [39].

An additional problem with Euler angles is known as **Gimbal lock**. This phenomenon happens when the second rotation angle is  $\pm 90^\circ$ , which causes the first and third rotations to be performed about the same axis. To correct this problem, one can set the roll to  $0^\circ$  if the pitch is  $\pm 90^\circ$  [39].

## Quaternions

Quaternion is a number system used to represent orientation, which is very popular in CG. A quaternion is represented by a scalar value ( $w$ ) and a vector ( $v$ ) with  $x$ ,  $y$ , and  $z$  components, as shown in Equation 2.1. Its popularity is due to some of its advantages. First, quaternions using the slerp function (see [39]) provide smooth interpolation. Second, quaternions use only four numbers. This makes it fairly easy to convert them from and to matrix form. It is very easy to find the inverse of a given angular displacement. Finally, it provides fast concatenation using the quaternion cross product. Nevertheless, they have some disadvantages as well. While four numbers is less than nine numbers in a matrix, quaternions are larger than Euler angle representation. Also, if the values provided to

---

<sup>5</sup>[http://commons.wikimedia.org/wiki/File:Yaw\\_Axis\\_Corrected.svg](http://commons.wikimedia.org/wiki/File:Yaw_Axis_Corrected.svg).



the quaternion are invalid, the quaternion may become invalid. This can be overcome by normalizing the quaternion. Finally, quaternions are not as easy to visualize as Euler angles. Regardless of the disadvantages mentioned, quaternions are the preferred method in graphics and game engines (e.g., OGRE [100]).

$$[w \ (x \ y \ z)] = [\cos(\theta/2) \ (\sin(\theta/2)n_x \ \sin(\theta/2)n_y \ \sin(\theta/2)n_z)] \quad (2.1)$$

### 2.1.3 Geometric Modeling

This dissertation makes use of various polygon meshes<sup>6</sup>, which represent a 3D model to be used in a rendering application [218]. The polygon representation is not the only way to model objects. Alternative methods include bi-cubic parametric patches, constructive solid geometry, and implicit surface representation, among others. For this dissertation, the polygon representation is the only one used. In particular, Object-Oriented Graphics Rendering Engine (OGRE) uses its own mesh binary format (and an available Extensible Markup Language (XML) version) to load 3D models. It is important to note that graphics processing units (GPUs) have been highly optimized to use polygon representations [77]. In addition, polygon models can use hierarchical and spatial representation to define a group of objects working as a unit (e.g., person with legs and arms).

Polygon models are represented using vertices, faces, and edges. A very common approach is to use the half-edge data structure [13]. For more information about data structures for polygon meshes, see [13, Chapter 2]. A mesh also contains vertex normal and texture coordinates, among other properties. Sometimes, the mesh may also contain data related to the animation for the object. A very typical technique is called skeletal animation,

---

<sup>6</sup>Commonly referred to as mesh.

where specific parts of the mesh are defined for movement. The animation topic is beyond the scope of this dissertation. For more information about animation, see [162, 168].

The topic of geometric modeling is quite large. The reader is suggested to look at [13, 141, 217].

#### **2.1.4 Scene Managers**

Scene managers are useful to handle large virtual worlds. While there are different ways to approach the design of a scene manager, the “common practice is to build scene as ontological hierarchies with spatial-coherency priority” [207]. Different types of scene managers are available to graphics engines, such as binary space partitioning (BSP), quad-tree, and octree, among others. These types of hierarchies are called scene graphs. For example, Theoharis et al. defines a scene graph as the set of nodes that “represents aggregations of geometric elements (2D or 3D), transformations, conditional selections, other renderable entities, (e.g., sound),” operations and additional scene graphs [207]. A more compact definition is that a scene graph is a spatial representation of rendering objects in a virtual world where operations can be applied to parents or children of this graph. Mukundan defines a scene graph as a data structure that “represents hierarchical relationships between transformations applied to a set of objects in three-dimensional scene” [144]. The general type of graph used in a scene manager is directed, non-cyclic graphs or trees of nodes [207]. The definition of a scene graph may vary depending on the actual type. The list below details some of the items that a scene graph includes in a node [45]:

- Rendering elements:
  - Static meshes.
  - Moving meshes.
  - Skeletal meshes.

- Materials.
- Collision elements:
  - Bounding volumes.
  - Trigger actions.
  - Bullets.
- Other elements:
  - Artificial Intelligence (AI) path.
  - Game data.
  - Sounds.

Three very useful functionalities of a scene manager are instancing, operations, and culling. Instancing allows the use of existing objects (e.g., meshes) to create nodes. This means that instead of duplicating all the geometric information, a node can be created that points to the primary object. For example, this could mean that a car with certain material and geometric representation could be referenced 100 times, without having to duplicate this information. Later, using different transformation operations or changing basic details, some of those cars may look different and be placed in different locations. This is the importance of being able to perform different types of operations in the node. The most common operation is to perform geometric transformations to the object. The usefulness of the scene node is that the operation can be local to the node, without affecting the parents. Finally, culling allows the scene manager to make a decision about which objects to render. While this is done by the GPU as well, the scene manager can be highly optimized to minimize work. An extended study about culling is found in [51, Chapter 15].

This dissertation uses OGRE [100] and its scene manager. The generic scene graph was used for this dissertation. This scene graph (**ST\_GENERIC**) is a basic representation

of a hierarchical tree with parent/child relationships. The most common scene graph used for large virtual worlds is the spatial (using partitions) hierarchical representation, called octree. For a detailed study on spatial representations, see [184]. For additional information, the reader is suggested to consult [41, 42, 51]. For a specific formal study about BSP, quadtrees, and octrees, see [10, 121].

### **2.1.5 Collision Detection**

Collision detection deals with the problem of how to find two objects occupying the same space at a given time [212]. For 3D real-time environments (e.g., games, simulators), this topic defines how to detect the intersection (collision), and what to do when it happens [47]. The latter part, which deals with the action following the intersection, is up to the designer. A common approach is to use physics engines (e.g., Bullet) to perform some type of reaction to the collision (e.g., bounce). That is outside the scope of this dissertation, but the reader is suggested to see [43].

In this dissertation, the type of collision detection performed was a simple collision using the built-in functions provided by OGRE. This collision type is referred to as a bounding volume, where you can set spheres, boxes, and other 3D shapes that can help with the collision algorithms [47].

## **2.2 Human-Computer Interaction**

HCI is the field that studies the exchange of information between computers and their users. The field emerged in the decade of the 1980s, with influences from many other fields, such as Psychology, Computer Science, Human Factors, and others. An alternate definition of HCI is the communication between computer systems and computer users. How the communication enables the use of the technology is a central part of the study of HCI.

A field closely related to HCI is Human Factors, which is concerned with the capabilities, limitations, and performance by people when using any type of systems<sup>7</sup>. Also, from the perspective of the system design, Human Factors studies the efficiency, safety, comfortability, and enjoyability when people use a given system. This seems quite close to HCI if you think of systems as computer systems [133]. The HCI field also draws knowledge from Psychology. Some examples are Cognitive Psychology, Experimental Psychology, and others. This is very useful, as it helps us to understand users, and tests how they interact with different types of computer systems. HCI is a very broad field. In HCI, you can find Virtual Reality (VR), 3DUI, Affective Computing (which draws as much as from Psychology as it does from AI), and others.

Major events in the HCI community [133] started with the vision of Vannevar Bush with “As We May Think,” the development of the Sketchpad by Ivan Sutherland, the development of the computer mouse by Douglas Engelbart, and the launch of the Xerox Star system. Then, the first interest group formed (SIGCHI<sup>8</sup>) in 1982, followed by the release of the seminal book, *The Psychology of Human-Computer Interaction*, and the release of the Apple Macintosh, starting a new era for computer systems.

### **2.2.1 Usability**

Usability is defined as the user acceptability of a system, where the system satisfies the user’s needs [154]. Furthermore, usability is an engineering process that is used to understand the effects of a given system with the user, in other words, how the system communicates with the user and vice-versa. Usefulness “is the issue of whether the system can be used to achieve some desired goal” [154]. This breaks down into utility and usability.

---

<sup>7</sup>It doesn’t have to be a computer system.

<sup>8</sup>See <http://www.sigchi.org>.

Utility addresses the question of the functionality of the system, which means if the system can accomplish what it was intended for [154]. Usability in this case refers to “how well users can use that functionality” [154]. Finally, it is important to understand that usability applies to all aspects of a given system that a user might interact with [154]. The following list briefly describes the five traditional usability attributes (adapted from [154]):

- **Learnability:** The system must have a very small learning curve, where the user can quickly become adapted to the system to perform the work required.
- **Efficiency:** Once the user has learned the system, the productivity level achieved must be high in order to demonstrate a high efficiency.
- **Memorability:** The system is considered to have a high memorability if when the user returns after some time of usage, the interaction is the same or better than it was the first time.
- **Errors:** The system is considered to have a low error rate if the user makes few errors during the interaction. Furthermore, if users make an error, they can recover quickly and maintain productivity.
- **Satisfaction:** The interaction with the user must be pleasant.

### 2.2.2 The Light Pen and the Computer Mouse

There are two major events that are critical moments in HCI for people who study input user interfaces: the Sketchpad by Ivan Sutherland<sup>9</sup> and the invention of the mouse by Douglas Engelbart. This all happened before the explosion of the field in the 1980s.

Having more than one device prompted English, Engelbart, and Berman [46] to ask: Which device is better to control a cursor on a screen? This was the first user study dealing

---

<sup>9</sup>He also introduced the VR HMD.

with input devices for computing systems that may have marked a “before and after” in glshci [46].

In this study (“Display-Selection Techniques for Text Manipulation”), the subjects were presented with a computer mouse, a light pen, a joystick with two modes (absolute and rate), Grafacon<sup>10</sup>, and a user’s knee lever (also referred to as the knee controller). The study used repeated-measures, meaning that all devices were tested for each subject. The dependent variables that were measured were: **time** to acquire target and the **error rate** when trying to acquire the target. The combination of both measurements led researchers to the conclusion that the computer mouse was a better device based on the study. This was great news, because a knee lever doesn’t seem quite exciting to use. Figure 2.4 shows how the knee controller was faster than the rest of the devices. However, looking at Figure 2.5, it is clear that the mouse has the lowest error rate compared to the other devices tested. Another interesting part of the study is that while the light pen was faster than the computer mouse, it was known to cause discomfort<sup>11</sup> after prolonged use<sup>12</sup>.

### 2.2.3 Graphical User Interfaces and WIMP

GUI is a type of interface that allows the user to interact with windows or icons, as opposed to a text interface. The GUI was first seen in the Xerox Star system, developed at PARC [87]. The GUI has allowed systems like Microsoft Windows and Apple Macintosh to become pervasive for day-to-day use in desktop computers and mobile devices.

---

<sup>10</sup>Graphical Input device for curve tracing, manufactured by Data Equipment Company.

<sup>11</sup>Which was probably known from the times of the sketchpad [202].

<sup>12</sup>Using a pen while resting a hand on the surface may be more comfortable [93].

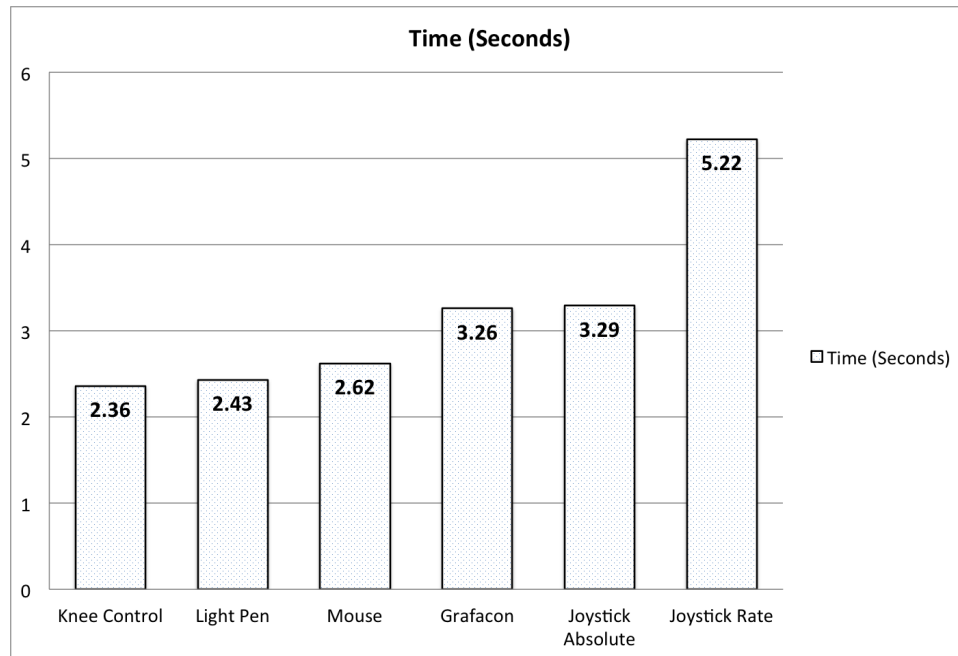


Figure 2.4: Mouse Study: Time [46]

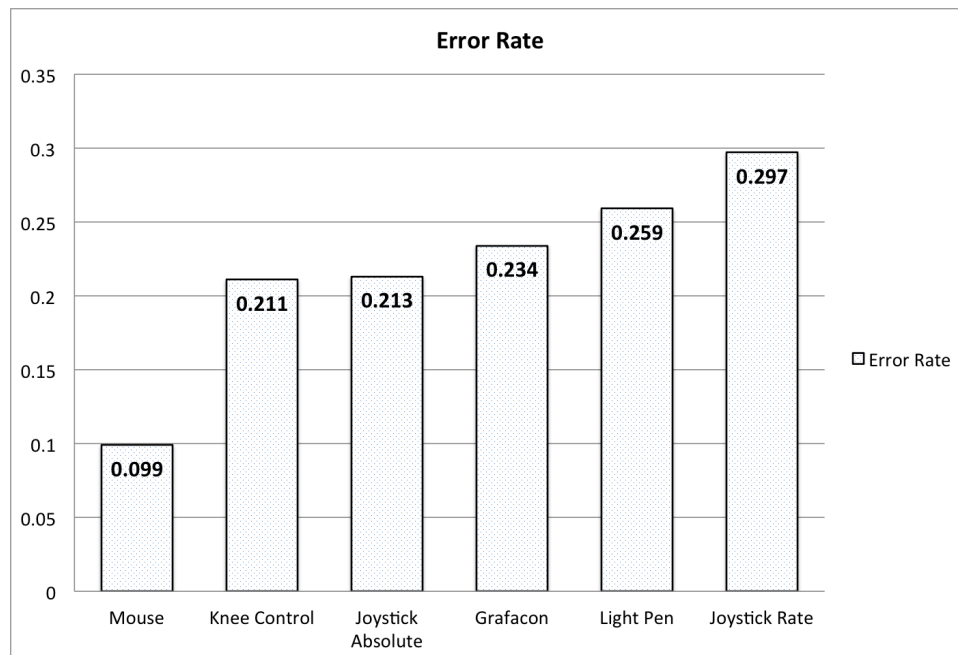


Figure 2.5: Mouse Study: Error Rate [46]



The WIMP paradigm<sup>13</sup> emerged with the availability of Apple and IBM desktop computers. This paradigm refers to the interaction of users with the GUI, which includes WIMP [178]. The **Windows**, which can be scrolled, overlapped, moved, stretched, opened, closed, minimized, and maximized, among other operations, were designed to be the core of the visual feedback and interaction with the user. The **Menus** allow the user to select different options that augmented the use of the Window. The menu can be activated by clicking on it. Once selected, the menu expands down (or in a different direction), which allows the user to select an option. The **Icons** allow a representation of applications, objects, commands, or tools that, if clicked, become active. Icons are commonly found in a desktop area of the display or in a menu. Finally, the central part of WIMP is the **Pointing**. This allows the user to interact with any of the interface elements already mentioned, using a mouse with a click button [178]. The paradigm along, with the GUI interface, has evolved to allow many new types of interactions. However, the basic principle remains the same.

## 2.2.4 Input Technologies

Input devices are the primary tool for providing information to a computer system, which is essential for the communication between a user and a computer. In general, input devices can sense different physical properties, such as “people, places, or things” [88]. When working with input devices, feedback is necessary. For example, using a brush to paint in mid-air without proper feedback will be as useless as using the pen without paper, or using a car alarm system without some feedback from the device or the car. This is why input devices are tied to output feedback from a computer system.

---

<sup>13</sup>Referred to as interface by Dix et al. [37].

While each device has unique features and it is important to know the limitations and advantages from a given device, there are some common properties that may apply to most devices. The following is a list of input device properties (adapted from [88]):

- **Property Sensed:** Devices can sense linear position and motion. Some devices will measure force and others change in angle. The property that is used will determine the transfer function that maps the input to the output. Position-sensing devices are *absolute input devices*, such as multi-touch. *Motion-sensing devices* can be relative input devices, with the example of the mouse as one of them. There is room for ambiguity depending on the actual design of the transfer function. For example, a multi-touch device can be used as a relative input device if desired, even though it is a direct-input system.
- **Number of Dimensions:** The number of dimensions sensed by the device can determine some of its use. For example, the mouse senses two dimensions of motion for X and Y coordinates, while a vision-input system may sense three dimensions of motion for X, Y, and Z axes. While the mouse does sense two dimensions, the transfer function may be mapped to a higher degree of dimensions with some additional input, to provide a richer set of interaction (e.g., 3D navigation). Some other devices have different types of 3D sensors (e.g., gyroscope, accelerometer, etc.). Understanding the dimensions of the device can provide a designer with the correct mapping for the transfer function.
- **Indirect versus Direct Devices:** *Indirect devices* provide a relative measurement to be used with the computer system. For example, the motion of the mouse is relative to the position of the cursor and the movement of the user. In other words, while the cursor may traverse the complete screen from left to right, the movements by the user with the mouse may be confined to a smaller space than the screen. A mouse may even be picked up and moved to a different location without causing any

movement on the display, using the transfer function to map the movements. *Direct devices* provide a different interaction for users. In general, they lack buttons for state transitions and allow the user to work directly on the display. One example is the use of a digital pen with a tablet. The user can paint directly on the screen, having a 1:1 relationship with the surface display. There are typical problems with direct devices, such as occlusion, i.e., the user may occlude a part of the display with their hand.

- **Device Acquisition Time:** “The average time to move one’s hand to a device is known as *acquisition time*” [88]. *Homing time* is defined as the time that a user takes to go from one device to another. One example is the time that it takes to return from a multi-touch desktop to a keyboard.
- **Gain:** The control-to-display (C:D) gain or ratio is defined as the distance an input device moved (C) divided by the actual distance moved on the display (D). This is useful to map indirect devices, such as the computer mouse.
- **Sampling Rate:** Sampling rate is given by how many samples are processed each second. For example, if the WiiMote Motion Plus has a sampling rate of 100 hertz (Hz), this means that for every one second, there are 100 digital samples from the device. In some devices, the sampling rate will be fixed, and in others, it may vary. The difference may be due to an actual delay on the computer system and not to the input device. Understanding the sampling rate by a designer can provide additional features for the user’s interactions. For additional information about sampling rate and Digital Signal Processing (DSP), see [94, 235].
- **Performance Metrics:** Performance metrics are useful to study in order to understand the interaction of a input device, its user and the computer system. One example is seen in the classic study of the mouse versus other devices [46], where the error-rate was a significant factor of why the mouse was the device found to be

most effective. Other metrics include pointing speed and accuracy, learning time, and selection time, among others.

- **Additional Metrics:** There are several other metrics, including some that may be unique to an input device. One example is the pressure size in multi-touch displays<sup>14</sup>. Understanding all the possible metrics are important. The recommendation by Bowman et al. is to know the capabilities and limitations of each device [17].

### **Isometric and Isotonic Devices**

It is important to make a difference between isometric and isotonic devices because this study used a GamePad in the experiment, which contains two thumb-sticks (miniature joystick). Isometric devices are pressure (force) sensing. This means that the user must apply force to move the device (e.g., joystick). Given this force, the movement could be precise (e.g., isometric joystick). Isotonic devices are those with “varying resistance” [234]. In other words, as the “device’s resistive force increases with displacement” [234], the device becomes sticky, “elastic, or spring loaded” [234]. For example isotonic joysticks “sense the angle of deflection” [88], with most isotonic joysticks moving from their center position [88]. There are also designs of joysticks that are hybrid [88].

#### **2.2.5 Input Device States**

Defining the states for modern input devices has proven not to be an easy task. In the seminal work by Bill Buxton [21], he defined that almost any input device to be used with a WIMP could be expressed in three states. His model was able to model devices, such as PC mice, digital pens, or single-touch displays. However, even his model could fail in

---

<sup>14</sup>If this is available to the device.

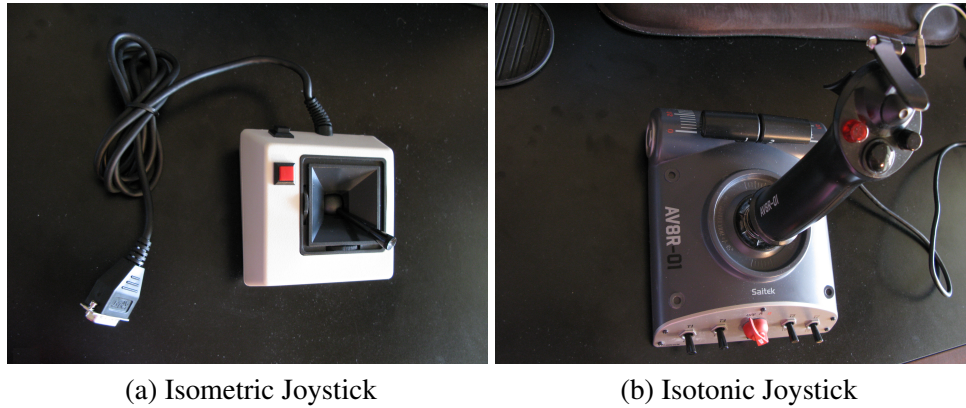


Figure 2.6: Images

some instances. One example is in the work by Hinckley et al., which demonstrated that a digital pen required a five-state model [89].

With the explosion of many new input devices, such as multi-touch, Microsoft Kinect, and Leap-Motion, among others, the three-state model is no longer valid. Multiple attempts to define a new model made have been made, as was described in 1.6. Some of the attempts for modeling multi-touch have included Regular Expressions (Regex), FSMs, and high-level Petri Nets (HLPNs).

### 2.2.6 Bi-Manual Interaction

Humans have used both hands when needed to perform some specific task [133]. While typing is possible with one hand, both hands seems to be the most effective way to type for most users, when using a PC keyboard. For example, when using a hammer, a user may need to hold the nail with the non-dominant hand, while hitting the nail with a hammer, in the dominant hand.

Psychology researchers studied the bi-manual behavior years before HCI researchers. For example, in 1979, Kelso et al. studied the coordination of two-handed movements [108]. In their work, they found that while the kinematic movements of the hands are dif-

ferent, the movements of both hands are synchronized for certain tasks. In their own words: “The brain produces simultaneity of action as the optimal solution for the two-handed task by organizing functional groupings of muscles,” which act as a single-unit [108]. Another example of psychology that deals with two hands is the work by Wing [224]. He studied the timing and coordination of repetitive tasks using both hands. While he used only four subjects, the study led to the conclusion that there may be a difference between synchronous movements versus asynchronous movements [224]. The users did report that synchronous movements in bi-manual tasks were easier [224]. In general, studies have shown that most tasks are asymmetric [133]. Studies have also shown that while hands work together, they have different roles to “perform different type of tasks” [133].

Later, in 1987, Guiard proposed a model for bi-manual action [67]. His model makes two assumptions: First, the hand represents a motor (people having two motors), which serve to create a motion. Second, the motors cooperate with each other, forming a kinematic chain; one hand articulates the movement of the other hand. Guiard’s model describes the “inter-manual division of labor” [67].

In 1986, Buxton and Myers published a study for bi-manual input [22]. They found that users benefited by using both-hands, because of the efficiency of hand motion in bi-manual actions [22]. Later, in 1993, Kabbash, MacKenzie, and Buxton [102] studied the use for preferred and non-preferred hands. They found that the non-preferred hand performs better in tasks that do not require action (e.g., scrolling). The preferred hand was found to be better for fine movements. This meant that each hand has “its own strength and weakness” [102]. Figure 2.7 shows an example of a bi-manual task and the difference between both hands. The following describes the roles and actions for each hand [67, 102, 133] (adapted from table in [133, Chapter 7]):

- Non-preferred hand:
  - Leads the preferred hand.

- Provides a spatial frame of reference for the preferred hand.
- Achieves non-fine movements.
- Preferred hand:
  - Follows the other hand.
  - Utilizes a frame of reference set by the non-preferred hand.
  - Achieves fine movements.

Later, in 1994, Kabbash, Buxton, and Selen published “Two-Handed Input in a Compound Task” [101]. This is a significant contribution, as it is the first publication to adapt the bi-manual model by Guiard [67]. The experiment had four modes when using the computer: one uni-manual, one bi-manual, with each hand having independent tasks, and two bi-manual, requiring asymmetric dependency. The study found that one of two bi-manual asymmetric modes performed better (as expected by them) than the other methods. This method is called the **Toolglass** technique, previously published by Bier et al. [11], which provided the use of additional widgets for the user’s interactions. The reader is suggested to look at [126] to read more about the benefits of two-handed input.

The bi-manual model has been used for multi-touch devices, such as [9, 143, 229]. Benko et al. showed that the **Dual Finger Stretch** technique was found to provide a simple and powerful interaction [9]. Moscovich and Hughes found that indirect mapping of multi-touch input was compatible (one hand versus two hands) in most cases [143]. They also found that “two hands perform better than one at tasks that require separate control of two points” [143]. Bi-manual modeling plays an important role in multi-touch interaction and HCI.

Year	Name	Description
1960	Single Touch	Single Touch (not pressure-sensitive), developed at IBM, the University of Illinois, and Ottawa, Canada. The development occurred in the second part of the 1960s.
1972	PLATO IV Touch Screen Terminal	The invention of flat-panel plasma display, which included a 16x16 single touch (not pressure-sensitive) touch screen. This machine included real-time random-access audio playback and many other features. The touch panel worked by using beams of infrared light in horizontal and vertical directions. When the infrared light was interrupted at a point, it triggered the touch [40].
1978	Vector Touch	One-Point Touch Input of Vector Information for Computer Displays [86]. It included the ability to detect 2D position of touch, 3D force and torque.
1981	Tactile Array Sensor for Robots	Multi-touch sensor for the use of robotics to enable different attributes, such as shape and orientation. This consisted of an array of 8x8 sensors in a 4-inch squared pad [226].
1982	Flexible Machine Interface	The first multi-touch system developed by Nimish Mehta at the University of Toronto. The system consisted of a frosted-glass panel, producing black spots in the back of the panel. This allowed simple image processing, marking white spots as non-touch and black spots as touch.
1983	Soft Machines	Soft Machines: A Philosophy of User-Computer Interface Design by Nakatani et al. provides a full discussion about the properties of touch screen. The attributes discussed outline certain contexts and applications where they can be used [150].
1983	Video Place - Video Desk	A multi-touch vision-based system [118], allowing the tracking of hands and multiple fingers. The use of many hand gestures currently ubiquitous today, was introduced by Video Place. These included pinch, scale and translation.
1984	Multi-Touch Screen	Multi-touch screen using capacitive array of touch sensors overlaid on a CRT. It had excellent response. This machine was developed by Bob Boie and presented at SIGCHI in 1985.

Table 2.1: Partial History of Multi-Touch Until 1984



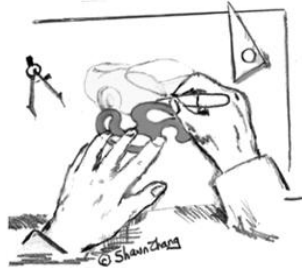


Figure 2.7: Bi-Manual Interaction [133]

## 2.3 Multi-Touch Displays

Multi-touch displays come in many flavors. The most representative type of multi-touch has been the smart phone, with the introduction of the Apple iPhone. Other types seen in daily use are tablets, such as the Apple iPad. The introduction of Microsoft Windows 7 and 8 opened a possibility to use multi-touch displays with a desktop PC. For public spaces, vertical displays can be a solution for multi-user interaction. Finally, another modality is to have tabletop displays (horizontal) that will act just as a desk (e.g., Microsoft Surface). Multi-touch has even been extended to work with stereo vision [68].

To take into perspective the history of multi-touch, Tables 2.1, 2.2, 2.3 provide a look at (in a big part from the Perspective of Bill Buxton<sup>15</sup>) the milestones in multi-touch technologies. In particular, the **PLATO IV** Touch System and the philosophy of “Soft Machines” are shown in Table 2.1.

---

<sup>15</sup>See <http://www.billbuxton.com/multitouchOverview.html>.

Year	Name	Description
1985	Multi-Touch Tablet	Lee, Buxton and Smith developed a multi-touch tablet with the capability of sensing not only location but degree of touch for each finger. The technology used was capacitive [125]. It is important to note that the development of this tablet started in 1984, when the Apple Macintosh was released.
1985	Sensor Frame	Paul McAvinney at Carnegie-Mellon University. This multi-touch tablet, which read three fingers with good accuracy (errors could occur if there were shadows), had optical sensors in the corners of the frame. The sensors allowed the system to detect the fingers when pressed. In a later version, the system could detect the angle of the finger when coming in contact with the display.
1986	Bi-Manual Input	Buxton and Meyers studied the effect of bi-manual multi-touch. One task allowed positioning and scaling, while the hand performed the selection and navigation task. The results reflected that continuous bi-manual control provided a significant improvement in performance and learning [22]. The control was very easy to use.
1991	Digital Desk Calculator	Wellner developed a front projection tablet top system that sensed hands and fingers, as well as a set of objects. He demonstrated the use of two-finger scaling and translation [220].
1992	Simon	IBM and Bell South introduced the first smart phone that operated with single-touch.
1992	Wacom Tablet	Wacom released digitizing tablets, which included multi-device and multi-point sensing. The stylus provided position and tip pressure, as well as the position of the mouse-like device, enabling commercial bi-manual support [126].
1995	Graspable Computing	The foundation of what has become graspable or tangible user interfaces to use with multi-touch devices. [50]. The work by Fitzmaurice included Active Desk as well.

Table 2.2: Partial History of Multi-Touch from 1985 to 1995

Year	Name	Description
1997	The metaDESK	metaDesk was developed by Ullmer et al. [209], which demonstrated the use of Tangible User Interfaces(TUI).This addressed the problem that “dynamic assignment of interaction bricks to virtual objects did not allow sufficient interaction capabilities” [145].
2001	Diamond Touch	Diamond Touch addressed the multi-user problem with its multi-touch system [36].
2005	FTIR	Han introduced FTIR [76].
2007	Apple iPhone	Apple introduced the iPhone, a smart phone with multi-touch capabilities.
2007	Microsoft Surface Computing	Microsoft released an interactive tabletop surface for multiple fingers, hands, and tangible objects. This led to the later introduction in 2011 of Surface 2.0.
2011	Surface Computing 2.0	Second generation of the Microsoft Surface.
2013	Tangible for Capacitive	Voelker et al. developed PUCs. A tangible device for capacitive multi-touch displays [214].
2014	Houdini	Hüllsmann and Maicher demonstrated the use of LLP [93].

Table 2.3: Partial History of Multi-Touch from 1997 to 2014.

### 2.3.1 Projective Capacitive Technology

Projective Capacitive Technology (PCT) has become pervasive in the day-to-day use of consumers with the introduction of the Apple iPhone in 2007, and all the smart phones and tablets thereafter. PCT detects the touch “by measuring the capacitance at each addressable electrode” [1]. In other words, if a finger or a conductive stylus gets closer the electrode, it disturbs the electromagnetic field. When this occurs, the change in capacitance allows the touch to be detected with a specific location, with coordinates X and Y. PCT has two main forms of sensing the touch: Self-capacitance and mutual-capacitance, shown in Figures 2.8a and 2.8b, respectively.

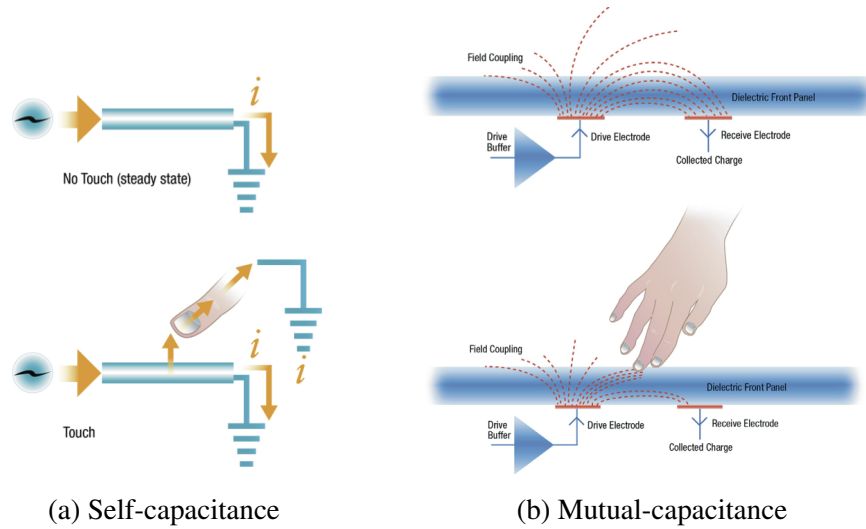


Figure 2.8: Multi-touch Capacitance Technology

†: ©2014 3M. All rights reserved.  
 3M, the 3M logo, and  
 other 3M marks are owned by 3M.

### Self-Capacitance

In the case of self-capacitance, the measurement of the electrode is with the ground. One option is to use a multi-pad with addressable electrodes, with a connection for each of them, as shown in Figure 2.9b. While this allows a multi-touch approach, screens greater than 3.5 inches become challenging because of the individual connection between the electrode and the controller. A second option is to use the row and column approach, as shown in Figure 2.10a. However, this approach has “ghost” points, as shown in Figure 2.10b, because the electronics are not able to measure each individual intersection (the electronics can only measure each electrode). This limits the approach to a single or dual touch screen because it can provide false positive points, known as “ghost” points. When using the row and column approach, the system can determine which is the closest location. Then, using interpolation, the system can determine the location of a touch.

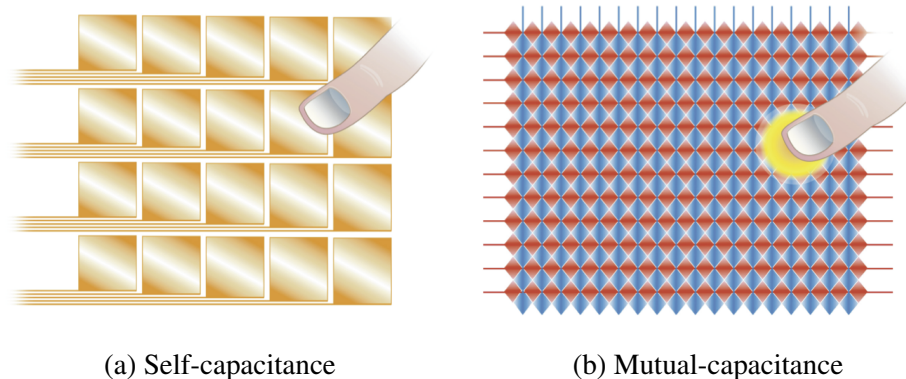


Figure 2.9: Multi-touch Capacitance Technology

†: ©2014 3M. All rights reserved.  
 3M, the 3M logo, and  
 other 3M marks are owned by 3M.

### Mutual Capacitance

Mutual capacitance exists when two objects hold charges. Projected capacitance displays create mutual capacitance between columns and rows, where each intersects the other object, as shown in Figure 2.9b. The system is able to measure multiple touches simultaneously during each screen scan. When the finger touches down near an intersection, the mutual capacitance is reduced, causing the threshold to indicate that a touch has occurred.

### 2.3.2 Optical Touch Surfaces

Optical multi-touch systems provide additional features not available in regular multi-touch systems. One example is the ability to use tangible objects of any type. It is important to note that there have been advances in tangible technology, for capacitive displays [214]. Optical approaches use image processing to determine the location and type of interaction with the surfaces.

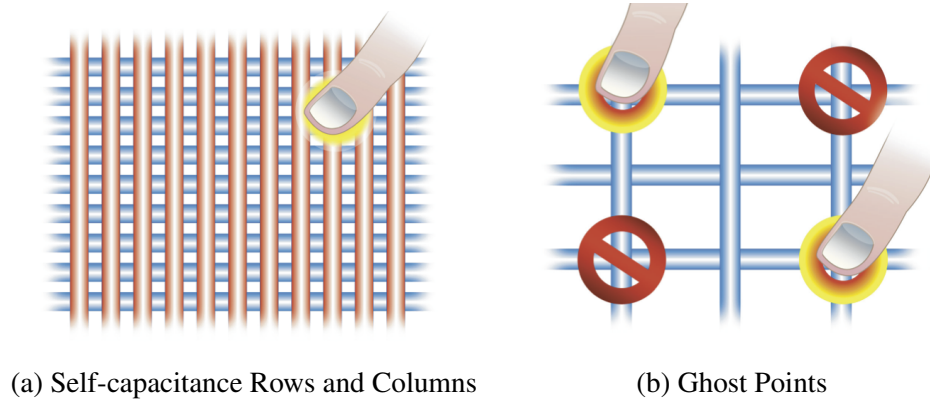


Figure 2.10: Multi-touch Capacitance Technology

†: ©2014 3M. All rights reserved.  
3M, the 3M logo, and  
other 3M marks are owned by 3M.

### Frustrated Total Internal Reflection

The Frustrated Total Internal Reflection (FTIR) approach [76] is based on the optical total internal reflection within an interactive surface. The electromagnetic waves are transmitted into the transparent surface given the following two conditions:

1. If the refractive index of the inner material is higher than the outer material.
2. If “the angle of incidence at the boundary of the surface is sufficiently small.” [145].

A common FTIR configuration uses a transparent acrylic pane. This pane injects infrared light using strips of LEDs around its edges. When the user touches down, the light escapes and therefore, it reflects the surface display, to be captured by a camera set perpendicular to the panel. In other words, the infrared light is “brought into the surface from the side where it is trapped” [93] until a user presses down into the surface. Also, since the acrylic is transparent, the projector can be in the back of the panel. A computer vision algorithm is applied to obtain location and other features of the touch [145].

## **Diffused Illumination**

The Diffuse Illumination (DI) approach produces infrared light below its surface. DI uses a projector and an infrared-sensitive camera on the back of the surface. The infrared lighting is also placed behind the projection surface (opposite in this case to FTIR) “to be brightly lit in the infrared” [145]. Given this configuration, DI technology allows for robust tracking of fingers and physical objects (tangibles). The advantage of physical objects, which use fiducial markers or size of their shape, gives a clear edge to DI technology. This approach also has the potential for hovering interaction.

## **Laser Light Plane**

The Laser Light Plane (LLP) is another approach of an optical multi-touch system (see a variation to this approach, called LLP+ [163]). The LLP dates back to 1972 by Johnson [52]. One of the major advantages that it has over DI and FTIR is that it is the most inexpensive system to build, as seen in [93], while remaining very effective.

The LLP system directs the infrared light above the surface. This gets “scattered at every touch point” [93]. The major advantage to infrared light is that it gets scattered above the display during the user’s interaction. This enables fast and reliable tracking of fingers and tangibles, regardless of how fast the movements are from the user. This is because the image is rich in contrast.

The LLP can use acrylic surfaces as well, as stable glass panes, which are less expensive than acrylic panes. Also, by using lasers, the “illumination becomes independent from the tabletop size” [93].

The approach described in this section makes reference to the system used in Houdini [93]. There are alternative approaches to LLP, which can be found in *Exploring Multi-Touch Interaction* [103].

## 2.4 3D User Interfaces

A UI is the medium of communication between a user and a computer system. The UI receives the input (actions) from the users, it delegates the computation to a designated process, and then it represents the computer's output into a representation that the user can understand [17]. A 3DUI is the medium of communication between a user performing 3D interaction with a computer. These are tasks “performed directly in a 3D spatial context” [17]. The fact that someone is working in a 3D environment does not mean that they are doing 3D interaction. For example, if the user is navigating a 3D landscape of Miami by using commands or menu-driven actions, they do not constitute 3D interaction. Conversely, 3D interaction does not have to involve 3D input devices. For example, by using a 2D non-stereo display with multi-touch capability, the user performs a 2D gesture to move along the Z axis. This means that the 2D action has been mapped (or translated) to a 3D action. This example is the case of the study conducted in this dissertation. There are several reasons why 3DUIs are important to study. Bowman et al. provided five reasons in their seminal book [17]:

1. Real-world tasks are performed in 3D.
2. 3DUI are becoming mature.
3. 3D interaction is difficult to solve, hence, it provides a challenge to many researchers.  
This also means that there are many problems that need to be solved.
4. 3DUIs require better usability.
5. 3DUIs still have many open questions, leaving space to study and to innovate as much in academia as in industry.



## 2.5 3D Output Interfaces

Required components of a 3DUI are the hardware and software that allows the graphical representation of the output. The hardware devices, which are often referred to as display devices, surface displays, or just plain displays, are the focus of this section. In addition to displays, it is also important to describe the perception of the output from the point of view of the user.

### 2.5.1 Visual Displays Characteristics

There are a few characteristics that visual displays have in common. Some of the characteristics are field of regard (FOR), FOV, refresh rate (in Hertz), and spatial resolution.

**Field of regard** is the measurement of the physical space surrounding the user, measured in visual angle (in degrees). In other words, it contains all the available points from the user's perspective. **Field of view** is the measurement of the maximum visual angle observed by the user. The FOV must be less than 200 degrees, which is approximately the maximum for the human visual system [17]. **Refresh rate** is the speed of the next rendering cycle. Simply, it is how fast the screen is redrawn. In other words, each rendering cycle produces new output. **Spatial resolution** is a measurement of quality given by the pixel size. The unit for this measurement is dots per inch (DPI). Spatial Resolution is also affected by the distance between the display and the user. There is a proportional connection between pixels and resolution. As pixels increase, the resolution also increases, and vice versa. Two different sizes of monitors with the same number of pixels do not have the same resolution. In other words, the number of pixels is not equivalent to the resolution [17].

Some other important aspects are the different type of screen geometries (e.g., spheres) and how they are drawn. **Light transfer** is a characteristic that it is important for the design of user interfaces. How the transfer of light happens is determined by the type of

projection (e.g., front, rear). Finally, how comfortable the display device is in respect to the user (**ergonomics**) is very important to consider when designing 3DUIs.

## 2.5.2 Understanding Depth

When using a 3DUI, depth plays an important role for human perception [17, 73]. In a regular display, the drawing canvas is still 2D. It is easy to tell that the moon is far away when looking at it. It appears small and far. However, when looking at two far away objects in space, it is harder to tell the difference in distance from one's point of view. At near distances, multiple cues allow users to easily process depth [17, 73]. There are four visual depth cue categories:

- Monocular, static cues.
- Oculomotor cues.
- Binocular disparity and stereopsis.
- Motion parallax.

### Monocular Cues

Monocular cues are present in static images viewed with only one eye. This type of cues is also called **pictorial cues**. These techniques have been employed by artists for a long time (they were well known before computers) [92]. The following techniques are used to convey depth (list adapted from [17, 74]):

1. **Relative Size:** The user can be influenced by apparent sizes. One example is if you have a set of objects in decreasing size order, this will give an apparent distance effect.

2. **Interposition or Occlusion:** Conveys a sense of depth by opaquing and occluding pars that are farther away.
3. **Height Relative to the Horizon:** For objects above the horizon, the higher the object is located, the closer it appears. For objects below the horizon, the lower the object is drawn, the closer it seems.
4. **Texture Gradients:** It is a technique that includes density, perspective and distortion. As the density of the surfaces increase, so it does the depth understanding for the user, making the denser objects appear farther away.
5. **Linear Perspective:** As parallel (or equally spaced) lines are seen farther away, it appears that the two lines are converging. It is important to note that lines do not have to be straight but equally spaced. For example, a long train track.
6. **Shadows and Shading:** The user can determine depth information based on lighting. Light will hit the nearest parts of an object, allowing someone to deduce this information. Any object that has illumination will also produce shadows on the closer surface.
7. **Aerial Perspective:** This effect (also called atmospheric attenuation) gives closer objects the appearance of being brighter, sharper and more saturated in color than objects far away.

## **Binocular Cues**

The world is experienced by humans using both eyes but vision is a single unified perspective. By viewing a scene first with one eye only, and then with the other eye only, a significant difference can be perceived. This difference between those two images is called **binocular disparity**. A clear way to understand this phenomenon is to hold a pen close to the face and alternate each eye to view this object.

The fusion of these two images with accommodation and convergence can provide a single **stereoscopic** image. This phenomenon (**stereopsis**) provides a strong depth cue. If the two images cannot be fused, then **binocular rivalry** will cause the user to experience just one image or part of both images.

### **Oculomotor Cues**

When dealing with binocular viewing, both eyes diverge with far-away objects and converge with near objects. This superficial muscle tension is thought to have an additional depth cue [74] called **oculomotor cues**. This muscle tension, in the visual system, is called accommodation and convergence. The focus of the eye to a given image cause the eye muscle to stretch and relax while obtaining the target image. This physical stage is called **accommodation**. The rotations of the eyes are needed to fuse the image. This muscular feedback is called **convergence**.

### **Motion Parallax**

Motion Parallax is the change of perspective of the view. This can happen if the viewer is moving in respect to the target object. The target object moves in respect to the user, or both [73]. This phenomenon will make far-away objects appear to be moving slowly, and near objects will appear to be moving faster [17]. For example, take a race where two cars are traveling at 200 kilometers per hour. The first car is passing in front of the viewer and the second car is far away on the opposite side of the track. The second car will appear to move slower, while the first car will seem to be moving faster. Another example is a car traveling on the highway, with lights and trees that seem to be moving faster, and far-away buildings appear to be moving slower.

## More About Depth Cues

There are more depth cues than the ones described above. For example, color is also used as a depth cue. The color blue appears far away because the atmosphere has a higher absorption for warm colors, giving the horizon a more accentuated blue color [73]. Adding blur to blue objects can give the illusion that an object is farther away. For additional references, see [17, 73].

### 2.5.3 Displays

There are multiple types of displays, such as conventional monitors, HMD, and optical HMD, among many others. The following is a partial list of output interfaces:

- **Conventional Monitor** is the most pervasive display found today in desktop computers, notebooks, tablets and phones. Before the liquid-crystal display (LCD), the common form was the cathode ray tube (CRT). With the right equipment, a conventional monitor can achieve stereopsis with a pair of glasses and refresh rate of at least 100 Hertz. A few years ago, most monitors did not fit the specifications mentioned, except for a few high-end displays. However, today, there are some affordable options available in the market, with 100 hertz or greater [17].
- **Head-Mounted Display** or **Helmet-Mounted Display** is a user-attached visual display commonly used in VR environments. The most recent and well-known example to consumers is the Oculus Rift, recently acquired by Facebook. There are a variety of HMDs with distinct designs; some of them may include 3D audio and other functionalities.
- **Optical Head-Mounted Display** allows the user to see through the device while still looking at the computer's graphical-generated images. The most recent example is Google glasses.

- **Surround-Screen Display** is an output device with at least three large projection displays screens, which surround the user. The screens are typically between 8 to 12 feet. The projectors are installed near the screens to avoid shadows. Front projectors can be used as long as their position avoids the user's shadows [17].

Additional output display devices exist. Some of these are Workbenches and Hemispherical Displays, among others. For a presentation in detail about the devices and further references, consult [17, 73].

## 2.6 3D Input Interfaces

3D input interfaces are devices that provide data, direction, or commands to a 3DUI, using different types of input devices. An input device does not require the user to have 6-DOF to be considered a 3D input device. This means that an input device, to be a 3D input device, has to operate in such a manner that the input is mapped to a 3D action. The distinction here between the interaction technique and input device is critical when designing a system. The input device makes reference to a physical system (e.g., computer mouse) that allows some measurement from the user (or by the user), which is then forwarded to the computer system. The interaction technique is defined as how this input device is mapped into the 3DUI system [17].

Section 2.2.4 described characteristics for input technologies, which can be applied to 3D input devices. In addition, there are specific characteristics for 3D input interfaces. The following attributes are important to consider when working with 3D input devices:

- **Degrees of Freedom:** The degrees of freedom (DOF) is the most critical characteristics for a 3DUI interaction. For example, a 6-DOF device can give you a direct mapping between translations for X, Y, Z axes and rotations about X, Y, Z axes. A

*degree of freedom* is “simply a particular, independent way that a body moves in space” [17].

- **Frequency of Data:** The frequency<sup>16</sup> of the data determines how the data is sent to the computer system. The frequency of data may be continuous components, discrete components, or both [17]. For example, the button in a GamePad will only return pressed or not pressed. The thumb-stick found in the GamePad produces continuous values for a given range.
- **Active Devices:** Devices are said to be active (or purely active) if they require the user to perform a physical action with the device. One example is the multi-touch display, which requires the user to touch the screen before any action can be taken [17].
- **Purely Passive:** Devices are said to be passive (or purely passive) if they do not require any physical action from the user to generate data. This could be a vision-based input system that is reading information from the environment, without the user input. Nevertheless, the user may work with this device as an active input system. For example, the device becomes an active system when the user performs hand-gestures in the vision-based system to issue a command [17].

## 2.7 3D Navigation

The following section covers the background for 3D navigation, which is the central topic of this thesis. This section outlines what 3D navigation is (travel), what types of tasks exist for 3D navigation, and what the characteristics in 3D navigation are.

---

<sup>16</sup>Not to be confused with sampling frequency.

### 2.7.1 3D Travel

3D navigation is defined as traveling in a virtual 3D space. This is why it is also referred to as travel<sup>17</sup>. Bowman et al. define **travel** as “the motor component of navigation — the task of performing actions that move us from our current location to a new target location or in the desired direction.” [17].

3D travel is critical for 3DUI, given that it is a common interaction task [17]. Travel is also important because it usually supports a primary task. An example is when a user is searching for objects in a game. The travel allows the user to search for the object (primary task). For this reason, 3D navigation must be designed correctly. If the user needs to think how to navigate for too long, then he or she will be distracted from the primary task, which may be to find objects in a virtual world. There are different types of travel tasks. For the study in this dissertation, it is important to understand exploration, search, maneuvering tasks, and travel characteristics.

### 2.7.2 3D Travel Tasks

The most common travel task is **exploration**. In this type of task, the user has no specified or required goal to complete. The user will only travel through the environment while building knowledge of the objects around and the locations of those objects. The most common example is navigating around a new city that the user has not known yet, to build knowledge of places and monuments, for a future visit. It is important in the **exploration** task that users are allowed to travel without constraint (other than outer limits or object collisions) around the virtual world [17].

Another type of task is called **search**. This type of task has a goal or target location within the virtual world. The user knows the final location of the required task. The user

---

<sup>17</sup>3D navigation and travel have some differences [17].



may or may not know how to get to the location, but he or she knows the objective. The search task can be divided into sub-categories. The first one is called **naïve search**, where the “user does not know the position of the target or path to it in advance” [17]. This type of search starts out as a basic exploration. The number of clues given to the user to complete the goal are limited and focused to the exploration. The second sub-category of a **search** task is called **primed search**. In the **primed search** task, the user “has visited the target before or has some other knowledge of its position” [17]. In this type of task, the user may know the final location; however, the user may still need to explore the virtual world, or the user may know the path to the target. In other words, the **primed search** provides more information to the user about the environment, in order to complete the assigned task. While there are clear differences between **naïve search** and **primed search**, the line dividing these two categories of search tasks can become blurred, depending on the design of the 3DUI and the user.

One task, often overlooked in the discussion of travel, is called **maneuvering**. This task is meant to take place “in a local area and involves small, precise movements” [17]. One example of this type of task is a user who may need to read a sign. In this scenario, the user moves slightly down and rotates  $10^\circ$  on the Y axis. The small-scale movements are critical for certain applications. A possible approach to facilitate **maneuvering** is to use vision-based systems that provide fine reading of the face (as it turns) or devices that provide physical motion with little or no error in the readings.

Another task to travel, quite useful in maps or large sets of data, is **travel-by-scaling** [17]. This technique allows intuitive zoom-in or zoom-out of a given portion of the virtual world. However, it is important to note that there are several challenges with this technique. For example, does the user understand their current position when they scaled the world? The view may be closer but the user is probably at the same location (before the zoom). Does the user understand the visual feedback when the virtual world has scaled in or scaled

out? There are different solutions to these challenges, such as using a virtual body to understand the dimensions of the scale. Another major issue is that users may have trouble performing finer movements because when scaling the virtual world, the movements are larger.

Finally, there are different travel techniques [17]. The most important techniques that must be taken into consideration when designing a system are: active versus passive, and physical versus virtual. The **active** technique is where the user has complete control of the movement of the scene. **Passive** is where the system has control of the movements. A middle ground is a **semi-automated** system, where the user has some level of control while the system automates the rest. The **physical** technique refers to the actual body of the user involved in performing the movements in the virtual scene (e.g., walking). The **virtual** technique allows the user to utilize a virtual device to control the movements.

### **3D Travel Task Characteristics**

3D travel tasks have characteristics that may be used to classify them. It is also a good idea to have these characteristics in mind when designing a system. The following list contains a few important characteristics (adapted from [17]):

- **Distance to be traveled:** This is the distance that it takes to go from location A to location B. This may require velocity control in some instances.
- **Turns:** A travel task can also look into the number of turns taken (or the number of turns required) in a given task.
- **Required DOF:** The number of DOF required for the travel task.
- **Accuracy of movements:** Travel task may need a very accurate movement (e.g., maneuvering) or realistic physics.

- **Easy to use:** Given that 3D navigation in most cases is a secondary task, the user's interaction must be intuitive.

## 2.8 Petri Nets

In the early years of the 1960s, Dr. Carl Adam Petri defined a general-purposed mathematical model<sup>18</sup>. PN is a model-oriented language. The model describes the relationship between conditions and events [35]. This provides a solution to a central challenge of computer science (as well as other fields, such as engineering), which is the construction of systems that can be specified, verified and executed, while providing a solid mathematical framework.

PN is defined as a five tuple  $N = (P, T, F, W, M_0)$ , which is the net structure. Places  $\mathbf{P}$  and Transitions  $\mathbf{T}$  are finite sets of places and transitions. The set of arcs is defined as  $F \subseteq (P \times T) \cup (T \times P)$ . The weight function  $\mathbf{W}$  is defined as  $F \rightarrow \{1, 2, 3, \dots\}$ . The dynamics definition (the dynamic state of the net) has the following rules [28]:

- A transition  $t$  is enabled if for each input place  $p$ , it has a minimum of tokens  $w(p, t)$ , the weight of the arc from  $p$  to  $t$ .
- If the transition  $t$  is enabled, then the transition may be fired.
- When an enabled transition  $t$  fires, then  $t$  removes  $w(p, t)$ , from each input place  $p$  of the given  $t$ . Once removed, the token  $w(p, t)$  is added to each output place  $p$  for the given  $p$ .

---

<sup>18</sup>Dr. Petri created the net to visualize the chemical process in 1939 [174].

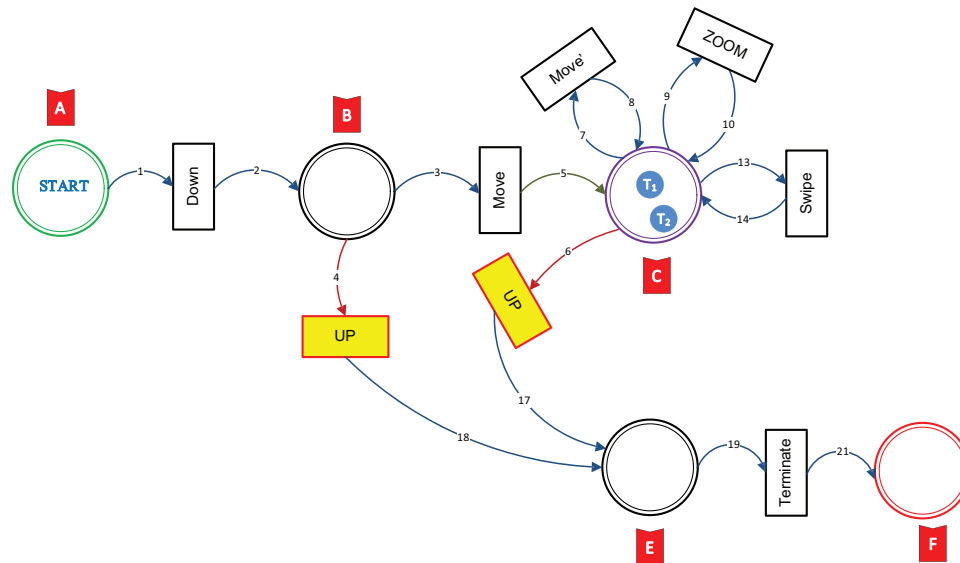


Figure 2.11: Vending Machine (Adapted from [174]).

The definition presented represents the spirit of the the PN created by Dr. Petri<sup>19</sup>, and it is what the literature may refer to as low-level PNs. HLPNs<sup>20</sup> were created out of the necessity to have a more expressive model, while keeping the formal mathematical idea of Dr. Petri. The difference between the low-level PN and HLPN could be seen as the difference between Assembly Language and high-level languages, such as Java or Python<sup>21</sup>.

As stated, HLPNs have many variations [35]. This dissertation has adopted a HLPN called Predicate Transition Net (PrT Net), which was formulated by Genrich [59]. For a basic introduction to PNs, please see [174]. For a detailed view of other HLPNs, see [35]. For a general quick overview, the reader is suggested to read [82, 146].

<sup>19</sup>Other definitions can be found. See Reisig [174].

<sup>20</sup>There are many HLPN types.

<sup>21</sup>This analogy is not meant to be an exact comparison.

## 2.8.1 Graphical Representation

PNs have the advantage of having a graphical representation of their mathematical model. In the case of a HLPN, the net consist, of places represented by circles or ellipses, transitions represented by rectangles or squares, and arcs represented by arrows. Places are discrete states, which can store, accumulate or provide information about their state. A transition can produce, consume, transport or change data. The arcs provide the connections between places and transitions, or transitions and places. It is important to note that an arc cannot connect a transition to a transition or a place to a place. A complete PN model of a vending machine is shown in Figure 2.11. Note that the red labels are not part of the graphical representation. They are placed there to identify the places with a label (e.g., A, B, ...), for clarity.

## 2.8.2 Formal Definition

Note that following definition is similar to the one used Chapter 3. However, the one here is generic and the one in Chapter 3 applies to the model described in that chapter. The HLPN used for the entire dissertation is Prt Net [59]. The Prt Net is defined as a tuple  $(N, \Sigma, \lambda)$ . This contains the Net  $\mathbf{N}$ , the specifications  $\Sigma$  and the inscription  $\lambda$ . The Net  $\mathbf{N}$  is formed with places  $\mathbf{P}$ , transitions  $\mathbf{T}$ , and connecting arc expressions (as functions  $\mathbf{F}$ ). The following is a detailed definition:

- $\mathbf{N}$  represents the PN, which is defined by a three-tuple  $N = (P, T, F)$ .

- $F \subset (P \times T) \cup (T \times P)$ .

- $P \cap T = \emptyset$ .

- The specification  $\Sigma$  represents the underlying representation of type of tokens<sup>22</sup>.

---

<sup>22</sup>Tokens may be referred to as “sorts”.

- $\Sigma = (S, \Omega, \Psi)$ .
- $\Omega$  contains the set of token operands.
- Set  $\Psi$  defines the meaning and operations in  $\Omega$ . In other words, the set  $\Psi$  defines how a given operation (e.g., plus) is implemented.
- $\lambda$  defines the arc operation.
  - $\lambda = (\phi, L, \rho, M_0)$ .
  - $\phi$  is the association of each place  $p \in P$  with tokens.
  - $L$  is the labeling inscription for the arc expressions, such as  $L(x, y) \iff (x, y) \in F$ .
  - The operation of a given arc (function) is given by  $\rho = (Pre, Post)$ .
  - These are well-defined constraint mappings associated with each arc expression, such as  $f \in F$ .
  - The **Pre** condition allows the HLPN to enable the function, if the Boolean constraint evaluates to true.
  - Then, the **Post** condition will execute the code if the function is enabled (ready to fire).
  - Finally, the initial marking is given by  $M_0$ , which states the initial state of the HLPN.

The dynamic semantics of a Prt Net is defined [28] as follows:

- Markings of Prt Net is the mapping  $M : P \rightarrow Tokens$ . In other words, places map to tokens, which is defined in the specification  $\Sigma$ .
- The variable  $e$  denotes the instantiation of expression  $e$  with  $\alpha$ , where  $e$  can be either a label expression or a constraint. Therefore, the occurrence mode of  $N$  is a substitution  $\alpha = \{x_1 \leftarrow c_1, \dots, x_n \leftarrow c_n\}$ .

- For a given marking  $M$ , a transition  $t$  in  $T$ , and an occurrence mode  $\alpha_{enabled}$ ,  $t$  is enabled if and only if:

$$- \forall p \in P(\bar{L}(p,t):\alpha \subseteq M(p)) \wedge R(t) : \alpha \text{ where: } \bar{L}(x,y) = \begin{cases} L(x,y), & \text{if } (x,y) \in F \\ \emptyset, & \text{otherwise} \end{cases}$$

- If  $t$  is  $\alpha_{enabled}$  at  $M$ ,  $t$  may fire in occurrence mode  $\alpha$ .
- An execution sequence  $M_0[t_0 > M_1[t_1 > \dots$  is either finite when the last marking is terminal (no more transitions are enabled), or infinite.
- The behavior of the net model is the set of all execution sequences, starting from  $M_0$ .

## CHAPTER 3

### TOWARD 3D NAVIGATION WITH MULTI-TOUCH INTERACTIONS

With the explosion of modern input devices (e.g., multi-touch, WiiMote, Kinect), a set of new challenges has surfaced. With the challenges, a great opportunity has been presented to the current HCI researchers to close the gap between technology and users. The following sections describe the contributions from this research towards 3D navigation dealing with multi-touch systems.

#### 3.1 Multi-Touch Feature Extraction

As stated in the literature review, there are different approaches to gesture recognition. The approach described here uses feature extraction to determine the type of gesture. These sections describe the evolution of the Feature Extraction Multi-Touch System (FETOUCH), from an offline algorithm to a real-time solution.

##### 3.1.1 FETOUCH

FETOUCH was tested using offline data in Windows 7 with multi-touch technology [113], Microsoft Visual Studio (using C# language) and a 3M M2256PW multi-touch display. Windows 7 provides either detection of pre-defined gestures or unclassified raw-touch data when using its Application Programming Interface (API). FETOUCH uses the raw-touch data because it provides flexibility to create custom gestures, as well as to test different methods for detection.

When using raw touch data, most systems where multi-touch is available (e.g., iOS, Windows) provide a **trace**, which contains a set of points with coordinates  $x$  and  $y$ . In addition, the system will generate a **timestamp** for each point and a unique **ID**, which indicates the trace that it belongs to. The system generate events when the trace is activated (finger



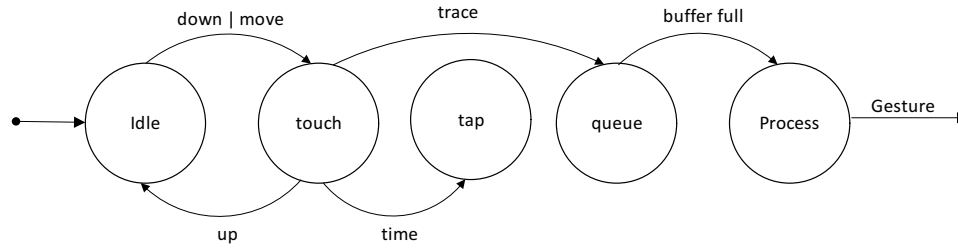


Figure 3.1: State Machine

down), moved (finger moving), and deactivated (finger up). Each touch includes the ID that is given at the moment of TOUCHDOWN, to be used during the TOUCHMOVE, and to end when TOUCHUP has been deactivated. The ID gives us a way to group points from each finger. For specific information on how Windows 7 handles multi-touch technology, please see<sup>1</sup> [113].

FETOUCH combines feature extraction with a finite state machine (FSM) [192], as shown in Figure 3.1. The idea is to allow a state machine to keep control of the process while the detection takes place. The input device’s state starts as idle. Here we have a state transition to the touch state with either down or move events. Once the finger has been lifted, we have a state transition back to idle with the UP event. Once the touch state, has been reached, the decision must be made to identify it as either a tap (e.g., double tap, two-finger tap) or a trace. After testing the system, it was noticed that differentiating taps from gestures was unnecessary because the tap can be part of the set of gestures to recognize. Nevertheless, maintaining a control of the states with a FSM is still useful.

When a point is detected, then the data is added to a to a thread-safe queue<sup>2</sup> [221]. The queue is used to continue storing the traces while a specific window (buffer) of data is processed. Once the queue is full, a transition will take place to the “process” state, where Algorithm 3.1 takes over.

---

<sup>1</sup>Windows 8 has additional features. Refer to [msdn.microsoft.com](http://msdn.microsoft.com).

<sup>2</sup>The reader can find ready to use thread-safe data structures.

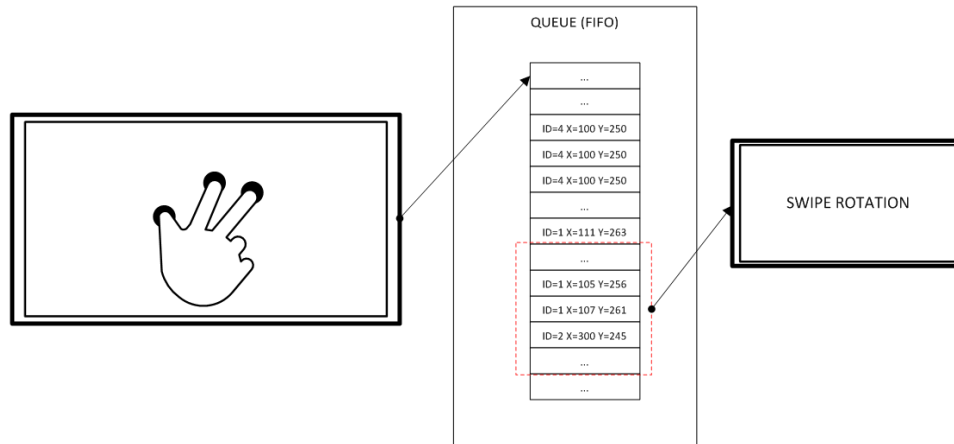


Figure 3.2: Queue

Figure 3.2 gives more details about the queue and how we are using it. On the left side of the figure, a three-finger gesture is performed by a user. The data points are stored in the buffer as those points are coming in. The window must be of a large enough size to detect a gesture (e.g.,  $N = 64$ ). By having this buffer, the user can perform multiple gestures while keeping his or her hands on the display (if desired). Figure 3.2 shows this thread-safe queue with a window size, for a given gesture.

Algorithm 3.1 detects swipe (translation), rotate, and pinch in/out (zoom in/out). If a system does not provide traces, one of the many clustering techniques available can be used to create them [57]. Given the critical nature of a real-time application, any necessary pre-computations must be performed while the traces are added to the queue. This is why, when running the algorithm, it is expected that the grip will have been pre-computed.

The primary motivation is to lower the running time of the gesture detection in order to use it with demanding 3D applications. Therefore, the running time of the algorithm is as important as the complete utilization of all the resources available in the system. In this context, it is important to note that the gesture detection runs in its own thread. For more details about this multi-threaded approach, a C++ implementation can be found in [221]

or a more detailed explanation with Java code can be found in *The Art of Multiprocessor Programming* [85].

Algorithm 3.1 starts by popping the buffer window in line 1. Since the data is collected offline, the buffer is divided into a top half and bottom half buffer (initial and final states). Before explaining the algorithm in more detail, the following variables are defined next: traces, trace, grip, trace vector, spread, and angle rotation.

**Traces** is a set that contains information for the path taken by each finger. In other words, for each finger, a set of properties is pre-computed, which is called **trace** in the algorithm. For example, the x and y coordinates are the average of a given trace, as shown in Equations 3.2 and 3.3 (for the y coordinate, replace the “x” for the “y”). Note that the variable  $n$  in the formulas refers to the total x and y points for a given trace. Because the buffer is divided into two snapshots, each snapshot has its own average. A **grip** is defined by the average of all points in each snapshot, as shown in Equation 3.4. A **trace vector** is defined as trace minus the grip, as shown in Algorithm 3.1, lines 12 through 15. The **spread** is given by lines 18-19 in Algorithm 3.1, which calculate the spread as the average difference between the grip point and the touch vector. Finally, the **angle rotation** is the average of the angle obtained by the formula  $atan2$  (see Equation 3.1) [39]. This is the angle between the final touch vector and the initial touch vector.

$$atan2(y,x) = \begin{cases} 0, & x = 0, y = 0, \\ +90^\circ, & x = 0, y > 0, \\ -90^\circ, & x = 0, y < 0, \\ arctan(y/x), & x > 0, \\ arctan(y/x) + 180^\circ, & x < 0, y \geq 0 \\ arctan(y/x) - 180^\circ, & x < 0, y < 0. \end{cases} \quad (3.1)$$

Finally, the chosen gesture is given by any of the three distance variables (swipeDistance, rotDistance, or zoomDistance) with the highest value found, as shown in Algorithm 3.1. The swipe distance is given by the spread of the first trace and the grip. The rotate distance is given by the arc length. This is calculated using average angle obtained in line 20 and the radius of the swipe distance. Remember that  $atan2$  [39] values range between  $\pm\pi$ . In order to obtain the distance, the proper factor 2 must be multiplied, as shown in Equation 3.5. The zoom distance is given by the average final spread distance and the average initial spread distance.

Once everything is computed in the for loop, all we have left to do is to determine the correct gesture. The gesture detected is assigned according to the highest distance value of the swipe distance, rotation distance or zoom distance. Additional information can be obtained for specific detected gestures. For example, if the gesture detected is a zoom gesture, the direction of the zoom can be obtained. While the primary goal of the algorithm is to find the correct gesture, additional information is important to be precise about the gesture. Algorithm 3.1 concentrates on finding the gesture type.

FETOUCH allows gesture detection while using off-line data for multi-touch displays. It is important to note that FETOUCH is aligned to a problem statement written by Greg Hamerly<sup>3</sup>, which provided some validation into the work of FETOUCH and ideas about the feature extraction problem already described. For an expanded discussion, see [161].

$$iTrace[id].x = \frac{1}{n/2} \sum_{i=0}^{\frac{n}{2}-1} trace[id][i].x \quad (3.2)$$

$$fTrace[id].x = \frac{1}{n/2} \sum_{i=\frac{n}{2}}^{n-1} trace[id][i].x \quad (3.3)$$

$$iGrip.x = \frac{1}{n/2} \sum_{t \in \mathbf{iTrace}} iTrace[t].x \quad (3.4)$$

---

<sup>3</sup>[http://cs.baylor.edu/~hamerly/icpc/qualifier\\_2012/](http://cs.baylor.edu/~hamerly/icpc/qualifier_2012/)

---

**Algorithm 3.1** GestureDetection

---

**Require:** TouchCount > 0

```
1: traces ← traceQueue.getWindow()
2: iTrace ← traces.getHalf()
3: fTrace ← traces.getHalf()
4: iGrip.x ← iTrace.getGrip.x
5: iGrip.y ← iTrace.getGrip.y
6: fGrip.x ← fTrace.getGrip.x
7: fGrip.y ← fTrace.getGrip.y
8: ivFirst.x ← iTrace[1].x − iGrip.x
9: ivFirst.y ← iTrace[1].y − iGrip.y
10: swipeDistance ← sqrt(ivFirst.x2 + ivFirst.y2)
11: for t = 1 to traces.Count do
12:   iv.x ← iTrace[t].x − iGrip.x
13:   iv.y ← iTrace[t].y − iGrip.y
14:   fv.x ← fTrace[t].x − fGrip.x
15:   fv.y ← fTrace[t].y − fGrip.y
16:   di ← sqrt(iv.x2 + iv.y2)
17:   df ← sqrt(fv.x2 + fv.y2)
18:   iSpread ← iSpread + di
19:   fSpread ← fSpread + df
20:   angle ← atan2(fv.y − iv.y, fv.x − iv.x)
21:   rotAngle ← rotAngle + angle
22: iSpread ← iSpread / traces.Count
23: fSpread ← fSpread / traces.Count
24: rotAngle ← rotAngle / traces.Count
25: zoomDistance ← fSpread − iSpread
26: rotDistance ← rotAngle / 360.0 * 2 *  $\pi$  * swipeDistance
27: return Gesture With Highest Distance
```

---

$$rotDistance = \frac{\Theta}{360} 2\pi r \quad (3.5)$$

### 3.1.2 FETOUCH+

FETOUCH allowed the understanding of working with specific features when using touch. However, the need for a real-time algorithm that would allow gesture detection was imperative. This is why FETOUCH+ was developed. This new approach took the ideas from the off-line algorithm and tested them in real-time. FETOUCH+ was developed with Windows 7 using the multi-touch available in the Windows API (WINAPI) [113], Microsoft Visual Studio (using C# 4.0 language), and a 3M M2256PW multi-touch monitor. This time, the use of C# **Tasks**<sup>4</sup> gave the process a multi-thread approach while keeping the implementation simpler. The testing and implementation was performed in C#, and the description of the algorithm is specific and provides details about how to implement it in other languages.

There are some definitions similar to FETOUCH. Nevertheless, it is important to state them again, as they encapsulate the new algorithm with small differences. First, raw touch data [113] (e.g., Windows, iOS) was used to capture data points stored in a set called **trace**. A **trace** starts when a user presses with one finger and continues until the user removes the finger from the device. Figure 3.3 shows a rotation gesture with two fingers. This constitutes two traces. Each **trace** has a unique identifier (**id**) and contains a set of points with 2D coordinates (**x,y**) and a timestamp **t**, for each point. The general events provided are the initial touch of a trace (TOUCHDOWN), the movement of the trace (TOUCHMOVE) and the end of the touch (TOUCHUP). A **trace point** structure contains coordinates **x** and **y**, timestamp **t<sub>0</sub>**, count **c** (indicating how many continuous points are found in the same location), Boolean **p** (indicating if the touchpoint was already processed) and the last timestamp

---

<sup>4</sup>Task is a higher level of implementation for using threads.



Figure 3.3: Rotation Gesture

$t_1$ . The **trace point** is also referred to as TOUCHPOINT. An additional data structure with the name of TRACE is also stored. This contains **id** for the unique identifier, the initial timestamp  $t_i$ , the final timestamp  $t_f$ , and the Boolean **d** to see if the trace is ready for deletion. For additional information on how Windows 7 handles the touch API, see [113].

It is important to keep the touch events and gesture detection in different thread processes [221]. Therefore, all the active traces are stored in a concurrent hash map and a concurrent arraylist (vector) to keep the set of touch points of a given trace. Once data points are processed, they can be safely removed. The advantage to having them in different threads (other than speed) is to have a real-time approach to gesture detection. A buffer with a maximum size of **windowSize** is defined. This means that when the buffer is full, it needs to perform the gesture detection while still allowing touch events to execute. During test trials, it was found that the windowSize works best for  $N = 32$  or  $N = 64$ .

TOUCHDOWN is the initial event that fires when a trace begins. This means that a user has pressed a finger on the multi-touch device. The event fires for each new trace. There is room to further improve the performance of the gesture detection system by creating additional threads for each trace. The first trace is stored in the vector **vtrace**, during this event. The vector is kept in a hash map **mtrace**, which contains a collection of all traces. For the first trace, The timestamp  $t_0$  is equal to the timestamp  $t_1$ .

As the user moves across the screen (without lifting his or her fingers), the event that fires is TOUCHMOVE (Listing 3.2). In line 3 of Listing 3.2, a method called *removeNoise*

is invoked by passing the previous and current traces. It is important to note that the algorithm may benefit from removing the noise or undesired points from the data while it is running. For example, one may consider that noise occurs if a new touch point is within  $\pm \mathbf{d}$  of a previous point, where  $\mathbf{d}$  is a pre-calculated value depending on the size of the screen (e.g., 2). If the noise variable is true, the counter  $\mathbf{c}$  and the timestamp  $\mathbf{t}_1$  need to be updated for this trace, as shown in lines 5-6. Otherwise, the touch point is added to the vector. At the end of the procedure, in line 13, the map is updated (depending on the implementation and language, this may be skipped). The noise removal depends on the actual requirement of the system.

---

**Algorithm 3.2** TOUCHMOVE

---

```

1:  $trace \leftarrow TRACE(id)$ 
2:  $vtraces \leftarrow mtraces.find(id)$ 
3:  $noise \leftarrow removeNoise(trace, vtrace)$ 
4: if  $noise$  then
5:    $trace.c+ = 1$ 
6:    $trace.t_1 = trace.requestTimeStamp()$ 
7: else
8:    $trace.t_1 \leftarrow trace.requestTimeStamp()$ 
9:    $trace.t_0 \leftarrow vtraces[length - 1].t_0$ 
10:  $vtrace \leftarrow mtraces.getValue()$ 
11:  $vtrace.push\_back(traces)$ 
12:  $mtraces.insert(id, vtrace)$ 

```

---

The final event is when the user removes his or her finger. Since the gesture detection algorithm may still be running, the data is marked for deletion only. Once Algorithm 3.3 finishes, the process can safely delete all the data touch points that have been used.

Algorithm 3.3 detects translation(swipe), rotation, and zooming (pinch). Once the buffer is full, the window of touch data is split into two lists named **top** and **bottom**. This creates an initial and a final snapshot to work with.

Algorithm 3.3 requires pre-computed values, grip points, and average touch points (described below) before it can execute. The choices to pre-compute the values can be done in



the TOUCHMOVE event or by firing a separate process before Algorithm 3.3 starts. The values for Algorithm 3.3 are stored in the **top** and **bottom** structures, respectively.

The features identified in each gesture are **grip**, **trace vector**, **spread** and **angle rotation**. A **grip** is the average of all points in each top and bottom list. A **trace vector** is a **trace** minus the **grip**, as shown in Algorithm 3.3, lines 11 through 14. The **spread** is calculated in lines 15-18 of Algorithm 3.3 as the average distance between the **grip point** and **the touch vector**. The **angle rotation** is given by the average of the angles obtained by *atan2* [39], which is the angle between the final touch vector and the initial touch vector. This is exactly the same as in FETOUCH.

To select the correct gesture, the algorithm finds the highest value from the three distance variables: *swipeDistance*, *rotDistance*, or *zoomDistance*. The definition of the *swipe distance* is the spread of the first trace and the grip. The *rotate distance* is calculated to be the arc length, which is given by the the radius of the *swipe distance* and the average angle, shown in lines 19-20 of Algorithm 3.3. It is important to note that *atan2* [39] values range between  $\pm\pi$ . This is why there is a factor of 2 in line 26 (just like in FETOUCH). Finally, the *zoom distance* is defined as the difference between the average final spread distance and the average initial spread distance. FETOUCH+ demonstrates the ability to find features that can be used to detected a gesture. This is an alternative to using template matching. For an expanded discussion, see [156].

### 3.1.3 Implementation: FETOUCH++

After testing FETOUCH and FETOUCH+, a few modifications were required to have a more optimized system. This is an enhanced version of FETOUCH+, meant to find additional features. This implementation is referred to as FETOUCH++. All the concepts of FETOUCH+ apply in FETOUCH++.

---

**Algorithm 3.3** GestureDetection

---

```
1:  $top \leftarrow traces.getTop(windowSize)$ 
2:  $bottom \leftarrow traces.getBottom(windowSize)$ 
3:  $tGrip.x \leftarrow top.getGrip.x$ 
4:  $tGrip.y \leftarrow top.getGrip.y$ 
5:  $bGrip.x \leftarrow bottom.getGrip.x$ 
6:  $bGrip.y \leftarrow bottom.getGrip.y$ 
7:  $spread.x \leftarrow iTrace[1].x - iGrip.x$ 
8:  $spread.y \leftarrow iTrace[1].y - iGrip.y$ 
9:  $swipeDistance \leftarrow \sqrt{spread.x^2 + spread.y^2}$ 
10: for  $t = 1$  to  $traces.Count$  do
11:    $i.x \leftarrow tTrace[t].x - tGrip.x$ 
12:    $i.y \leftarrow tTrace[t].y - tGrip.y$ 
13:    $f.x \leftarrow bTrace[t].x - bGrip.x$ 
14:    $f.y \leftarrow bTrace[t].y - bGrip.y$ 
15:    $di \leftarrow \sqrt{i.x^2 + i.y^2}$ 
16:    $df \leftarrow \sqrt{f.x^2 + f.y^2}$ 
17:    $iSpread \leftarrow iSpread + di$ 
18:    $fSpread \leftarrow fSpread + df$ 
19:    $angle \leftarrow \text{atan2}(f.y - i.y, f.x - i.x)$ 
20:    $rotAngle \leftarrow rotAngle + angle$ 
21:  $iSpread \leftarrow iSpread / traces.Count$ 
22:  $fSpread \leftarrow fSpread / traces.Count$ 
23:  $rotAngle \leftarrow rotAngle / traces.Count$ 
24:  $zoomDistance \leftarrow fSpread - iSpread$ 
25:  $rotDistance \leftarrow rotAngle / 360.0 * 2 * \pi * swipeDistance$ 
26: return Gesture With Highest Distance
```

---

The first objective was to find the lowest number of points required to be able to output a gesture while the gesture was active. The number found was 32, as shown in Listings 3.2 and 3.3. It is important to note that if the number of gestures to be recognized increases from the set tested on, this parameter may need to be incremented to 64.

A small difference from FETOUCH+, during the onDown event, is the finger count (fingers++), as shown in Listing 3.1. The idea is that while the traces can determine how many fingers are currently being used, there is a need to keep a separate value of the number of fingers touching the screen, to run the clean-up process, in the onUp (Listing 3.2) event. Of course, this depends on the definition of a multi-touch gesture. In the case of

FETOUCH++, the gesture recognition is defined as active, while fingers are on the surface display. The partial recognition will be fired every N samples (e.g., 32) to see if a gesture can be detected, as shown in Listing 3.3. The onUp event also fires the recognition method.

The recognition of FETOUCH++ is very similar to FETOUCH+. First, the split function (Listing 3.4) divides the points for each trace into two halves (top and bottom). This is performed in parallel using the **Task** class from C#. An example of extracting the top half is shown in Listing 3.5. Once this is completed, the split function calculates the features to be used with the recognition process by calling the method *getFeatures*, as shown in Listing 3.6. In conclusion, FETOUCH++ has allowed the possibility to investigate further features that can be helpful for multi-touch devices.

Listing 3.1: CloudFeatureMatch (onDown)

```
1 private void OnDown(WMTouchEventArgs e)
2 {
3     fingers++;
4     incPoints();
5     M.Point p = new M.Point(e.LocationX, e.LocationY, e.Id);
6     map.Add(e.Id, p);
7 }
```

Listing 3.2: CloudFeatureMatch (onUp)

```
1 private void onUp(WMTouchEventArgs e)
2 {
3     incPoints();
4     M.Point p = new M.Point(e.LocationX, e.LocationY, e.Id);
5     map.Add(e.Id, p);
6     fingers--;
7     if (localPoints >= 32 || fingers == 0)
8         recognize();
9     if (fingers == 0)
10        cleanup();
11 }
```

Listing 3.3: CloudFeatureMatch (onMove)

```
1 private void onMove(WMTouchEventArgs e)
2 {
3     incPoints();
4     M.Point p = new M.Point(e.LocationX, e.LocationY, e.Id);
5     map.Add(e.Id, p);
```

```

6     if ( localPoints >= 32)
7         recognize();
8 }

```

Listing 3.4: CloudFeatureMatch (Slit)

```

1 private void split(int half,int tCount)
2 {
3     int avgHalf = half;
4     Task<Point>[] taskSplitArray = new Task<Point>[tCount * 2];
5     int[] keys = mapTraces.getKeys();
6     int i = 0;
7     foreach (int k in keys)
8     {
9         taskSplitArray[i] = Task<Point>.Run(
10             () => { return BreakTop(k, avgHalf); } );
11         taskSplitArray[i+1] = Task<Point>.Run(
12             () => { return BreakBottom(k, avgHalf); });
13         i += 2;
14     }
15     Task<Point>[] t = {
16         Task<Point>.Run( () => { return getFeatures(taskSplitArray);} )
17     };
18     Task<Point>.WaitAll(t);
19 }

```

Listing 3.5: CloudFeatureMatch (BreakTop)

```

1 private Point BreakTop(int trace,int avgHalf)
2 {
3     int half = getHalf(trace, avgHalf);
4     Points pts = new Points(avgHalf);
5     float avgX = 0;
6     float avgY = 0;
7     for (int i = 0; i < half -1; i++)
8     {
9         Point p = mapTraces[trace][i];
10        pts.addPoint(p);
11        avgX += p.X;
12        avgY += p.Y;
13    }
14    mapTop = new MapPoints();
15    mapTop.AddPoints(trace, pts);
16    Point avgPoint = new Point(avgX / half, avgY / half); ;
17    avgPoint.CountFromAverage = half;
18    return avgPoint;
19 }

```

Listing 3.6: CloudFeatureMatch (Features)

```
1 private GestureDetected getFeatures(Task<Point>[] taskSplitArray)
2 {
3     traceFeatureList = new TraceFeatureList();
4     for (int i = 0; i < taskSplitArray.Length; i += 2)
5     {
6         Point p = taskSplitArray[i].Result;
7         Point p2 = taskSplitArray[i + 1].Result;
8         Point p3 = new Point((p.X + p2.X) / 2, (p.Y + p2.Y) / 2);
9         TraceFeatures tf = new TraceFeatures();
10        tf.addFeature(TraceFeatures.FeatureType.topAvg, p);
11        tf.addFeature(TraceFeatures.FeatureType.bottomAvg, p2);
12        tf.addFeature(TraceFeatures.FeatureType.Avg, p3);
13        traceFeatureList.traceFeatures.Add(tf);
14    }
15    return traceFeatureList.getGestureDetected();
16 }
```

## 3.2 GyroTouch



Figure 3.4: WiiMote with Motion Plus

In the search for a more intuitive interaction, the **Gyroscope Multi-Touch System (GyroTouch)** was developed to combine touch and a gyroscope. The first test was conducted with a Nintendo WiiMote with MotionPlus (see Figure 3.4) using its gyroscope. However, the switch between the WiiMote and the multi-touch was not user friendly. This is why **GyroTouch** was implemented using a gyroscope mounted on a wrist band, as shown in Figure 3.5c.

**GyroTouch** was developed with Visual Studio 2012 (using C++ language) running on a Windows 7 platform with a 3M 22-inch multi-touch display, and a Micro-electro-

mechanical systems (MEMS) module by YEI Technology (3 Space Sensor, shown in Figure 3.5c). The 3 Space Sensor (which is wireless) is used in the non-dominant hand of the user, as shown in Figure 3.5a. For the 3D rendering, the engine used was OGRE3D (shown in Figures 3.5a and 3.5b). By complementing touch with a gyroscope, the user can perform additional movements while keeping both hands free to use other devices if needed (e.g., keyboard). This allows the user to perform some of the rotations with the gyroscope and the rest using the touch, which allows them to keep the tactual feedback intact for some gestures.

**GyroTouch** allows the combination of multi-touch and inertial measurement unit IMU<sup>5</sup> devices. The motivation of GyroTouch is to support additional gestures while keeping the hands free. This also allows users to use devices like the Leap Motion because their hands are free. By keeping the hands free, multiple input devices can be used, as shown in Figure 3.5b. **GyroTouch** is designed with usability in mind and with the vision that smart watches will become more pervasive over time.

The approach used is to keep the touch algorithm to detect swipe, zoom and rotate gestures for the touch. The touch is used for translating in X and Y (using two-finger swipe). For Z translation, the one-finger swipe (same direction as the y-axis) is mapped to the z translation. A two-finger rotation is mapped to the z-axis (yaw), for the rotation, as is commonly done with touch tablets. In order to keep the interaction as natural as possible, the touch is complemented with the gyroscope for rotations about the x-axis (roll) and the y-axis (pitch) using the gyroscope.

The touch algorithm is described in Section 3.1. It consists of finding certain characteristics for each gesture using a very fast and simple algorithm. The gyroscope found in the MEMS, shown in Figures 3.5a and 3.5c, is used only to indicate the roll and pitch rotations, whereas the third rotation is indicated via the touch interaction. Algorithm 3.4 shows

---

<sup>5</sup>The IMU device uses MEMS technology.

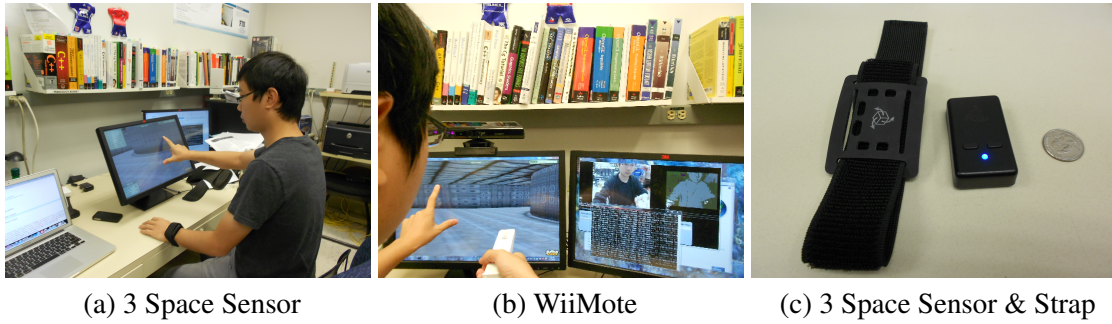


Figure 3.5: GyroTouch

the integration over time of the gyroscope signal, using the current and previous samples, required to obtain the angle of rotation about the x-axis. The same applies for each of the other two axes. The sensor already provides data processed by a Kalman Filter. In addition, the data can be filtered by the designer if needed, to remove noise within a threshold. In addition, the designer may provide a threshold for the idle state if needed. A possible additional filter is to ignore the gyroscope when the touch is in progress. Those are design choices. The sampling rate varies depending on the sensor. The 3 Space Sensor uses a sampling rate of 160 Hz (with a possible maximum for this device of 800Hz). This makes the period  $T = 1/FS$  or  $1/160$ . Since the data is already normalized, there is no need to use the *midLevel* and the *unitDegree* variable in Algorithm 3.4. Hence, for **GyroTouch**, those values are set to 0 and 1, respectively. It is important to point out that this is not always the case for all devices. For example, for the WiiMote with MotionPlus, it is necessary to set the *midLevel* at  $2^{13}$  and the unit degree at  $8192/592$ . The **GyroTouch** provides a way to complement and augment multi-touch interaction. It uses a watch form factor that allows for convenience and is non-intrusive. This is explained in more detail in [157, 158].

---

**Algorithm 3.4** Rotation Algorithm for a Gyroscope

---

**Require:** midLevel=0 & unitDegree = 1 for 3Space Sensor

```
1:  $roll \leftarrow rawdata.roll - midLevel$ 
2:  $rot.x[0] \leftarrow rot.x[1]$ 
3:  $rot.x[1] \leftarrow roll$ 
4:  $omega.x[0] \leftarrow omega.X[1]$ 
5:  $omega.x[1] \leftarrow roll.x[1]/unitDegree$ 
6:  $x \leftarrow angle.x[1]$ 
7:  $angle.x[1] \leftarrow x + T * ((omega.x[1] + omega.x[0])/2)$ 
8: return angle.x[1] as roll
```

---

### 3.2.1 Implementation

The implementation presented here shows an example of Algorithm 3.4 using the WiiMote. The concepts can be extended to any gyroscope device. For this dissertation, the library used is called **WiiYourSelf**<sup>6</sup>, which was extended from an original C# version by Brian Peek<sup>7</sup>. This library supports the WiiMote with many extensions, such as the Nintendo Wii MotionPlus. The actual concrete implementation is shown in Listing 3.8. This implementation deals with the gyroscope using the wiiMote Motion Plus. This is analogous to Algorithm 3.4. It is important to remember to initialize the WiiMote, as shown in Listing 3.7.

Listing 3.7: WiiMote (Initialize)

---

```
1 ...
2 m_wiiMote.ChangedCallback = on_wiimote_state_change;
3 m_wiiMote.CallbackTriggerFlags = (state_change_flags)(CONNECTED |
  EXTENSION_CHANGED | MOTIONPLUS_CHANGED);
4 ...
```

Listing 3.8: Gyroscope (wiiMote)

---

```
1 const double FS = 90;
2 const double T = 1.0/FS;
3 const double maxmp = 16383;
4 const double middle = 8192;
```

---

<sup>6</sup>See <http://wiiyourself.gl.tter.org>.

<sup>7</sup><http://blogs.msdn.com/b/coding4fun/archive/2007/03/14/1879033.aspx>.



```

5  const double rangeMax = middle - 1;
6  const double u_mid = middle / maxmp;
7  const double unitDegree = 8192.0/595.0;
8  const double fastSpeedFactor = 2000.0/440.0;
9  static Rotation lambda = {250.00,500.0,250.0} ;
10 static Rotation offset = {0,0,0};
11 static bool calibrated = false;
12 static Rotation mid = {0.0,0.0,0.0};
13 static unsigned long tCount;
14 Rotation current = {0,0,0};
15 Rotation raw = {0,0,0};
16 Rotation currentPrime = {0,0,0};
17 OgreFramework::t_RotationSpeed rspeed = {false,false,false};
18 static Rotation Omega[] = { {0.0,0.0,0.0} , {0.0,0.0,0.0} };
19 static Rotation RawG[] = { {0.0,0.0,0.0} , {0.0,0.0,0.0} };
20 static Rotation Angle[] = { {0.0,0.0,0.0} , {0.0,0.0,0.0} };
21 Rotation Delta = {0.0,0.0,0.0};
22 raw.yaw = (double)remote.MotionPlus.Raw.Yaw;
23 raw.pitch = (double)remote.MotionPlus.Raw.Pitch;
24 raw.roll = (double)remote.MotionPlus.Raw.Roll;
25 current.yaw = raw.yaw - middle - offset.yaw ;
26 current.pitch = raw.pitch - (middle);
27 current.roll = raw.roll - (middle);
28 if (!calibrated)
29 {
30     if (abs(current.yaw) > lambda.yaw
31         || abs(current.pitch) > lambda.pitch
32         || abs(current.roll) > lambda.roll)
33     {
34         return;
35     }
36     else
37     {
38         calibrated = true;
39         offset.yaw = current.yaw;
40         offset.pitch = current.pitch;
41         offset.roll = current.roll;
42     }
43 }
44 RawG[0].yaw = RawG[1].yaw;
45 RawG[0].pitch = RawG[1].pitch;
46 RawG[0].roll = RawG[1].roll;
47 RawG[1].yaw = current.yaw;
48 RawG[1].pitch = current.pitch;
49 RawG[1].roll = current.roll;
50 Omega[0].yaw = Omega[1].yaw;
51 Omega[0].pitch = Omega[1].pitch;

```

```

52 Omega[0].roll = Omega[1].roll;
53 rspeed.yaw = remote.MotionPlus.SlowBit.Yaw;
54 rspeed.pitch = remote.MotionPlus.SlowBit.Pitch;
55 rspeed.roll = remote.MotionPlus.SlowBit.Roll;
56 Omega[1].yaw = RawG[1].yaw / unitDegree ;
57 Omega[1].pitch =RawG[1].pitch / unitDegree ;
58 Omega[1].roll = RawG[1].roll / unitDegree ;
59 Angle[0].yaw = Angle[1].yaw;
60 Angle[0].pitch = Angle[1].pitch;
61 Angle[0].roll = Angle[1].roll;
62 Angle[1].yaw = Angle[0].yaw + T * ( ( Omega[1].yaw + Omega[0].yaw ) /
    1.0);
63 Angle[1].pitch = Angle[0].pitch + T * ( ( Omega[1].pitch + Omega[0].pitch
    ) / 2.0);
64 Angle[1].roll = Angle[0].roll + T * ( ( Omega[1].roll + Omega[0].roll ) /
    2.0);
65 m_RotationValues.yaw = Angle[1].yaw ;
66 m_RotationValues.pitch = Angle[1].pitch ;
67 m_RotationValues.roll= Angle[1].roll ;
68 Delta.yaw = Angle[1].yaw - Angle[0].yaw;
69 Delta.roll = Angle[1].roll - Angle[0].roll;
70 Delta.pitch = Angle[1].pitch - Angle[0].pitch;
71 if (current.yaw != 0 || current.roll != 0 || current.pitch != 0)
72 {
73     if (remote.Button.Two() && m_RotationValues.yaw != 0)
74         m_pCamera->yaw(Degree(Delta.yaw)) ;
75     if (remote.Button.B() && m_RotationValues.pitch != 0)
76         m_pCamera->pitch(Degree(Delta.pitch)) ;
77     if (remote.Button.A() && m_RotationValues.roll != 0)
78         m_pCamera->roll(Degree(Delta.roll)) ;
79 }

```

### 3.3 PeNTa: Petri Nets

The explosion of new input devices has added many challenges to the implementation of input user interfaces. **Petri Net Touch (PeNTa)** addressed this problem by offering a mathematical approach to modeling input, using HLPN. The actual HLPN used is called Prt Net, described in the background section, Chapter 2.

### 3.3.1 Motivation and Differences

It is important to note that this approach is not targeted toward the end-user. The target audiences for **PeNTa** are four: (1) Software developers who would like to graphically model multi-touch interactions. (2) Framework developers<sup>8</sup> who wish to incorporate modern input devices to their libraries. (3) Domain-Specific Language (DSL) developers who create solutions for domain-experts. (4) Researchers in the HCI field.

There are several reasons why the preferred modeling tool to tackle the problem was HLPN. There are several approaches which are described in the literature review section, Section 1.6. The first difference to consider is between low-level PN vs HLPN. The major difference is that HLPN have “complex data structures as data tokens, and [use] algebraic expressions to annotate net elements” [152]. This is similar to the difference between assembly language versus a high-level language (e.g., Python). For the complete standard defining HLPN, see [152]. HLPN is still mathematically defined as the low-level petri-net but it can provide “unambiguous specifications and descriptions of applications” [152].

There are further reasons that were considered when picking the use of HLPN to model multi-touch interactions. These reasons might be better understood when **PeNTa** is placed in the context of other existing approaches for input modeling: Proton and Proton++ [112], Gesture Coder [130], and GestIT [75, 197, 199]. While **PeNTa** offers more expressiveness and distributed capabilities, simultaneously providing a solid mathematical framework, the work offered by Proton++, Gesture Coder, and GestIT offers great insight in modeling multi-touch interaction, with different benefits that must be evaluated by the developer. Finally, it is important to note that the work of **PeNTa** is inspired in part by Proton and Proton++.

---

<sup>8</sup>Library and/or language developers also fit in this category.

The question still may remain for some readers: Why use HLPN to define multi-touch interactions? PNs provide graphical and mathematical representations that allow verification and analysis; furthermore, they provide a formal specification that can be executed. This is important. They can be used to specify, to verify, and to execute. Petri Nets also allow distributed systems to be represented. Finally, a finite state machine can be represented as a PN, but a PN may not be represented as a FSM. In other words, PNs have more expressive representational power.

Proton and Proton++ offer a novel approach to multi-touch interaction using RegEx. Such an approach offers an advantage to those who are proficient with regular expressions. However, there may be some disadvantages in using RegEx for our goals. Expressions of some gestures can be lengthy, such as the scale gesture [112]:  $D_1^s M_1^s * D_2^a (M_1^s | M_2^a) * (U_1^s M_2^a * U_2^a | U_2^a M_1^s * U_1^s)^9$ . Spano et al. [199] presented additional differences between PNs and the use of RegExs in Proton++. Another potential disadvantage of using RegExs is that some custom gestures may become harder to represent.

The Gesture Coder method offers a great approach to creating gestures by demonstration. PeNTa could also create a HLPN using machine learning. The representation of the training for the Gesture Coder results in a FSM. FSMs can become large and does not provide the expressive power and distributed representation of HLPNs.

GestIT is the closest approach to PeNTa. GestIT uses low-level PNs. The approach of GestIT is similar to Proton++, but it uses a PN. The trace is broken down into points and the gesture becomes a pattern of those points. The expressiveness of HLPNs allow the Proton++ technique to be used. However, this author chose to define the tokens as individual traces. GestIT represents a valuable contribution, but it lacks the expressiveness of HLPN using data structures in tokens, which are essential in PeNTa. Nonetheless, GestIT can provide some great ideas to further improve PeNTa.

---

<sup>9</sup>D=Down, M=Move, U=Up; s=shape, b=background, a=any (alb).

**PeNTa** includes a novel approach by using HLPN (in specific, Prt Net) for multi-touch recognition, including the definition required for HLPN to work with multi-touch. PN allows formal specifications to be applied to multi-touch, and perhaps other modern input devices (e.g., Leap Motion), and enables a distributed framework while keeping the approach simpler (in comparison to low-level PNs). This means that by encapsulating complex data structures in tokens and allowing algebraic notation and function calling in the transitions of a PN, the modeling is not restricted to one individual approach. Furthermore, the data structure kept in each token maintains history information that may be useful for additional features.

### 3.3.2 HLPN: High-Level Petri Nets and IRML

**PeNTa** is defined using HLPN [59, 60, 128], consisting of the Prt Net definition [59]. The definition is based on the Input Recognition Modeling Language (IRML) specification by this author (see Appendix E). The model is defined as  $HLPN = (N, \Sigma, \lambda)$ . This contains the Net  $\mathbf{N}$ , the specifications  $\Sigma$  and the inscription  $\lambda$ .

The Net  $\mathbf{N}$  is formed with places  $\mathbf{P}$ , transitions  $\mathbf{T}$ , and connecting arc expressions (as functions  $\mathbf{F}$ ). In other words, a PN is defined as a three-tuple  $N = (P, T, F)$ , where  $F \subset (P \times T) \cup (T \times P)$ . Petri Nets' arcs can only go from  $\mathbf{P}$  to  $\mathbf{T}$ , or  $\mathbf{T}$  to  $\mathbf{P}$ . This can be formally expressed, stating that the sets of places and transitions are disjoint,  $P \cap T = \emptyset$ . Another important characteristics of Petri Nets is that they use multi-sets<sup>10</sup> (elements that can be repeated) for the input functions,  $(I : T \rightarrow P^\infty)$  and output functions,  $(O : P \rightarrow T^\infty)$  [164].

The specification  $\Sigma$  defines the underlying representation of tokens. This is defined as  $\Sigma = (S, \Omega, \Psi)$ . The set  $S$  contains all the possible token<sup>11</sup> data types allowed in the system.

---

<sup>10</sup>Also known as Bag Theory.

<sup>11</sup>Some Petri Nets' publications may refer to tokens as "sorts".

For the particular case of multi-touch in **PeNTa**, the data type is always the same<sup>12</sup>, which is a multi-touch token **K**, as shown in Table 3.1. The set  $\Omega$  contains the token operands (e.g., plus, minus). The set  $\Psi$  defines the meaning and operations in  $\Omega$ . In other words, the set  $\Psi$  defines how a given operation (e.g., plus) is implemented. In the case of **PeNTa**, which uses Prt Net, the operations use regular math and Boolean Algebra rules. This is the default for **PeNTa** tokens, which is of signature  $(S, \Omega)$ .

The inscription  $\lambda$  defines the arc operation. This is defined as  $\lambda = (\phi, L, \rho, M_0)$ . The data definition represented by  $\phi$  is the association of each place  $p \in P$  with tokens. This means that **places** should accept only variables of a matching data type. **PeNTa** has token **K**, which represents the multi-touch structure. The inscription also has labeling **L** for the arc expressions, such as  $L(x, y) \iff (x, y) \in F$ . For example, a transition that goes from place  $B$  to transition  $4$  will be represented as  $L(B, 4)$ . The operation of a given arc (function) is defined as  $\rho = (Pre, Post)$ . These are well-defined constraint mappings associated with each arc expression, such as  $f \in F$ . The **Pre** condition allows the HLPN model to enable the function, if the Boolean constraint evaluates to true. The **Post** condition will execute the code if the function is enabled (ready to fire). Finally, the initial marking is given by  $M_0$ , which states the initial state of **PeNTa**.

### Dynamic Semantics

In order to finalize the formal definition of the HLPN used in **PeNTa**, there are some basic details about the dynamic aspects of these Prt Nets. First, markings of HLPN are mappings  $M : P \rightarrow Tokens$ . In other words, places map to tokens. Second, given a marking  $M$ , a transition  $t \in T$  is enabled at marking  $M$  iff  $Pre(t) \leq M$ . Third, given a marking  $M$ ,  $\alpha_t$  is an assignment for variables of  $t$  that satisfy its transition condition, and  $A_t$  denotes the set of all assignments. The model defines the set of all transition modes to be  $TM =$

---

<sup>12</sup>This is up to the designer. If the designer wants to create additional tokens, it is also possible.

$\{(t, \alpha_t) \mid t \in T, \alpha_t \in A_t\} \iff Pre(TM) \leq M$ . An example of this definition is a transition spanning multiple places, as shown in Figures 3.6 and 3.7 (concurrent enabling). Fourth, given a marking  $M$ , if  $t \in T$  is enabled in mode  $\alpha_t$ , firing  $t$  by a step may occur in a new marking  $M' = M - Pre(t_{\alpha_t}) + Post(t_{\alpha_t})$ ; a step is denoted by  $M[t > M']$ . In other words, this is the transition rule. Finally, an execution sequence  $M_0[t_0 > M_1[t_1 > \dots$  is either finite when the last marking is terminal (no more transitions are enabled) or infinite. The behavior of a HLPN model is the set of all execution sequences, starting from  $M_0$ .

### 3.3.3 PeNTa and Multi-Touch

A multi-touch display (capacitive or vision-based) can detect multiple finger strokes at the same time. This can be seen as a finger trace. A **trace** is generated when a finger touches down onto the surface, moves (or stays static), and is eventually lifted from it. Therefore, a trace is a set of touches of a continuous stroke. While it is possible to create an anomalous trace with the palm, **PeNTa** takes into consideration only normal multi-finger interaction. However, the data structure (explained in detail later) could be modified to fit different needs, including multiple users and other sensors that may enhance the touch interaction. Given a set of traces, one can define a **gesture**. For example, a simple gesture may be two fingers moving on the same path, creating a **swipe**. If they are moving in opposite ways (at least one of the fingers), this can be called a **zoom out** gesture. If the fingers are moving towards each other, then this is a **zoom in** gesture. A final assumption that **PeNTa** makes for multi-touch systems is the following: if a touch interaction is not moving, it will not create additional samples but increment the holding time of the finger. Note that this is not inherently true in all multi-touch systems. For example, in native WINAPI (Windows 7) development, samples are generated, even if the finger is not moving, but holding. To adjust this characteristics of the WINAPI to PeNTa assumption, the samples are just filtered

Table 3.1: Multi-Touch Data Structure

Name	Description
<b>id</b>	Unique Multi-Touch Identification
<b>tid</b>	Touch Entry Number
<b>x</b>	X display coordinate
<b>y</b>	Y display coordinate
<b>state</b>	Touch states (DOWN, MOVE, UP)
<b>prev</b>	Previous sample
<b>get(Time t)</b>	Get previous sample at time <b>t</b>
<b>tSize</b>	Size of sample buffer
<b>holdTime</b>	How many milliseconds have spawn since last rest
<b>msg</b>	String variable for messages

by creating a small threshold that defines the following: If the finger is not moving or if it is moving slightly, a counter is incremented.

### 3.3.4 Arc Expressions

Each arc is defined as a function **F**, which is divided into two subsets of inputs **I** and outputs **O**, such that  $F = I \cup O$ . In the inscription  $\rho$  of this HLPN, the arc expression is defined as **Pre** and **Post** conditions. Simply put, the **Pre** condition either enables or disables the function, and the **Post** condition updates and executes call-back events, in the HLPN model.

Each function **F** is defined as  $F = Pre \cup Post$ , forming a four-tuple  $F = (B, U, C, R)$ , where **B** and **U** are part of the **Pre** conditions, and **C** and **R** are part of the **Post** conditions. **B** is the Boolean condition that evaluates to true or false, **R** is the *priority function*, **C** is the call-back event, and **U** is the update function.

The Boolean condition **B** allows the function to be evaluated using standard Boolean operators with the tokens, in C++ style (e.g.,  $T_1.state == STATE.UP$ ). If no condition is assigned, the default is *true*. The priority function **R** instantiates a code block, with the purpose of assigning a priority value to the arc expression. The call-back event **C** allows the



Petri Net to have a function callback with conditional *if* statements, local variable assignments, and external function calling. This can help to determine which function should be called. If no callback event is provided, then a default *genericCallback(Place p, Token t)* is called. The update function **U** is designed to obtain the next touch sample using *update( $T_1$ )*, or setting a value to the current sample of a given token using *set( $T_1$ , TYPE.STATE, STATE.UP)*.

Places and transitions in **PeNTa** have special properties. This is true for **P**, which has three types: *initial*, *regular*, and *final*. This allows the system to know where tokens will be placed when they are created (*initial*) and where the tokens will be located when they will be removed (*final*).

Picking the next transition or place for the case when there is one input and one output is trivial (the next **T** or **P** is the only available one). However, when there are multiple inputs or outputs, picking the next one to check becomes important in a PN implementation [140]. The “picking” algorithm used in **PeNTa** is a modified version of the one by Mortensen [140]. The algorithm combines the random nature found in other CPN [98] selection and the use of *priority functions*. The algorithm sorts the neighboring **P** or **T** by ascending value, given by the priority function **R**, and groups them by equivalent values, (e.g.,  $G_1 = 10, 10, 10$ ,  $G_2 = 1, 1$ ). The last **P** or **T** fired may be given a higher value if found within the highest group. Within each group, the next **P** or **T** is selected randomly. Also note that the algorithm presented here is just one of many possible ones. Given the flexibility of Petri Nets and the amount of work available in this field, the developer may wish to modify or change the algorithm in its entirety. The important part of the algorithm is how to pick the next transition, doing so in a way that does not violate PNs or Prt Nets.

### **Tokens and the Structure**

A powerful feature of HLPNs is their discrete markings, called **Tokens**. This feature allows the marking of different states and the execution of the PN. When tokens go through

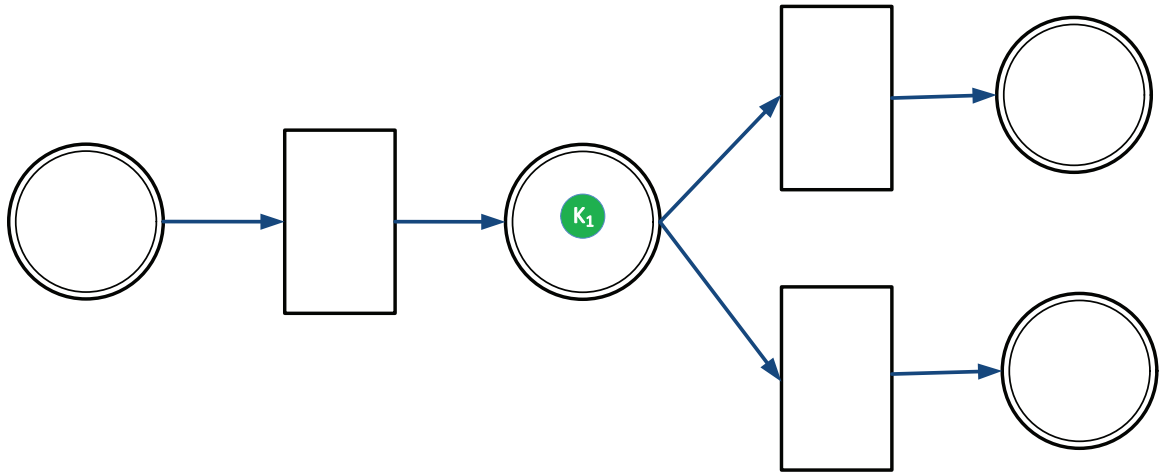


Figure 3.6: Parallel PN: State 1

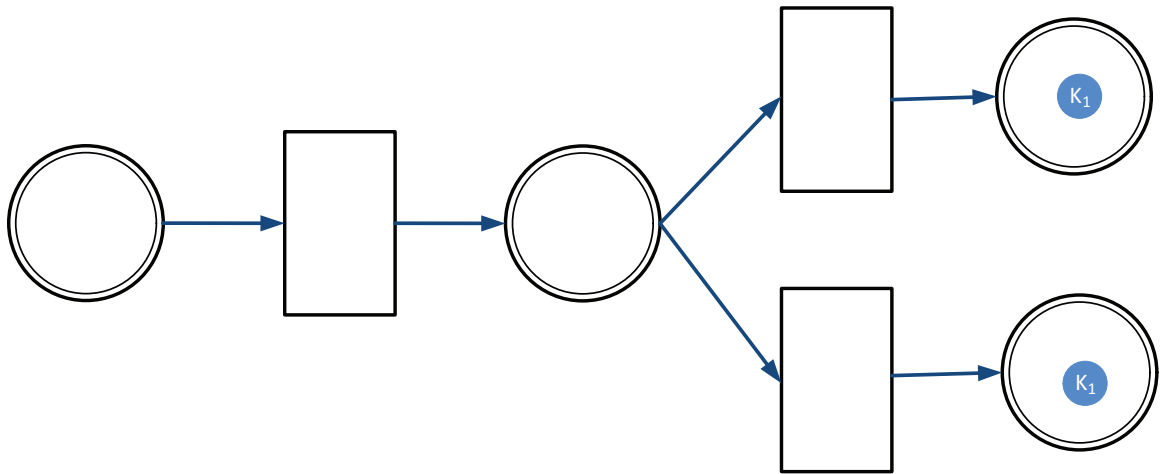


Figure 3.7: Parallel PN: State 2

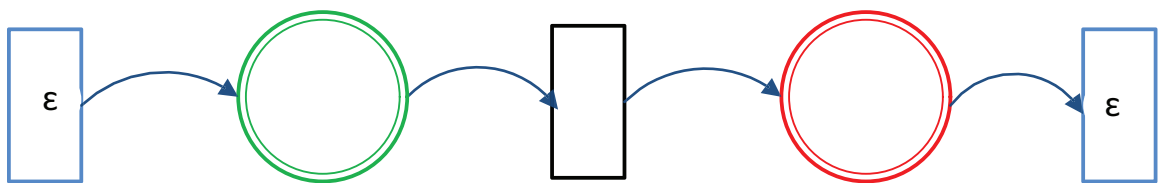


Figure 3.8: Cold transitions (Entry and Exit)

an input function **I** or output functions **O**, they are consumed. For **PeNTa**'s particular modeling requirements, the token is a representation of a data structure that defines a **trace**, as shown in Table 3.1. This data structure contains the current sample and a buffer of previous samples (**touches**).

When tokens are consumed into a transition, the **Post** condition creates a new token. If the desired effect is of a parallel system, as shown in Figure 3.6, then a transition can create  $N$  number of tokens based on the original one. In Figure 3.7, token  $K_1$  is cloned into two identical tokens. To represent the new tokens, different colors were chosen in Figures 3.6 and 3.7.

The only token data type for **PeNTa** is a multi-touch structure, type **K**, as shown in Table 3.1. The identification given by the system is denoted as **id**. Depending on the system, this may be a unique number while the process is running (long integer type), or as a consecutive integer, starting from  $1 \dots m$ , lasting through the duration of the gesture performed by the user. The latter definition is also contained in the variable **tid**. Display coordinates are given by **x** and **y**. The **state** variable represents the current mode given by DOWN, MOVE, and UP. The **holdTime** helps to determine how many milliseconds have lapsed, if the user has not moved from his or her current position on the display. This data structure assumes that a new sample only happens when the user has moved beyond a threshold. Finally, this data structure acts as a pointer to previous history touches in a buffer of size  $\eta$ . Therefore, the buffer (if it exists) can be accessed with the function  $get(Time \iota)$  or the previous sample with the variable **prev**.

In HLPNs, at least one initial Marking  $M_0$  needs to be known, which is the initial state of **PeNTa**. In the case of simulations (which is discussed later), this is not a problem. For the case of real-time execution, the initial marking must have empty tokens. This is solved by the concept of hot and cold transitions [174], represented by  $\epsilon$ . A hot transition does not

Table 3.2: Transitions

Arc	From	To	Condition	Token Count
1	A	Down	K.state == DOWN	1
2	Down	B	update(T)	1
3	B	Move	K.state == MOVE	1
4	B	UP	K.state == UP	1
5	Move	C	update(T)	1
6	C	UP	K.state == UP	1
7	C	Move	K.state == MOVE	1
8	C	Move	update(T)	1
9	C	Zoom	K.state == MOVE && IsZoom( $K_1, K_2$ )	2
10	Zoom	C	Update( $K_1, K_2$ )	2
14	Swipe	C	K.state == MOVE && IsSwipe( $K_1, K_2$ )	2
15	Swipe	D	Update( $K_1, K_2$ )	2
17	UP	E	K.state == UP	1
18	UP	E	K.state == UP	1
19	E	Terminate	true	1

require user input. A cold transition requires user (or external system) input. Therefore, in **PeNTa**, the model defines entry and exit cold transitions, as shown in Figure 3.8.

### 3.3.5 A Tour of PeNTa

A tour of **PeNTa** is needed to better explain how it works. Take for example, an interaction that has two possible gestures using two fingers: swipe and zoom. A swipe implies that the user moves two fingers in any direction. Therefore, the callback required is always the same, which is a function that reports the direction of the swipe. In the case of the zoom, zoom-in and zoom-out could be modeled separately or together. In the case of the example shown in Figure 3.9, zoom is configured as one entity.

The example shown in Figure 3.9 is created for two-finger gestures. The figure has places, arcs, transitions, and two tokens ( $K_1, K_2$ ), representing two active traces in place **C**.

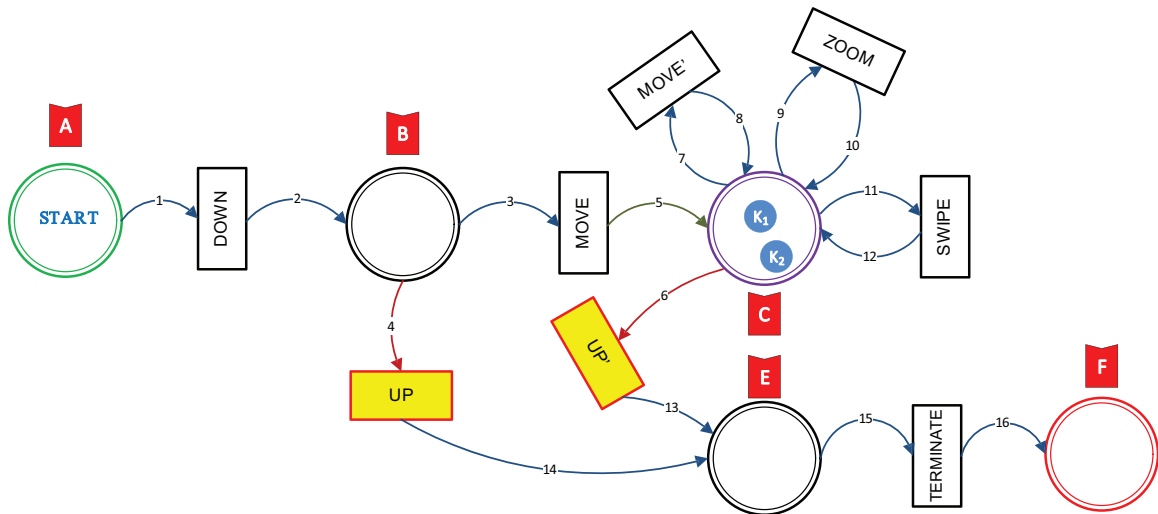


Figure 3.9: Multiple Gestures in PeNTa

For this particular example, Figure 3.9 has additional graphical representations, which are letter labels in places, numbers in arcs, and transitions with names. This is done to make the example easier to follow. However, those additional items in the graph are not part of the actual Prt Net. In addition, Table 3.2 shows each arc expression with their Boolean condition and the tokens that are required to fire (e.g, two tokens). The system starts with a empty initial marking (no tokens), while it waits for the user input. Once the user touches down onto the surface, tokens are created (using cold transitions) and placed in **START**. Given that the tokens will start with a **DOWN** state, they will move from place **A** into place **C**, using transitions 1 and 2. The first arc just consumes the token, and arc 2 updates the token with the next touch data sample into place **B**. Once in place **B**, since the token was updated with the touch sample, **PeNTa** infers the next transition using the constrained provided. It has two options, either arc 3 or arc 4. Assuming that the state is **MOVE**, now each token is moved into place **C** with an updated touch data sample. Now, we are at the point shown in Figure 3.9. **PeNTa** infers the next step. This is where the picking algorithm explained earlier comes into play. For this example, **MOVE'**, **ZOOM**, **SWIPE**, and **UP** have priority function values 1, 10, 10, and 2, respectively. This means the group with

**ZOOM** and **SWIPE** are the first to be checked for constraints, since they have the highest values. **PeNTa** will randomly pick either one and check if it can be enabled (fired) based on the Boolean condition. Assume, for example, that it picks **SWIPE** and the Boolean condition is *true*. It is *true* if two tokens are in place **C**, both tokens are in state **MOVE**, and the function *isSwipe* is true, which is an external C++ method. If the value is true, then it will call back a swipe function, passing a copy of the token data, and then update it to the next sample via arc 12. This finally brings back both tokens into place **C**. At some point, the tokens will have the **UP** state, and they will move to place **E** and place **F**.

While the example presented in Figure 3.9 shows the interaction model in a single PN for various gestures, it is possible to create individual PNs, as shown in Figure 3.10, and combine them in a set of PNs. For example, individual PNs ( $PN_i$ ) can form a model  $P = (PN_1, PN_2, PN_3, \dots, PN_n)$ , which, once constructed, can be executed. Each  $PN_i$  can run in parallel and disabled itself when the corresponding condition is not met.

### 3.3.6 Simulation and Execution

PNs could be used for analysis (e.g., Linear Temporal Logic), simulations and execution. However, the initial motivation for **PeNTa** was simulation and execution. Nevertheless, analysis could be performed if needed, but it is beyond the scope of this dissertation.

There are different ways to simulate **PeNTa**. Non-user-generated data could be used for the simulation, providing an initial marking with a history of the possible samples. Another option it is to record the user interaction, creating tokens and a buffer to feed those tokens. There are multiple ways to go about this, but two ways are very common: store the data in a transactional database or in plain text files. The former can be very useful if the set is large and different conditions may need to be applied.

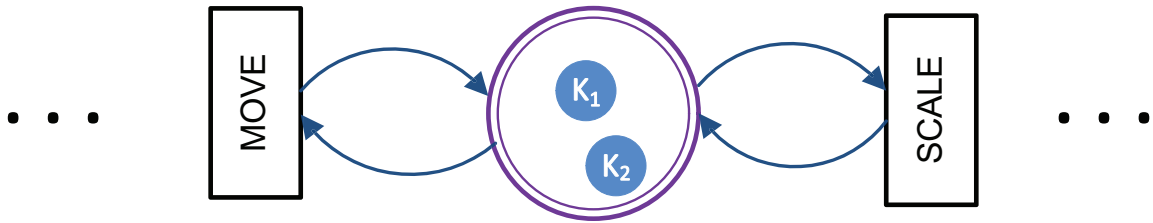


Figure 3.10: Partial Petri Net for Scale

Execution is the primary purpose of **PeNTa**. Given a well-defined model, this can run in real-time, using **PeNTa**, which has already been defined. As stated before, **PeNTa** needs additional entry and exit **cold** transitions, if there are additional inputs involved.

### 3.3.7 Overview

In summary, **PeNTa** provides a novel way to model multi-touch interactions (and possibly other modern input devices) with the use of a well-established mathematical and graphical model called Petri Nets. In specific, this approach uses HLPN called PrT Nets. **PeNTa** works under the model of IRML, which is the language definition created by this author. The IRML specification used in **PeNTa** can be found in Appendix E. For an expanded discussion, see [159, 160].

## 3.4 Yield: Removing ambiguity

When performing gesture detection, the number of gestures needed to be recognized increases the difficulty of selecting the correct gesture. Also, if the movements are faster than average and the gestures are constantly changing, which can be the case for a 3D navigation task, the process of gesture recognition becomes more challenging. An example of conflicting gestures being detected may be found in some of the sets of results generated by the well-known commercial software GestureWorks Core (GWC) API that allows

users to detect different types of gestures. GWC, developed by *ideum*<sup>13</sup>, claims to have 300 pre-built gestures, which include stroke gestures.

GWC has an XML file, where users can define gestures using the GML definition. It is possible to control filters and properties for a gesture. However, in most cases, it is not possible to have only one gesture fire when using GWC. Note that this is not a unique problem to GWC. FETOUCH++, given a set of additional gestures, may also produce ambiguity when recognizing. This is actually normal in many types of recognition systems.

The approach taken to remove ambiguity and select the right gesture from the initial set of results has been tested using Microsoft Visual Studio 2012 (using C++<sup>14</sup> language) running on a Windows 7 platform with a 3M M2256PW multi-touch display. In addition, GWC<sup>15</sup> API was used for the multi-touch gesture recognition. OGRE3D was used for the 3D rendering.

### **3.4.1 Yield: How it Works**

**Yield** is divided into sub-modules (inner classes) that work in conjunction to detect the correct gesture with one main module (outer class) that contains additional help to wrap the approach of removing ambiguity in gesture detection.

#### **Touch Gesture**

The main class, called *TouchGesturePad*, contains all the inner classes, which are described next, as well as important variables, data structures, and functions, that are required for the functioning of Yanked Ambiguous Gestures (Yield).

---

<sup>13</sup>In part with grants from National Science Foundation, grant #1010028.

<sup>14</sup>C++11.

<sup>15</sup>Version: early 2014.



First is *gesture\_classification* (**gc**) that contains all the possible gestures available in GWC. Ideally, this should be loaded from a settings file, as opposed to hard-coding it in the class. The classification shown in Listing 3.9 is a reflection of the gesture list that was available from gesture works, which included specific trace count in some, such as *finger\_drag\_1*, and others with variable trace count, such as *scale\_n*. This classification includes states of no recognition. Finally, **gc** contains for each gesture state a string counterpart. This is because the names in GML are stored as strings. The **gc** data structure is the primary link for all the inner classes.

Listing 3.9: Yield (gc)

```
1 typedef enum gesture_classification {
2   drag_n, rotate_n, rotate_n_4, scale_n,
3   drag_inertia_n, rotate_inertia_n,
4   scale_inertia_n, what_drag_n,
5   drag_x_n, finger_drag_1, finger_drag_2,
6   finger_drag_3, finger_drag_4,
7   rotate_noise_filter_n, finger_rotate_2,
8   finger_rotate_3, finger_rotate_4,
9   finger_scale_2, finger_scale_3,
10  finger_scale_4, finger_scale_5,
11  point_power_drag_2, rotate_to_scale_n,
12  manipulate_n, manipulate_inertia_n,
13  manipulate_inertia_boudary,
14  flick_n, swipe_n, scroll_n,
15  finger_tilt_2, finger_tilt_3,
16  finger_tilt_4, finger_tilt_n,
17  finger_pivot_1,
18  finger_pivot_inertia_1,
19  finger_orient_5, hold_n, tap_n,
20  double_tap_n, triple_tap_n,
21  finger_hold_3, finger_tap_3,
22  finger_double_tap_1,
23  finger_triple_tap_1,
24  unknown_gesture, gesture_not_detected,
25  gesture_not_found, gesture_not_set,
26  Unknown
27 };
```

## Gesture Criteria

The *GestureCriteria* class helps with the definition of certain features that allows the **Yield** algorithm to select the correct gesture. The two primary features used were **criteria** and **strength**, required for each gesture. The other features, **adjacency**, **strength total**, and **frequency**, were tested but they were not found useful when using GWC. It is important to note that all the features were configured using C++11 lambdas<sup>16</sup>, which allows the definition of functions. The fact that the features that help with the selection of the correct gesture are functions gives the designer more flexibility. as opposed to defining actual values. In other words, all the features were functions that could be defined by the developer. Nevertheless, default functions are provided in the implementation. This is mentioned for two reasons: First, features were not obtained with just a threshold but as a function of some value(s). Second, it was mentioned to remind the interested readers that when looking at the listings, the lambdas take functions and not values. Of course, a function could be set to be constant, as  $F(X) = 5$ . The description of each of the features for the *GestureCriteria* class is defined next.

The **strength** function takes a map (<string,double<sup>17</sup>>). The actual input can vary depending on the framework, but in the case of GWC, the data given by each gesture varies and it is provided inside of a map data structure. For example, the rotate gesture only has one key value for the rotation angle, while the drag gesture contains two key values for the X and Y coordinates. The default strength function that is provided is the sum of all the values. However, there are cases in which the developer may need to adjust it. The **criteria** is the function of the **strength** value, with the default function as  $F(x) = 1.0 * x$ . In other words, it is the weight factor for the **strength**.

---

<sup>16</sup>A functional language by default have first-class functions.

<sup>17</sup>It may also use float type.

The additional features were not used in GWC, but they are important to mention, as they can be helpful in other instances. First, **adjacency** is a ranking function that determines, based on a the previous recognition, how apart each gesture is from the previous list. For example, if a rotate gesture was the highest ranking gesture for the previous time frame, and rotate now is the second highest ranking, the adjacency is -1. If the case is reversed, the adjacency would be +1. Nevertheless, given that this is a function, the developer can modify its meaning and usage. This particular gesture was not helpful when tested because it did not prove to make a significant difference with GWC. Second, **strength total** is a function that takes the new strength and the total strength up to that point, to calculate the strength total. While this value is easy to calcualte over time, it did not prove to be highly beneficial for the API used to test **Yield**. Finally, **frequency** is the function of previous gestures triggered, which counts the frequency over time that the gesture was fired (or in a top quartile of possible ones). The frequency over time did not prove to be needed when recognizing gestures in GWC.

## **Gesture Type**

The *GestureType* class encapsulates the *GestureCriteria* class and includes additional features that help with the gesture detection. This class defines the type of gesture. First, by storing the criteria class, the gesture type now has access to it. The reason to keep the classes separated is because *GestureCriteria* can be shared across multiple *GestureType* objects. For example, all types of drag gestures may share one common criteria class. Second, the **inception values** are a structure that holds different types of factors used during the time the gesture is active. This is different from the values stored in the criteria class, since it generated (via functions) values to determine the gesture to select. The **inception values** contain rate of decay, noise factors, and threshold, which will be explained more in detail later. Third, the *GestureType* class contains a structure called **gain**, which is used to

modify the weights (if needed) from the criteria class. This is very useful when the designer needs to share a *GestureCriteria* between different *GestureType* but may need to change the weight factor to a specific parameter. This includes the **criteria gain** and **strength gain**. Finally, the *GestureType* class contains factor values for the navigation operation, called **controller values**.

The **inception values** contain important factors that are used during the time the gesture is active and alive. First, the **threshold** determines what is the minimum strength value for a gesture to be considered a candidate. Second, the **decay value**<sup>18</sup> indicates when an active gesture is considered to be in the process of decay. This information can be used to make decisions about the other candidates. Finally, a group of three noise factor values are used to minimize noise or user error, which is the minimum amount of strength that a current gesture may have. The **noise factor x** and **noise factor y** indicate the minimum requirement for the gesture to be sent to the navigation controller, and the **noise dual factor** indicates the minimum requirement for both axes to be sent to the navigation controller. All of the inception values can be use in different ways by the developer. Later in this section, a more concrete algorithm and an example will be given.

## **Gesture**

The *Gesture* class encapsulates the *GestureType* to help with the active gesture when selected and to use a main container for the criteria, since *GestureType* encapsulates *GestureCriteria*. In addition, the class contains the GWC event data, including gesture and touch points. Also, the class contains some important methods required for the **Yield** to work. The most important functions are listed below:

- **processGesture**: This method calculates the values for the candidate gesture using the *GestureType*. It also uses the **gain values** to determine the weight required.

---

<sup>18</sup>Refer to as MaxRateOfDecrease in the C++ code.

- **refreshCriteria:** This method modifies the current criteria value and calls the compute method.
- **compute:** This computes the criteria value multiply by the gain value.
- **getTraces:** This gets the trace counts.
- **setTraces:** This sets the trace count.

In addition to concept of gesture, this class contains a definition for an inner gesture. An **inner gesture** is defined as a subset of the main gesture. In other words, an inner gesture may be a drag going up and down, and the complete gesture is the drag going in any direction. This is very useful for navigation. **Yield** used the inner gesture classes for some drag gestures.

### 3.4.2 Yield: The Algorithm

---

#### Algorithm 3.5 Yield: Fetch Points

---

**Require:** multi-touch device is connected and data queue exists.

**Ensure:** Return  $\emptyset$  if no touch points.

- 1: *pointEvents*  $\leftarrow$  **consumePointEvents**() // In the case of GWC: Assign points to Display Object.
  - 2: **return** *pointEvents*
- 

**Yield** is better explained by illustrating the core concepts using pseudo-code. Note that some of the variable names have been shortened in this algorithm compared to their counterparts in the concrete implementation. The primary objective is to select the correct gesture candidate. Once the gesture is selected, the action for a possible 3D navigation task is secondary.

While **Yield** was tested with GWC, the algorithm is agnostic to the actual gesture detection method or algorithm used. Nevertheless, it makes some assumptions that could be easily modified to meet other needs. The main assumption is that the points and gestures

fired by the primary detection algorithm are stored in a queue until a consumption method is called to retrieve the current data events. Of course, this process is designed for detection algorithms that output multiple detection candidates. Another assumption made in the algorithms presented in here is that external methods accomplish their expected goals correctly. An example of this is that the trace count (how many fingers are interacting with the display) are set in the *Gesture* class. This is omitted in the algorithm part of *Yield*. Nevertheless, the implementation is not omitted. Finally, it is assumed that a main **event loop** will trigger the process every *n* milliseconds. In the actual implementation, the process has knowledge of the time elapsed between one call and the next one (time elapsed between two event loops), and the cycle count, which indicated (using 64 bits unsigned long) the current cycle.

The first part of **Yield** is to retrieve the points, in the function called **Fetch Points**, shown in Algorithm 3.5. The points are retrieved by calling **ConsumePointEvents**. If no touch points are found, which is possible, the algorithm will return the empty set<sup>19</sup> ( $\emptyset$ ).

The second part of **Yield** is to **Fetch Gestures** using Algorithm 3.6. First, in line 1 of the algorithm, **Fetch Points** is called, passing the active multi-touch device. If there are no points, as stated before, there is nothing else to do, therefore, it returns the empty set ( $\emptyset$ ). Then, in line 4, the **ConsumeGestureEvents** method is called to retrieve all event data stored in its corresponding queue. In lines 5-8, the force conditions are checked and reseted if needed. The force conditions are *this.noGestureCount* and *this.traceChanged*. The first one, **NoGestureCount**, is a counter that keeps a record of how many times a gesture was not retrieved when the method **ConsumeGestureEvents** was called and returned the empty set ( $\emptyset$ ). The **traceChanged** keeps count of how many times the number of fingers placed on the display are changed during an active gesture (the current gesture selected by **Yield**). Therefore, the **forceConditions** method checks if either of the two force variable counters

---

<sup>19</sup>In C++: NULL or nullptr.

---

**Algorithm 3.6** Yield: Fetch Gestures

---

**Require:** Gesture recognition algorithm with data queue.

*this.gestureState* starts as idle.

**Ensure:** Return  $\emptyset$  if no gestures or no points. Otherwise, gestures data set.

```
1: points  $\leftarrow$  FetchPoints(mtDevice)
2: if points =  $\emptyset$  then
3:   return  $\emptyset$ 
4: gesture  $\leftarrow$  consumeGestureEvents()
5: force  $\leftarrow$  false
6: if gestures  $\neq$   $\emptyset$  and forceConditions() then
7:   force  $\leftarrow$  true
8:   resetForceConditions()
9: if gestures.size() = 0 or force = true then
10:  WidgetActions(points) // Perform additional actions (e.g., Buttons)
11:  this.gestureState  $\leftarrow$  nogesture
12:  increment(this.noGestureCount)
13:  resetGesture()
14:  return  $\emptyset$ 
15: return gestures
```

---

are true. The **resetForceConditions** method is used to reset each counter independently of each other. Each counter has a maximum threshold assigned. If the current count is greater, then the force condition is set to zero. At the end, if the if statements in line 11 is not executed, then this algorithm will return the gesture data. On the other hand, if the if statement where either **force** is true or the *gesture.size*() is equal to zero, then a few steps are taken before returning the empty set ( $\emptyset$ ). In this case, the algorithm changes the current state of recognition, it increments the count of *this.noGestureCount*, and it resets the gesture. This last step is important to expand. The **Reset Gesture** method, shown in Algorithm 3.7, changes the recognition state of the gesture and sets the empty set ( $\emptyset$ ) to *this.detectedGesture*. The selected gesture by **Yield** is stored in **detectedGesture**. Note that the reference to *this* makes reference to the outer class of **Yield** already explained.

Once the points and gesture fetchers are executed, then the gesture is processed, in Algorithm 3.8, called **Process Gesture**. This process calls **Fetch Gesture**, which in turn, calls **Fetch Points**. At this point, if no gestures are found, the process returns and no further

---

**Algorithm 3.7** Yield: Reset Active Gesture

---

1:  $this.classificationType \leftarrow \mathbf{gesture\_not\_set}$   
2:  $this.detectedGesture \leftarrow \emptyset$   
3:  $this.isNewGesture \leftarrow \mathbf{true}$

---

action is taken, as shown in lines 2-3. Since the **gesture** data structure may contain more than one gesture (which was always the case with GWC), a for loop to process them and find the correct gesture is executed between lines 6-32. While it is possible that the loop will run for each input, in many instances, the loop will exit before it is complete, as will be described next. Algorithms 3.8 and 3.9 require two additional data structures: a hash map data structure and a priority queue data structure. The map, named **gestureMap**, stores each gesture processed, which can be accessed by using the key. In other words, the key for this map is the **gesture classification (gc)**, described in Listing 3.9. Simply put, the **gc** is what identifies the gesture. The value of the map is the *Gesture* class. The other data structure used is a priority queue, also described in Chapter 2. The priority queue holds *GestureType* objects in descending order by criteria value. Later in this section, some of the details of the actual implementation will be discussed.

The bulk of the process happens during the only loop found in Algorithm 3.8. The running time of the loop is  $O(N \lg N)$ . The size of **N** is very small. It depends in the amount of gestures. the common amount will be between 5 to 20 gestures. As stated previously, the loop does not always run for each input, except when there is no active gesture. In testing this algorithm, there was no indication of excessive delay, therefore, the performance of **Yield** was acceptable by all users. The  $\lg(N)$  is added to the running time because of the priority queue in line 32.

The loop is the core of detection and update after detection for **Yield**. When the loop starts, the identification of the gesture given by the detection algorithm is converted to a **gesture classification**. In line 8, the algorithm checks to see if there is an active gesture (*this.detectedGesture*), and if the current gesture **g** is not the same type as the **detect-**



**edGesture**, then the process goes back to the top of the loop, as shown in line 9. Then, in line 10, the tuple containing a Boolean value and a *GestureType* class is formed using the classification **gt** and the trace count (the designer can configure either one *GestureType* class for all drag gestures or can define different types for the corresponding trace count). If *gType.found* is false, then there is no point to continue. This is one of the cases for which the designer may not have designed a default *GestureType* class. It is up to the developer or designer how to handle this. In the implementation tested, we reported the problem if found. A possible solution is to create a default *GestureType* for any gesture that does not have a definition. It was not the decision that this author made, but it is a perfectly fine design choice if needed. Finally, using the tuple, the algorithm creates an object called **cGesture**, which contains the current gesture to be evaluated. This gesture is processed by calling **processGesture**. This method will be explained later, but for now, suffice it to say that it updates the criteria values using the function features described earlier.

Up to this point, most of the work was housekeeping<sup>20</sup>; however, the algorithm has three possible cases: First, in line 15, the algorithm decides if there is an active gesture, and if the type of current active gesture (*this.detectedGesture*) is the same *GestureType* as the gesture being evaluated (*cGesture*), lines 16-27 are executed. The second case simply states that if this is not case 1, and *cGesture.criteria* is less than *cGesture.type.getThreshold()*, then it must go back to the top of the for loop. The reason is that the criteria value for this gesture is below the minimum requirement assigned by the designer. Finally, case 3 (shown in lines 31-32) indicates that gesture must be saved to be processed later, by the **preProcessAction** method.

The first case must be expanded to understand the few sub cases found. In all three sub-cases (if statements), if the condition is true, this algorithm will exit. The first sub case identifies that trace count has changed and it increments one of the force conditions, called

---

<sup>20</sup>Making small decisions and changing states.

*this.traceChanged*. The second sub-case checks if the gesture has died using a minimum threshold (**minTh**), usually set to 0.01. The third sub-case checks if the current gesture has an inner gesture. If it has an inner gesture, and the inner gesture is the same as the current one, then it exits. Finally, if none of the sub-cases are true, then lines 25-27 are executed. In this part of the code, the gesture state is changed, the **processGestureAction** is fired, and then the function exits. The reason is that **processGestureAction** is directly linked to the action required to be taken by the controller (e.g., 3D navigation system). Therefore, there is no need to keep checking. It is important to note that if case 2 or case 3 are triggered, it is possible that the for loop may end (it is not always the case), causing the **preProcessAction** method to be fired.

Algorithm 3.8 may fire **preProcessAction** (see Algorithm 3.9). This algorithm is fired only when there is a new gesture to be detected because of ambiguity. First, if there is no new gesture (*gestureMap.size()* = 0) and there is an active gesture (*this.detectedGesture*), then **processGestureAction** is fired. There are two important facts to be considered here. One is that the probability of this occurring is very low because gestures will decay and force the process to have a new gesture check, as Algorithm 3.8 describes. The second fact is that this can cause a non-desired effect to some developers (but it was intended for the Hold-and-Roll gesture). This is because a known gesture, may behave differently if lines 2-6 are present in the algorithm. This is because the code is telling the system to process the previous detected gesture when there is no gesture available. This is only true for a small amount of time because the gesture will decay. This worked well for the Hold-and-Roll gesture, which is described in Section 3.6. If this causes any undesired behavior, an alternate algorithm is provided, as Algorithm 3.10. The second part of the algorithm does the actual selection of the correct gesture. Lines 8-15 show the last part of **Yield**, as far as selecting the correct gesture is concerned. The gesture with the highest criteria is popped

---

**Algorithm 3.8** Yield: Process Candidate

---

**Require:**  $gestureMap.size() = pqGestures.size() = 0$

- 1:  $gestures \leftarrow \mathbf{FetchGestures}(mtDevice)$
- 2: **if**  $gestures = \emptyset$  **then**
- 3:   **return**
- 4: **if**  $this.isNewGesture = \mathbf{true}$  **then**
- 5:    $this.isNewGesture \leftarrow \mathbf{false}$
- 6: **for**  $g$  in  $gestures$  **do**
- 7:    $gt \leftarrow \mathbf{getClassification}(g.id)$
- 8:   **if**  $this.detectedGesture \neq \emptyset$  **and**  $this.detectedGesture.type \neq gt$  **then**
- 9:     **continue**
- 10:    $gType \leftarrow \mathbf{getGestureType}(gt, g.traceCount)$
- 11:   **if**  $gType.found = \mathbf{false}$  **then**
- 12:     **continue**
- 13:    $cGesture \leftarrow \mathbf{Gesture}(g.id, g, gType.type)$  // Traces must be set.
- 14:    $cGesture.\mathbf{processGesture}()$
- 15:   **if**  $this.detectedGesture \neq \emptyset$  **and**  $this.detectedGesture.type = cGesture.type$  **then**
- 16:     **if**  $cGesture.traceCount \neq this.detectedGesture.traceCount$  **then**
- 17:        $\mathbf{increment}(this.traceChanged)$
- 18:       **return**
- 19:     **if**  $cGesture.criteria < minTh$  **then**
- 20:        $\mathbf{resetGesture}()$
- 21:        $this.gestureState \leftarrow \mathbf{gesture\_detected\_changed}$
- 22:       **return**
- 23:     **if**  $\mathbf{checkInnerGesture}(this.detectedGesture, cGesture) = \mathbf{false}$  **then**
- 24:       **return**
- 25:      $this.gestureState \leftarrow \mathbf{gesture\_detected\_changed}$
- 26:      $\mathbf{processGestureAction}(cGesture)$
- 27:     **return**
- 28:     **else if**  $cGesture.criteria < cGesture.type.\mathbf{getThreshold}()$  **then**
- 29:       **continue**
- 30:     **else**
- 31:        $gestureMap[gt] \leftarrow cGesture$
- 32:        $pqGestures.\mathbf{push}(cGesture)$
- 33:      $\mathbf{preProcessAction}(gestureMap, pqGestures)$
- 34:     **return**

---

into **newGt**. If there is no detected gesture, **newGt** is used to update *this.detectedGesture*. Finally, `processGestureAction` is fired using **newGt**.

---

**Algorithm 3.9** Yield: Pre-Process Action

---

**Require:**  $gestureMap.size() \neq pqGestures.size() \neq \emptyset$

- 1: **if**  $gestureMap.size() = 0$  **then**
- 2:   **if**  $this.detectedGesture \neq \emptyset$  **then**
- 3:      $nGest \leftarrow \mathbf{Gesture}(this.detectedGesture)$
- 4:      $this.gestureState \leftarrow \mathbf{gesture\_detected\_same}$
- 5:      $\mathbf{processGestureAction}(nGest)$
- 6:      $this.newGesture \leftarrow \mathbf{false}$
- 7:   **return**
- 8:  $newGt \leftarrow pqGesture.top()$
- 9:  $pqGesture.pop()$
- 10: **if**  $this.detectedGesture = \emptyset$  **then**
- 11:    $this.detectedGesture \leftarrow \mathbf{Gesture}(newGt)$
- 12:    $this.gestureState \leftarrow \mathbf{gesture\_detected\_new}$
- 13:    $\mathbf{processGestureAction}(newGt)$
- 14:    $this.isNewGesture \leftarrow \mathbf{false}$
- 15: **return**

---



---

**Algorithm 3.10** Yield: Pre-Process Action (alt)

---

**Require:**  $gestureMap.size() \neq pqGestures.size() \neq \emptyset$

- 1: **if**  $gestureMap.size() = 0$  **then**
- 2:   **return**
- 3:  $newGt \leftarrow pqGesture.top()$
- 4:  $pqGesture.pop()$
- 5: **if**  $this.detectedGesture = \emptyset$  **then**
- 6:    $this.detectedGesture \leftarrow \mathbf{Gesture}(newGt)$
- 7:    $this.gestureState \leftarrow \mathbf{gesture\_detected\_new}$
- 8:    $\mathbf{processGestureAction}(newGt)$
- 9:    $this.isNewGesture \leftarrow \mathbf{false}$
- 10: **return**

---

### 3.4.3 Yield: The Implementation

The system was tested with GWC. There was no need to thread the process, therefore, it wasn't tested using a multi-threaded approach. The actual algorithms provided are not

ready to be run in parallel without some additional changes to the code. In the next chapters, there will be further discussion about the actual impact of **Yield**, as it was used in the experiment performed for 3D navigation.

In the algorithms section, the **processGestureAction** was not presented. This is because while **Yield** classes have knowledge of the navigation, the primary contribution of **Yield** is to remove ambiguity. However, in Listing 3.10, there is a basic demonstration of how the code works. This method allows for navigational tasks in the system to take place (or any other task). It is meant to be more developer-oriented. Another part that is important is the type of data structure used for the gesture map and the priority queue of gestures, which are shown in Listing 3.11. Notice that the priority queue is defined with *std::less*, which happens to be the default for a priority queue. This means that the highest value will be on top. This is not true for a regular sorted data structure (e.g., sorted set), but is the design of the priority queue. Finally, the **Gesture.processGesture** function is shown in Listing 3.12. The code uses some features of C++11<sup>21</sup>.

Listing 3.10: Yield (processActionGesture)

---

```
1 ...
2 case gt::drag_n:
3 {
4     candidateGesture = "drag_n";
5     auto fx = gesture.mGestureEvent.values["drag_dx"];
6     auto fy = gesture.mGestureEvent.values["drag_dy"];
7     ...
8     auto afx = std::abs(fx);
9     auto afy = std::abs(fy);
10    int sgnx = NMath<float>::sgn(fx);
11    int sgny = NMath<float>::sgn(fy);
12    if (gesture.getTraces() == 1)
13    {
14        ...
15        removeDualNoise(afx,afy,_move_drag1.noiseFactorX,_move_drag1.
            noiseFactorY,_move_drag1.noiseDual);
16        float sfx = sgnx * afx * s1;
17        float sfy = sgny * afy * s2;
```

---

<sup>21</sup>At least the ones available in Visual Studio 2012.

```

18     mpNavigation->applyTranslationForce(Navigation::AXIS_X_NEG,sfx,t,"
        drag2_axisX");
19     mpNavigation->applyTranslationForce(Navigation::AXIS_Y,sfy,t,"
        drag2_axisY");
20 }
21 if (gesture.getTraces() == 2)
22 {
23     if (gesture.getInnerGesture() == "Drag_Horizontal" && afy > afx)
24         break;
25     ...
26     if (afx > afy)
27     {
28         afy = 0.0f;
29         gesture.setInnerGesture("Drag_Horizontal");
30         gesture.mHasInnerGesture = true;
31     }
32     ...
33 }
34 ...
35 }
36 ...

```

Listing 3.11: Yield (Data Structures)

```

1 std::map<gesture_classification,Gesture> activeGestureMap;
2 std::priority_queue<Gesture,std::deque<Gesture>,std::less<Gesture>>
    pqGestures;
3 //operators
4 inline bool operator< (const Gesture &lhs,const Gesture &rhs)
5 {
6     return lhs.mCriteria < rhs.mCriteria;
7 }
8 inline bool operator<=(const Gesture &lhs,const Gesture &rhs)
9 {
10    return lhs.mCriteria <= rhs.mCriteria;
11 }
12 inline bool operator> (const Gesture &lhs,const Gesture &rhs)
13 {
14    return lhs.mCriteria > rhs.mCriteria;
15 }
16 inline bool operator>=(const Gesture &lhs,const Gesture &rhs)
17 {
18    return lhs.mCriteria >= rhs.mCriteria;
19 }

```

Listing 3.12: Yield (Gesture.ProcessGesture)

```

1 void Gesture::processGesture(const GestureType & gestureType)

```

```

2 {
3   this->mGestureType.computeStrength(this->mGestureEvent.values);
4   this->mGestureType.computeCriteria(this->mGestureType.mCriteriaValues.
      strength);
5   //compute globals
6   this->mCriteria = this->mGestureType.getCriteria();
7 }

```

### 3.5 FaNS: Navigational System - A Fair Approach

The central theme of this dissertation is 3D navigation with multi-touch displays. With this in mind, it was imperative that the navigation system that controlled different interfaces could handle the input in a way that would not be biased toward any of the input devices being experimented upon. There are some factors that even a navigational system could not take into consideration, which are independent considerations for each input. Nevertheless, the Fair Navigation System (FaNS) provides an experimental framework, such as 3DNav, to centralize the translation and rotation of a navigational system.

#### 3.5.1 FaNS: The Implementation

The implementation of **FaNS** consist of a few parts: the update method, the translation and rotation, the structure to hold data, and some additional helper methods, which will be described below. All the listing includes partial or complete C++11 implementation.

The update function is triggered by an event loop (e.g., game loop) for every cycle. This function, as shown in Listing 3.13, updates the queue of navigation steps sent by input devices (e.g., multi-touch). In specific, the implementation uses a list<sup>22</sup> to be able to process them in the right order. The first step is to see if the buffer needs to be reset, as shown in line 1. This is needed when the navigation is set to pause and the developer

---

<sup>22</sup>A vector in C++.

requires the buffer to be clear before resuming. Then, if the **mEnabledNavigation** guard is not true, the update function will not run at this given cycle. The reason that this is after the **mResetBuffer** is to allow the removal of all buffer data before stopping the navigation, if the developer chooses this path. During the update, the two major parts required for navigation are performed next: the translations (the linear movement over the X, Y, Z axes) and the rotations (the orientation changes about the X, Y, Z axes). The actual methods to translate or rotate are up to the designer. For example, for translation, this author used pseudo-physics linear movements, and for rotations just a basic incremental angular step to change the orientation, as shown in lines 13-27, and lines 28-36. What is important is to keep a centralized system, such as **FaNS**, such that the system provides a fair navigation platform for experiment design. Once the translation and rotations have been completed, a delete criterion should be applied. The most trivial alternative is to delete all the data queued. A more interesting approach is to delete after the movement has decayed or after a given time, if using some physics or pseudo-physics approach.

In order for **FaNS** to work, the input system needs to apply (or push) the translations and rotations. This is done by directly calling **applyTranslationForce** and **applyRotationForce**, as shown in Listings 3.14 and 3.15, respectively. The input devices can also use some of the helper methods, such as **moveLeft**, **moveDown**, **rotateZ**, **rotateReverseZ**, and so forth. An example for **rotateZ** is shown in Listing 3.16. All of the helper methods that help the navigation system to move will end up calling either **applyRotationForce** or **applyTranslationForce** methods. **FaNS** also includes a few additional methods to help with navigation, such as **setCamera** and **resetYaw**. More than the actual method, the important goal is to centralize the navigation, to be able to provide a unbiased experimental environment.

Listing 3.13: Navigation (Update)

```
1 if (mResetBuffer)
2 {
```



```

3   mResetBuffer = false;
4   mTranslationForceList.clear();
5   mRotationForceList.clear();
6 }
7 if (!mEnabledNavigation)
8     return;
9 bool hasChanged=false;
10 if (mTranslationForceList.size() != 0
11     || mRotationForceList.size() != 0)
12     hasChanged = true;
13 for (auto &ft : mTranslationForceList)
14 {
15     ft.pos = ((ft.vel_initial + ft.vel) / 2 ) * ft.t;
16     ft.vel_initial = ft.acel * ft.t;
17     ft.pos_initial = ft.pos;
18     ft.step = ft.dir.normalisedCopy() * ft.pos;
19     auto check_step = mCamera->getPosition() + ft.step;
20     const auto bound =650.0f;
21     if (bound > std::abs(check_step.x) &&
22         bound > std::abs(check_step.y) &&
23         bound > std::abs(check_step.z))
24     {
25         mCamera->moveRelative(ft.step);
26     }
27 }
28 for (auto &fr : mRotationForceList)
29 {
30     Ogre::Quaternion q;
31     q.FromAngleAxis(fr.r,fr.axis);
32     fr.rot = q;
33     fr.r *= DEFAULT_FRICTION;
34     fr.step = fr.rot;
35     mCamera->rotate(fr.step);
36 }
37 //Delete items past criteria
38 ...

```

Listing 3.14: Navigation (Push Translation)

---

```

1 ...
2 //public
3 void Navigation::applyTranslationForce(const direction& dir, const
4     translation_force& n, const navigation_time& t,
5     const navigation_type& ntype)
6 {
7     applyTranslationForce(dir,n,t,ntype,"TranslationForce");
8 }
9 //private

```

```

9 void Navigation::applyTranslationForce(const direction& dir, const
    translation_force& n, const navigation_time& t,
10         const navigation_type& ntype, const std::string &
            moveType)
11 {
12     //acceleration
13     translation_force_data f;
14     f.navType = ntype;
15     f.dir = dir;
16     f.n = n;
17     f.t = t;
18     f.vel = 0;
19     f.elapsedTime =0;
20     f.acel =0;
21     f.vel_initial = 0;
22     f.pos_initial = 0;
23     f.moveType = moveType;
24     mTranslationForceList.push_back(f);
25 }
26 ...

```

Listing 3.15: Navigation (Push Rotations)

---

```

1 ...
2 //public
3 void Navigation::applyRotationForce(const direction& axis, const
    rotation_force& r, const navigation_time& t,
4         const navigation_type& ntype)
5 {
6     applyRotationForce(axis,r,t,ntype,"RotationForce_Radian");
7 }
8 void Navigation::applyRotationForce(const direction& axis, const
    rotation_force_degrees& r, const navigation_time& t, const
    navigation_type& ntype)
9 {
10     applyRotationForce(axis,r,t,ntype,"RotationForce_Degrees");
11 }
12 void Navigation::applyRotationForce(const direction& axis, const
    rotation_force_degrees& r, const navigation_time& t, const
    navigation_type& ntype,
13         const std::string & moveType)
14 {
15     const rotation_force rd(r);
16     applyRotationForce(axis,rd,t);
17 }
18 //private
19 void Navigation::applyRotationForce(const direction& axis, const
    rotation_force& r, const navigation_time& t,

```

```

20         const navigation_type& ntype, const std::string &
           moveType)
21     {
22         rotation_force_data f;
23         f.navType = ntype;
24         f.t = t; f.vel = 1;
25         f.elapsedTime =0;
26         f.vel = 0;
27         f.acel =0;
28         f.r = r;
29         f.axis = axis;
30         f.vel_initial = 0;
31         f.pos_initial = 0;
32         f.moveType = moveType;
33         mRotationForceList.push_back(f);
34     }
35     ...

```

Listing 3.16: Navigation (Push Translation)

```

1     ...
2     void Navigation::rotateZ(const rotation_force_degrees & degree,
3         const navigation_time & t, const navigation_type& ntype)
4     {
5         direction dir(0,0,1);
6         applyRotationForce(dir, degree, t, ntype, "RotateZ-RotationForce_Degrees");
7     }
8     ...

```

### 3.6 Hold-and-Roll: Finding a Gesture for the Z Axis

In the search for a more intuitive 3D interaction, different gestures were considered. There are many possible gestures, as detailed in Chapter 2. The consideration of other gestures and the design choices for those will be explained in more detail in Chapter 4. The following section describes the **Hold-and-Roll** gesture used for the Z axis.

**Hold-and-Roll** is intended to be a bi-manual multi-touch interaction. With this said, it is possible to try to perform this gesture with one hand, as will be clear when the gesture is explained. The gesture was originally designed with the following criteria:

- It is meant to be used on a desktop display, wall display or surface display. This gesture was not meant to be used with mobile or tablet devices.
- The gesture uses the two stationary fingers with the non-dominant hand and a rolling gesture with the finger of the dominant hand.
- The gesture could be modified to have one stationary finger and one rolling finger.
- The finger from the dominant hand rolls vertically, either up and down, or down and up (in the case of the desktop display).
- It is possible to extend the rolling to horizontal and diagonal movements, if needed.
- If the user releases the rolling finger, the navigation continues while the stationary fingers are pressed. This can be modified with the following options:
  - The rolling finger can be assigned a momentum value, which will decay over time.
  - When the user is not rolling the finger, there can be a time factor to stop the movement.
- The velocity is constant while the user is not rolling the finger. However, when the user rolls it again, the navigation stops and continues, creating a small break when navigating.
- The velocity is determined with the movement of the rolling finger. Small movements will not create enough momentum to keep the automatic movement.

The gesture designed allowed for a different gesture to move forward and back (Z direction) than the typical scale gesture. Actually, the scale gesture was disabled from the experiment because it is reserved for zoom in or out or adjusting the camera's view angle. It is important to consider that moving forward and back, is not the same as zooming in or out of the virtual world. It may appear to be so, but they are two different actions.

A simple example is to stay stationary at the end of one street. If one uses binoculars, it can come toward and forward with the vision. However, if one walks toward the other end of the street, it is actually translating. The actual details of how **Hold-and-Roll** was used for the experiment, and the decision to include it in the experiment are explained in the following chapters.

## CHAPTER 4

### **3DNAV: MULTI-TOUCH SYSTEM PROTOTYPE**

This chapter covers the essential parts of **3DNav** and the considerations that led to the final design of this prototype. Therefore, the chapter has two objectives: First, describe 3DNav and its functioning, including the contribution of 3DNav. Second, explain how structure of 3DNav facilitated making the design of experiment (Chapter 5) independent of the input technologies tested. This requires providing some technical details of the apparatus.

#### **4.1 Preliminary Device Testing**

During the initial part of the research, it wasn't completely clear which device needed to be used to compare the multi-touch device, nor which gestures were needed for 3DNav. The system itself was developed with various devices in order to address these questions. The decision to test the multi-touch display versus the GamePad controller is explained in detail in Chapter 5. This section provides information about different devices that were tested and how they were implemented.

##### **4.1.1 Device Listeners and Common Interfaces**

Some devices needed to be implemented using third-party API or using the WINAPI low-level access to devices. This led this author to adopt the common design pattern [56, 64], known as the observer pattern. This pattern allows a central object (subject) to keep a list of dependents (observers). In this approach the device has a listener waiting for an event. Each device has its own device listener. An example for the 3D mouse is shown in Listings 4.1, 4.2, and 4.3. The listener (Listing 4.1) provides an interface for any class that needs to receive the event data input when it triggers. The registry (Listing 4.2) provides two

functionalities: registration to the list of observers (*registryListener(...)*), and the signaling method for the raw input methods (e.g., WINAPI) to signal an event to all subscribed observers. The concrete implementation is shown in Listing 4.3. Finally, it is important to mention that a common interface was created for all game navigation controllers, which is called **InputPad**. This interface allows users to have common functionalities across devices. The **InputPad** interface is shown in Listing 4.4.

Listing 4.1: Observer Pattern (Listener.h)

```
1 typedef struct DOF6 {
2     int tx;
3     int ty;
4     int tz;
5     int rx;
6     int ry;
7     int rz;
8 } DOFSix;
9
10 typedef struct BDATA {
11     unsigned char p3;
12     unsigned char p2;
13     unsigned char p1;
14 } BData;
15
16 class Win3DX0greListener
17 {
18 public:
19     virtual bool handleNavigator(int message,DOFSix & D,BData & B) = 0;
20 };
```

Listing 4.2: Observer Pattern (Registry.h)

```
1 #include "Win3DX0greListener.h"
2 #include <vector>
3 #include <map>
4
5 class Win3DX0greEventRegistry
6 {
7     public:
8         static bool registerListener(int inMessage, Win3DX0greListener*
9             inListener);
10         static bool signalNavigator(int inMessage, int outMessage,DOFSix
11             & D,BData & B);
```

```

12     enum MESSAGES {M_UNKNOWN = 100,M_BUTTON = 101,M_NAVIGATOR = 102};
13     enum REGISTER {LISTEN_ALL=0};
14
15     protected:
16         static std::map<int, std::vector<Win3DX0greListener*>>
17             sListenerMap;
18 };

```

Listing 4.3: Observer Pattern (Registry.cpp)

```

1 // Implements the EventRegistry class
2 #include "Win3DX0greEventRegistry.h"
3 using namespace std;
4 map<int, vector<Win3DX0greListener*>> Win3DX0greEventRegistry::
5     sListenerMap;
6 bool Win3DX0greEventRegistry::registerListener(int inMessage,
7     Win3DX0greListener* inListener)
8 {
9     if (inMessage != REGISTER::LISTEN_ALL) return false;
10    sListenerMap[inMessage].push_back(inListener);
11    return true;
12 }
13 bool Win3DX0greEventRegistry::signalNavigator(int inMessage, int
14     outMessage,DOFSix & D,BData & B)
15 {
16     if (inMessage != REGISTER::LISTEN_ALL || sListenerMap.find(inMessage)
17         == sListenerMap.end())
18         return false;
19     for (auto iter = sListenerMap[inMessage].begin();
20         iter != sListenerMap[inMessage].end(); ++iter)
21     {
22         (*iter)->handleNavigator(outMessage, D, B);
23     }
24     return true;
25 }

```

Listing 4.4: InputPad (Interface)

```

1 #include <string>
2 using namespace std;
3 class InputPad
4 {
5 public:
6     virtual void updateInput(const double,const unsigned long long)=0;

```





Figure 4.1: 3D Mouse Space Sensor<sup>†</sup>

<sup>†</sup>: ©2014 3Dconnexion. All rights reserved.  
3Dconnexion, the 3Dconnexion logo, and  
other 3Dconnexion marks are owned by  
3Dconnexion and may be registered.

```
7  virtual string getPlayerName()=0;  
8  virtual string getDeviceName()=0;  
9  virtual void setPlayerName(const string & playerName)=0;  
10 virtual void setDeviceName(const string & deviceName)=0;  
11 virtual void enableInputDevice(bool enabled)=0;  
12 virtual void enableWrite(bool enabled)=0;  
13 };
```

### 4.1.2 3D Mouse

The 3D Mouse by 3DConnexion<sup>1</sup>, a division of the popular company Logitech<sup>2</sup>, provides a 6-DOF interaction. The company has reported over one million units sold as of March, 2011<sup>3</sup>.

---

<sup>1</sup>See <http://www.3dconnexion.com>.

<sup>2</sup>See <http://www.logitech.com>.

<sup>3</sup>See <http://bit.ly/1nFpujp>.

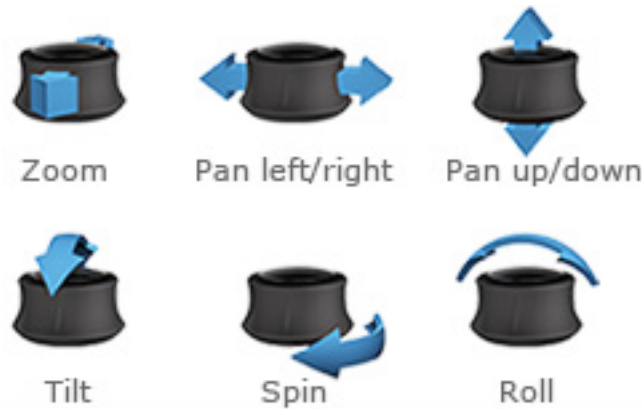


Figure 4.2: 3D Mouse Functions<sup>†</sup>

<sup>†</sup>: ©2014 3Dconnexion. All rights reserved.  
 3Dconnexion, the 3Dconnexion logo, and  
 other 3Dconnexion marks are owned by  
 3Dconnexion and may be registered.

This 3D mouse, shown in Figure 4.1, provides the three rotations yaw (spin), roll, and pitch (tilt). It also provides three additional movements that can be used for the translations in X, Y, and Z coordinates, which they called pan left/right, pan up/down, and zoom, respectively, as shown in Figure 4.2.

While the device does provide 6-DOF, the user is restricted by the device to small movements in each of the six possible interactions provided by the 3D Mouse. For example, the movements along the Z axis (zoom) are restricted to approximately 0.5 mm from center to either direction (in/out). The device can be a complement to current navigation techniques. However, during a pilot study users reported that the device was difficult to use. Nevertheless, it is used by some 3D cad and 3D modelers, working in AutoCAD, 3Ds Max, Maya, and other similar tools.

### Implementation

During the time that the 3D Mouse was tested, the Software Development Kit (SDK) provided by 3DConnexion had some problems remained unresolved. Currently, there is new SDK that has been updated as of late 2013. However, the testing was performed using low-

level driver support from WINAPI, running on Windows 7. The following implementation describes the low-level implementation<sup>4</sup> to access the 3D Mouse.

First, the device must be initialized, as shown in Listing 4.5. In lines 2-5, the process defines a few global variables required for the entire device recognition and device listening at the raw level. Then, the initialize function must be executed, as shown in lines 7-21. If the function is successful, it will return true, as shown in line 20.

After verifying during the initialization process that there is a device, there needs to be a search for the type of device present, which in this case, is the 3D mouse. The actual model is called 3D Mouse Space Navigator. The steps indicated in Listing 4.5 are shown in Listing 4.6. During the for loop included in lines 1-37, the process must determine if the 3D mouse is found, to later register up to 8 devices. First, the process must check if the device type is **RIM\_TYPEHID**, which is defined as the constant equal to 2. In line 15, the function *GetRawInputDeviceInfo(...)*, checks to see if the return value is a number greater than zero and the type is equal to **RIM\_TYPEHID**. Note that **RIDI\_DEVICEINFO** is equal to the hexadecimal number of 2000000B (5366870923 in decimal value). If this is the case, and the **usUsagePage** = 1 and the **usUsage**, the device is registered into the list **g\_RawInputDevices**. Finally, the devices found are registered using *RegisterRawInputDevices*, in line 38, using the list of devices already stored. It is important to note that **g\_pRawInputDeviceList** must have its memory released once the process completes using the free method (*free(g\_pRaw)*), which is equally important for any other data structure allocated using the C language method *malloc(...)*, when it is no longer needed.

Once the devices are registered, the actual data can be extracted from the device using the Windows event loop available. The actual data can be extracted from the device. The typical error checks are performed in Listing 4.7. The device input data process is shown in Listing 4.8. It is important to note that data packets are not triggered in one cycle, but

---

<sup>4</sup>See <http://www.ogre3d.org/forums/viewtopic.php?f=2&t=69156>.

independent events, which are captured by checking the **bRawData**. This variable is equal to one for the translation movement, equal to two for the rotation movements, and equal to three if the buttons were pressed. This is why lines 8-36 have an if-elseif-else logic, to obtain the data packets. Once the process is completed, there are different options for the developer. For example, the developer may desire to fire each event independently or test some criteria before firing. For example, if **bRotation** and **bTranslation** are true, then the input event is fired. An example of firing the event data listener is shown in line 35. The *signalNavigator(...)* is how the listeners are advised of new device data when fired. This is described in 4.1.1, including the listing for the observer pattern, already explained.

Listing 4.5: 3DMouse (Global and Initial Checks)

---

```

1  ...
2  //Global variables
3  PRAWINPUTDEVICELIST g_pRawInputDeviceList;
4  PRAWINPUTDEVICE     g_pRawInputDevices;
5  int                  g_nUsagePage1Usage8Devices;
6
7  BOOL InitRawDevices()
8  {
9      UINT nDevices;
10     if (GetRawInputDeviceList(NULL, &nDevices, sizeof(RAWINPUTDEVICELIST))
11         != 0)
12         return FALSE; // no 3D Mouse Found
13     if ((g_pRawInputDeviceList = (PRAWINPUTDEVICELIST)malloc(sizeof(
14         RAWINPUTDEVICELIST) * nDevices)) == NULL)
15         return FALSE; // malloc fails
16     if (GetRawInputDeviceList(g_pRawInputDeviceList, &nDevices, sizeof(
17         RAWINPUTDEVICELIST)) == -1)
18         return FALSE; //fails to get input devices
19
20     g_pRawInputDevices = (PRAWINPUTDEVICE)malloc( nDevices * sizeof(
21         RAWINPUTDEVICE) );
22     g_nUsagePage1Usage8Devices = 0;
23     ...
24     RETURN TRUE;
25 }
26 ...

```

Listing 4.6: 3DMouse (Partial Initialize)

---

```

1  for(UINT i=0; i<nDevices; i++)

```

```

2 {
3   if (g_pRawInputDeviceList[i].dwType == RIM_TYPEHID)
4     {
5       UINT nchars = 300;
6       TCHAR deviceName[300];
7       if (GetRawInputDeviceInfo( g_pRawInputDeviceList[i].hDevice,
8         RIDI_DEVICENAME, deviceName, &nchars) >= 0)
9         fprintf(stderr, "Device[%d]: handle=0x%x name = %S\n", i,
10           g_pRawInputDeviceList[i].hDevice, deviceName);
11
12       RID_DEVICE_INFO dinfo;
13       UINT sizeofdinfo = sizeof(dinfo);
14       dinfo.cbSize = sizeofdinfo;
15       if (GetRawInputDeviceInfo( g_pRawInputDeviceList[i].hDevice,
16         RIDI_DEVICEINFO, &dinfo, &sizeofdinfo ) >= 0)
17         {
18           if (dinfo.dwType == RIM_TYPEHID)
19             {
20               RID_DEVICE_INFO_HID *phidInfo = &dinfo.hid;
21               fprintf(stderr, "VID = 0x%x\n", phidInfo->dwVendorId);
22               fprintf(stderr, "PID = 0x%x\n", phidInfo->dwProductId);
23               fprintf(stderr, "Version = 0x%x\n", phidInfo->dwVersionNumber);
24               fprintf(stderr, "UsagePage = 0x%x\n", phidInfo->usUsagePage);
25               fprintf(stderr, "Usage = 0x%x\n", phidInfo->usUsage);
26
27               if (phidInfo->usUsagePage == 1 && phidInfo->usUsage == 8)
28                 {
29                   g_pRawInputDevices[g_nUsagePage1Usage8Devices].usUsagePage =
30                     phidInfo->usUsagePage;
31                   g_pRawInputDevices[g_nUsagePage1Usage8Devices].usUsage =
32                     phidInfo->usUsage;
33                   g_pRawInputDevices[g_nUsagePage1Usage8Devices].dwFlags = 0;
34                   g_pRawInputDevices[g_nUsagePage1Usage8Devices].hwndTarget =
35                     NULL;
36                   g_nUsagePage1Usage8Devices++;
37                 }
38             }
39         }
40     }
41   if (RegisterRawInputDevices( g_pRawInputDevices,
42     g_nUsagePage1Usage8Devices, sizeof(RAWINPUTDEVICE) ) == FALSE )
43     return FALSE; // fails to register
44   ...

```

Listing 4.7: 3DMouse (Process Input Error Checking)

---

```

1 RAWINPUTHEADER header;

```

```

2  UINT size = sizeof(header);
3  if ( GetRawInputData( (HRAWINPUT)lParam, RID_HEADER, &header, &size,
        sizeof(RAWINPUTHEADER) ) == -1)
4    return; \\error with device
5  }
6  size = header.dwSize;
7  LPRAWINPUT event = (LPRAWINPUT)malloc(size);
8  if (GetRawInputData( (HRAWINPUT)lParam, RID_INPUT, event, &size, sizeof(
        RAWINPUTHEADER) ) == -1)
9    return; \\error with input
10 ...

```

Listing 4.8: 3DMouse (Process Input)

```

1  ...
2  if (event->header.dwType == RIM_TYPEHID)
3  {
4    static BOOL bGotTranslation = FALSE,
5      bGotRotation = FALSE;
6    static int all6DOFs[6] = {0};
7    LPRAWHID pRawHid = &event->data.hid;
8    if (pRawHid->bRawData[0] == 1) // Translation vector
9    {
10     all6DOFs[0] = (pRawHid->bRawData[1] & 0x000000ff) | ((signed short)(
        pRawHid->bRawData[2]<<8) & 0xffffffff00);
11     all6DOFs[1] = (pRawHid->bRawData[3] & 0x000000ff) | ((signed short)(
        pRawHid->bRawData[4]<<8) & 0xffffffff00);
12     all6DOFs[2] = (pRawHid->bRawData[5] & 0x000000ff) | ((signed short)(
        pRawHid->bRawData[6]<<8) & 0xffffffff00);
13     bGotTranslation = TRUE;
14   }
15   else if (pRawHid->bRawData[0] == 2) // Rotation vector
16   {
17     all6DOFs[3] = (pRawHid->bRawData[1] & 0x000000ff) | ((signed short)(
        pRawHid->bRawData[2]<<8) & 0xffffffff00);
18     all6DOFs[4] = (pRawHid->bRawData[3] & 0x000000ff) | ((signed short)(
        pRawHid->bRawData[4]<<8) & 0xffffffff00);
19     all6DOFs[5] = (pRawHid->bRawData[5] & 0x000000ff) | ((signed short)(
        pRawHid->bRawData[6]<<8) & 0xffffffff00);
20     bGotRotation = TRUE;
21   }
22   else if (pRawHid->bRawData[0] == 3)
23   {
24     DOFSix D;
25     BData B;
26     D.tx = 0;
27     D.ty = 0;
28     D.tz = 0;

```

```

29     D.rx = 0;
30     D.ry = 0;
31     D.rz = 0;
32     B.p3 = (unsigned char)pRawHid->bRawData[3];
33     B.p2 = (unsigned char)pRawHid->bRawData[2];
34     B.p1 = (unsigned char)pRawHid->bRawData[1];
35     Win3DXOgreEventRegistry::signalNavigator(Win3DXOgreEventRegistry::
        LISTEN_ALL,Win3DXOgreEventRegistry::M_BUTTON,D,B);
36 }
37 }
38 ...

```

### 4.1.3 Inertial Navigation System

INS has become a popular method as an input device (see Chapter 2). One of the components is the gyroscope. As it was described in 3.2, the gyroscope sensor was used to complement multi-touch interaction. This type of system also includes accelerometers and compasses. With the advanced of MEMS devices, they have become small and pervasive, as seen with the WiiMote, and a standard in many tablets and smart phones, such as the iPhone. This sections describes how 3DNav implements the MEMS sensor by YEI Technologies<sup>5</sup>, called **3-Space Sensor**. The API provided by YEI Technologies works with C/C++ and Python. The actual API is pure C, where 3DNav can use the techniques already described in 4.1.2, to wrap it for a more modern C++ approach.

The actual sensor used comes with a wireless dongle that connects up to 15 devices. Listing 4.9 shows the C++ wrapper used in 3DNav that works with the YEI API, version 2.0.6.1<sup>6</sup>. The actual sensor wrapper is shown in Listing 4.10. YEI technologies provide additional software. For example, they provide a suite to do testing, which requires no coding at all. This can be quite useful for understanding the output of the device before creating a prototype.

---

<sup>5</sup>See <http://www.yeitechnology.com>

<sup>6</sup>Updated March 06, 2014.

Listing 4.9: YEI 3Space Sensor Wireless (Wrapper)

---

```

1 #include <yei_threespace_api.h>
2 using namespace std;
3 class MemsYeiDongle
4 {
5 public:
6     typedef enum YEI_DongleStatus
7     {
8         CONNECTED,
9         COMPORT_FAILED,
10        DONGLE_FAILED,
11        UNKNOWN
12    };
13    MemsYeiDongle(const string & friendlyDongleName);
14    ~MemsYeiDongle();
15    unsigned int getDeviceId();
16    YEI_DongleStatus connect();
17    shared_ptr<const string> getComPortFriendlyName();
18    shared_ptr<const string> getComPortName();
19    shared_ptr<const string> getFriendlyDongleName();
20    bool isConnected();
21    YEI_DongleStatus reconnect();
22    int getSensorType();
23    void closeDevice();
24    YeiSensorData getData();
25 private:
26    bool isConnected;
27    TSS_Device_Id dongle;
28    TSS_ComPort comport;
29    string dongleName;
30    const string comPortFriendlyName;
31    const string comPortName;
32    int sensorType;
33 };

```

Listing 4.10: YEI 3Space Sensor (Wrapper)

---

```

1 #include <yei_threespace_api.h>
2 using namespace std;
3 class MemsYeiDongle
4 {
5 public:
6     typedef enum YEI_DongleStatus
7     {
8         CONNECTED,
9         COMPORT_FAILED,
10        DONGLE_FAILED,

```



```

11     UNKNOWN
12 };
13 MemsYeiDongle(const string & friendlyDongleName);
14 ~MemsYeiDongle();
15 unsigned int getDeviceId();
16 YEI_DongleStatus connect();
17 shared_ptr<const string> getComPortFriendlyName();
18 shared_ptr<const string> getComPortName();
19 shared_ptr<const string> getFriendlyDongleName();
20 bool isConnected();
21 YEI_DongleStatus reconnect();
22 int getSensorType();
23 void closeDevice();
24 YeiSensorData getData();
25 private:
26     bool isConnected;
27     TSS_Device_Id dongle;
28     TSS_ComPort comport;
29     string dongleName;
30     const string comPortFriendlyName;
31     const string comPortName;
32     int sensorType;
33 };

```

#### 4.1.4 Microsoft Kinect

In the original design of 3DNav, the Microsoft Kinect was not contemplated to be one of the devices that would be used. The reasons were two-fold: First, the Kinect was not originally well-suited to capture close proximity movements (i.e., within a few centimeters of the Kinect camera). Second, the dissertation focus is not computer vision. However, given the popularity of the Microsoft Kinect and the easy-to-use API, this author considered doing some testing.

A bit of background is required to understand the Microsoft Kinect. The original design and intent was the capture of in-air gestures when using the Microsoft XBox 360 video game console. Given the popularity and potential of the Kinect, the working system was reverse-engineered to provide an unofficial API. It is not certain if this led Microsoft to

release Microsoft Kinect for Windows, but in any case, the release of Kinect for Windows came with an official SDK provided from Microsoft and a modified firmware that included a feature for the near/far distance detection. This SDK was used for testing. While Microsoft has released the new Kinect 2 for the XBox One video game console, and will be releasing a Kinect 2 for windows soon, note that when the Kinect sensor is mentioned in this dissertation, it refers to the Microsoft Kinect (version 1) for Windows.

The Kinect includes a microphone array, an infrared emitter, an infrared receiver, a color camera, and a DSP chip. This all connects using a Universal Serial Bus (USB) 2 [26]. The use of both cameras allows for 3D vision to be captured. In other words, depth can be captured using the Kinect. There are some limitations to the use of the Kinect, which are listed below (adapted from [26]):

- Horizontal viewing angle: 57°.
- Vertical viewing angle: 43°.
- Far mode distance: 1.2 meters to 4.2 meters.
- Near<sup>7</sup> mode distance: 0.4 meters to 3.0 meters.
- Standard Depth Range: 8000 mm.
- Near Mode Depth Range: 400 mm.
- Temperature: 5° to 35° Celsius.
- Motor controller vertical viewing position:  $\pm 28^\circ$

One feature provided by the Kinect SDK is the ability to track the human skeleton, called **Skeletal Tracking**. It allows the tracking of the movements of joints by a user, as shown in Figure 4.3. The tracking can be done for up to six persons simultaneously [26], in its FOV.

---

<sup>7</sup>Only available with firmware update

## The Implementation

The actual implementation differed from the other devices. While devices were kept in 3DNav, the Kinect was coded using an external program. There were different reasons to do this. First, the Kinect SDK samples at the time of testing were given in their majority in C# and all the documentation was created with C# in mind. Another reason is that people have used an array of Kinects, using an external process or computer to capture the data, and sent the information to a another process. Therefore, this author decided to use a different approach to this device, to show that 3DNav could also work with external processes. This provides the flexibility to have different processes in the same computer or across a network.

This implementation was achieved using a WINAPI messaging system to send data between processes, as shown in Listing 4.11. This allows the Kinect process to send messages to an external application running on the same computer. Of course, the implementation of the method **SendMessageTo** could be replaced by another type of system or networking communications (e.g., Communication Sockets [135]).

To perform the 3D navigation using the Kinect, a basic set of gestures was created, as shown in Figure 4.3. In this figure, there are 8 positions where the user can move his or her right or left hand to achieve a desired action. The user was presented with a model of ruins from a ancient civilization, as shown in Figure 4.4, in full screen. The user could also see the different parts of the system if needed<sup>8</sup>, as shown in Figure 4.5. What the user experienced, from his or her point of view, is shown in Figure 4.5.

The final goal for the Kinect sensor, in the case of this dissertation, was to see if it could be used for desktop environments. While it had been possible to mount it with a stand, in general, when using it in close proximity to the desktop, it did not yield the results needed.

---

<sup>8</sup>Used primarily for development.

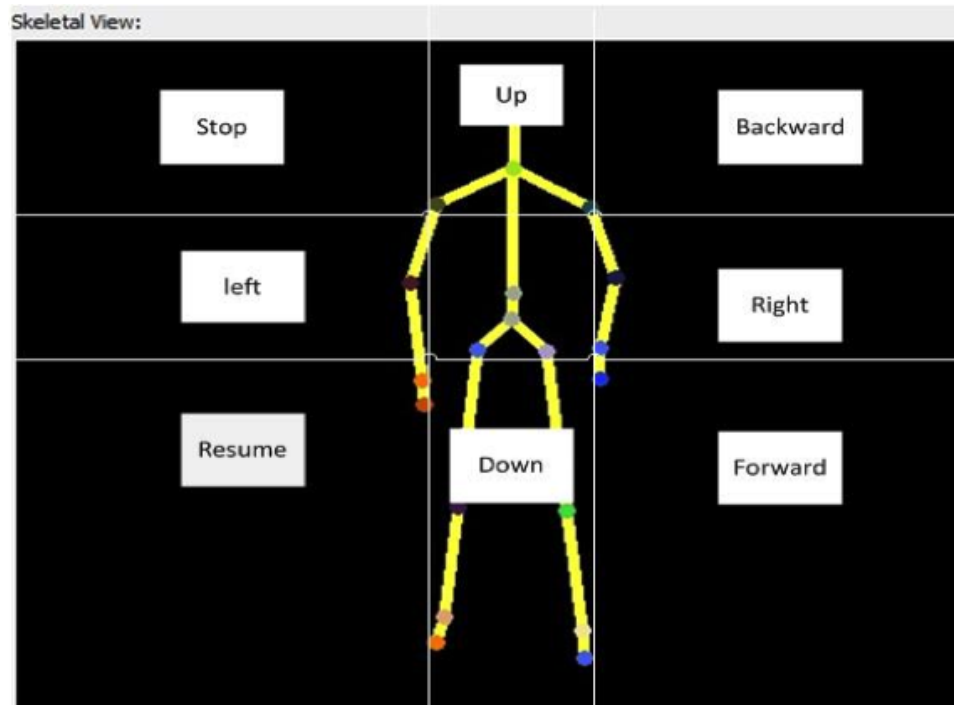


Figure 4.3: C# Skeleton Viewer (Possible Gestures)

With this said, this device was not pursued further. It is hoped that similar new devices present an emphasis in near version, while keeping similar purchase costs. Finally, it is important to mention that the development of the prototype was led by this author, with the collaboration of Jose Camino, Karina Harfouche, and Holly Smith.

Listing 4.11: Kinect (WINAPI Messages)

```

1 private struct COPYDATASTRUCT
2 {
3     public IntPtr dwData;
4     public int cbData;
5     [MarshalAs(UnmanagedType.LPStr)]
6     public string lpData;
7 }
8 private const int WM_COPYDATA = 0x4A;
9 [DllImport("user32.dll", SetLastError = true)]
10 static extern IntPtr
11     FindWindow(string lpClassName, string lpWindowName);
12 [DllImport("User32.dll", EntryPoint = "SendMessage")]
13 private static extern int
14     SendMessage(IntPtr hWnd, int Msg, int wParam, ref COPYDATASTRUCT
        lParam);

```

```

15
16 IntPtr hwnd FindWindow(null, "QTApp:KinectMessageHandler");
17
18 private void sendMessageTo(IntPtr hwnd, String msg)
19 {
20     int wParam = 0;
21     int result = 0;
22
23     if (hwnd != IntPtr.Zero)
24     {
25         byte[] sarr = System.Text.Encoding.Default.GetBytes(msg);
26         int len = sarr.Length;
27         COPYDATASTRUCT cds;
28         cds.dwData = IntPtr.Zero;
29         cds.lpData = msg;
30         cds.cbData = len + 1;
31         result = SendMessage(hwnd, WM_COPYDATA, wParam, ref cds);
32     }
33 }

```

Listing 4.12: Skelton Tracking (Left/Right)

```

1 ...
2 if (yMin != 0.0f && yMax != 0.0f && xMax != 0.0f && xMin != 0.0f)
3 {
4     //sends 'right' command: right hand in right square
5     if (skeleton.Joints[JointType.HandRight].Position.X >= xMax + 0.3f &&
6         skeleton.Joints[JointType.HandRight].Position.Y < yMax &&
7         skeleton.Joints[JointType.HandRight].Position.Y > yMin)
8     {
9         sendMessageTo(hwnd, "right");
10    }
11    //sends 'left' command: left hand in left square
12    if (skeleton.Joints[JointType.HandLeft].Position.X <= xMin - 0.3f &&
13        skeleton.Joints[JointType.HandLeft].Position.Y < yMax && skeleton
14        .Joints[JointType.HandLeft].Position.Y > yMin)
15    {
16        sendMessageTo(hwnd, "left");
17    }
18 }
19 ...

```

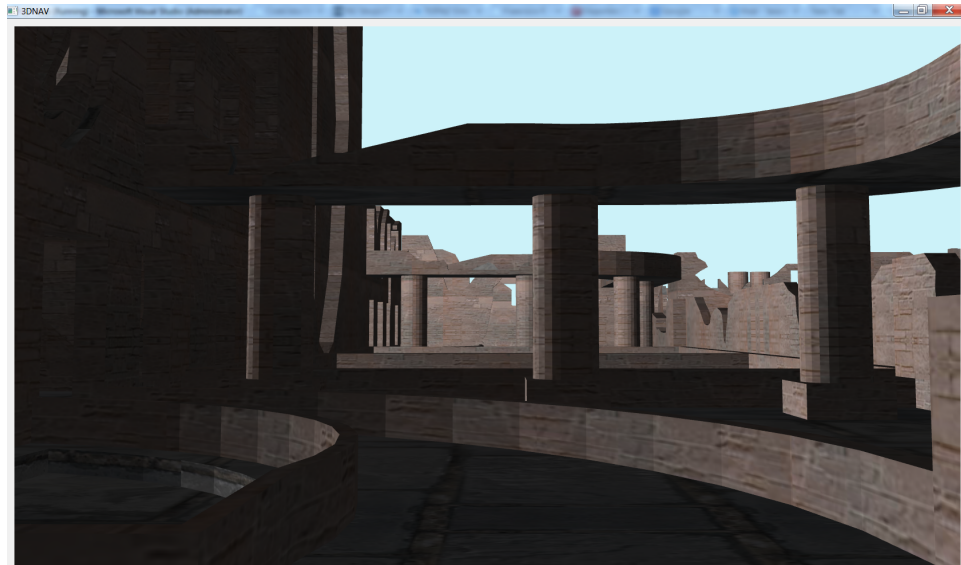


Figure 4.4: Ancient 3D ruins

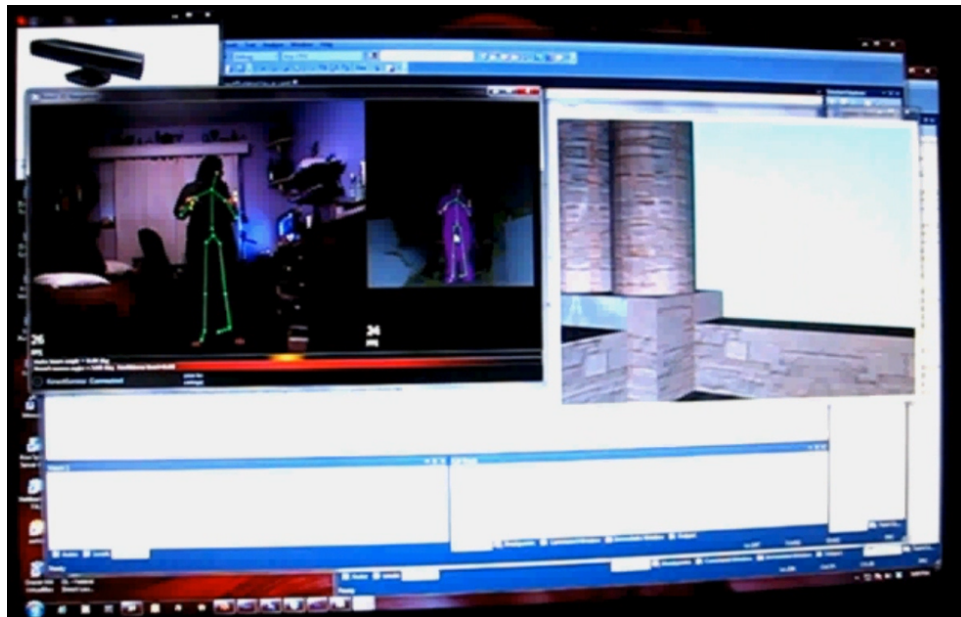


Figure 4.5: Screen Closeup



Figure 4.6: From the user's point of view

### 4.1.5 Keyboard and Mouse

The keyboard and mouse usage and implementation are pervasive. Therefore, very little will be mentioned in this dissertation about these input devices. However, they are available in 3DNav because of how common and available they are. What is more important is how to handle the 3D navigation using a keyboard or a mouse. The following example, in Listing 4.13, demonstrates the interface for keyboard navigation controller.

Listing 4.13: Keyboard (Navigation)

```
1 class KeyPad : public InputPad
2 {
3 public:
4     typedef float real;
5     KeyPad(Navigation * navigation, OIS::Keyboard* keyboard, const string &
6           playerName=string("Player"), const string & deviceName=string("
7           Keyboard"));
8     virtual void updateInput(const double, const unsigned long long);
9     virtual void enableInputDevice(bool enabled){mEnabled = enabled;}
10    virtual void enableWrite(bool enabled){mWriteEnabled = enabled;}
```

```

9   virtual void setName(const string & playerName) { mName =
      playerName; }
10  virtual void setName(const string & deviceName) { mDeviceName =
      deviceName;}
11  virtual string getName(){ return mName;}
12  virtual string getDeviceName(){ return mDeviceName;}
13
14  private:
15      //Methods
16      void moveLeft(real f,real t);
17      void moveRight(real f,real t);
18      void moveUp(real f,real t);
19      void moveDown(real f,real t);
20      void moveIn(real f,real t);
21      void moveOut(real f,real t);
22      void rotateX(real f,real t);
23      void rotateY(real f,real t);
24      void rotateZ(real f,real t);
25      void rotateInverseX(real f,real t);
26      void rotateInverseY(real f,real t);
27      void rotateInverseZ(real f,real t);
28      Navigation * mNavigation;
29      OIS::Keyboard * mKeyboard;
30      string mName;
31      string mDeviceName;
32      bool mEnabled;
33      bool mWriteEnabled;
34  };

```

## 4.1.6 GamePad

The GamePad is a type of video game controller that is the default device for every video console game today. While video games tried different controllers in the late seventies and early eighties, such as the joystick (4.7a)), paddles (Figure 4.7b), and other input devices, the release of the GamePad in the third generation of the Nintendo Entertainment System (NES) video game console marked the before and after for game controllers (Figure 4.7c). The GamePad (also called joypad), shown in Figure 4.8a, provides a common form to navigate in virtual worlds. In particular, 3DNav uses the XBox 360 controller, shown in Figure





Figure 4.7: Images

4.8b. This GamePad comes from the seventh generation of video game consoles [129]. The evolution of the GamePad has made them an excellent candidate for 3D navigation. The most current GamePad, from the eight generation, is the XBox One controller, shown in Figure 4.8c.

### The Implementation

To use the XBox 360 GamePad in 3DNav, the use of the Microsoft XInput<sup>9</sup> [81, 137] library was utilized. XInput is the most current Microsoft Library for input devices, in specific, game controllers. Just like in the previous devices already mentioned, the wrapping of the low-level access to the controller was developed, as shown in Listings 4.14 and 4.15. The actual navigation was configured just like in the keyboard example. Here is a partial sample of the code needed, shown in Listings 4.16 and 4.17. Finally, in 3DNav it is important to normalize the data and smooth the input<sup>10</sup>, to have a more continuous navigation when using the GamePad, as shown in Listing 4.18.

Listing 4.14: XBox360 Controller (h)

---

```

1 #define WIN32_LEAN_AND_MEAN
2 // We need the Windows Header and the XInput Header
3 #include <windows.h>

```

---

<sup>9</sup>See [http://msdn.microsoft.com/en-us/library/windows/desktop/ee417014\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ee417014(v=vs.85).aspx)

<sup>10</sup>See <http://www.altdevblogaday.com/2011/06/16/analog-input-processing/>



(a) Sony Dual-Shock (PS1)



(b) Xbox 360



(c) Xbox One

Figure 4.8: Images

```
4 #include <XInput.h>
5 // NOTE: COMMENT THE NEXT LINE, IF YOU ARE NOT USING A COMPILER THAT
6 // SUPPORTS #pragma TO LINK LIBRARIES
7 #pragma comment(lib, "XInput.lib")
8
9 class C XboxController
10 {
11 private:
12     XINPUT_STATE _controllerState;
13     int _controllerNum;
14 public:
15     typedef XINPUT_STATE controllerx_state;
16     C XboxController(int playerNumber);
17     XINPUT_STATE GetState();
18     bool IsConnected();
19     void Vibrate(WORD leftVal = 0, WORD rightVal = 0);
20 };
```

Listing 4.15: XBox360 Controller (cpp)

```
1 #include "C XboxController.h"
2 C XboxController::C XboxController(int playerNumber)
3 {
4     _controllerNum = playerNumber - 1;
5 }
6 XINPUT_STATE C XboxController::GetState()
7 {
8     ZeroMemory(&_controllerState, sizeof(XINPUT_STATE));
9     XInputGetState(_controllerNum, &_controllerState);
10    return _controllerState;
11 }
12 bool C XboxController::IsConnected()
13 {
14     ZeroMemory(&_controllerState, sizeof(XINPUT_STATE));
15     DWORD Result = XInputGetState(_controllerNum, &_controllerState);
```

```

16  if(Result == ERROR_SUCCESS)
17      return true;
18  else
19      return false;
20  }
21  void CXBoxController::Vibrate(WORD leftVal, WORD rightVal)
22  {
23      XINPUT_VIBRATION Vibration;
24      ZeroMemory(&Vibration, sizeof(XINPUT_VIBRATION));
25      Vibration.wLeftMotorSpeed = leftVal;
26      Vibration.wRightMotorSpeed = rightVal;
27      XInputSetState(_controllerNum, &Vibration);
28  }

```

Listing 4.16: Game Navigation Controller(h)

```

1  class XPad : public InputPad
2  {
3  public:
4      typedef enum {PlayerOne=1,PlayerTwo=2,PlayerThree=3,PlayerFour=4}
          player_number;
5      typedef enum {Analog,Digital} process_type;
6      typedef float real;
7      typedef struct controller_data { ... };
8      typedef struct normalized_data { ... };
9      typedef struct deadzone_data { ... };
10     typedef struct player_data { ... };
11     typedef struct controller_values { ... };
12     ...
13     XPad(...);
14     void setNormalizedDeadZoneL(real v){mDeadzone.left =v;}
15     void setNormalizedDeadZoneR(real v){mDeadzone.right =v;}
16     void setNormalizedDeadZoneTrigger(real v){mDeadzone.trigger =v;}
17     player_data getPlayerData() { return mPlayerData;}
18     void setPolarity(bool p){mPolarity=p;}
19     void setDigital(bool d){mDigital=d;}
20     bool isConnected(){return true;}
21     virtual void updateInput(const double timeSinceLastFrame,const unsigned
          long long cCount);
22     virtual void enableInputDevice(bool enabled){mEnabled = enabled;}
23     virtual void enableWrite(bool enabled){mWriteEnabled = enabled;}
24     virtual void setPlayerName(const string & playerName) { mPlayerName =
          playerName; }
25     virtual void setDeviceName(const string & deviceName) { mDeviceName =
          deviceName;}
26     virtual string getPlayerName(){ return mPlayerName;}
27     virtual string getDeviceName(){ return mDeviceName;}
28     XINPUT_STATE getState(){ return mPad.GetState();}

```

```

29 private:
30 ...
31 };

```

Listing 4.17: Game Navigation Controller(cpp)

```

1 class XPad : public InputPad
2 {
3 public:
4     typedef enum {PlayerOne=1,PlayerTwo=2,PlayerThree=3,PlayerFour=4}
        player_number;
5     typedef enum {Analog,Digital} process_type;
6     typedef float real;
7     typedef struct controller_data { ... };
8     typedef struct normalized_data { ... };
9     typedef struct deadzone_data { ... };
10    typedef struct player_data { ... };
11    typedef struct controller_values { ... };
12    ...
13    XPad(...);
14    void setNormalizedDeadZoneL(real v){mDeadzone.left =v;}
15    void setNormalizedDeadZoneR(real v){mDeadzone.right =v;}
16    void setNormalizedDeadZoneTrigger(real v){mDeadzone.trigger =v;}
17    player_data getPlayerData() { return mPlayerData;}
18    void setPolarity(bool p){mPolarity=p;}
19    void setDigital(bool d){mDigital=d;}
20    bool isConnected(){return true;}
21    virtual void updateInput(const double timeSinceLastFrame,const unsigned
        long long cCount);
22    virtual void enableInputDevice(bool enabled){mEnabled = enabled;}
23    virtual void enableWrite(bool enabled){mWriteEnabled = enabled;}
24    virtual void setPlayerName(const string & playerName) { mPlayerName =
        playerName; }
25    virtual void setDeviceName(const string & deviceName) { mDeviceName =
        deviceName;}
26    virtual string getPlayerName(){ return mPlayerName;}
27    virtual string getDeviceName(){ return mDeviceName;}
28    XINPUT_STATE getState(){ return mPad.GetState();}
29 private:
30 ...
31 };

```

Listing 4.18: XBox360 Controller (Smooth Input)

```

1 void XPad::normalizedData(controller_data & data,normalized_data & ndata)
2 {
3     //scale data will give you input from -1 to 1
4     //normalized will give you 0 to 1.

```

```

5  ndata.LX = NMath<short, real>::scaledData(MINTHUMB, MAXTHUMB, data.LX);
6  ndata.LY = NMath<short, real>::scaledData(MINTHUMB, MAXTHUMB, data.LY);
7  ndata.RX = NMath<short, real>::scaledData(MINTHUMB, MAXTHUMB, data.RX);
8  ndata.RY = NMath<short, real>::scaledData(MINTHUMB, MAXTHUMB, data.RY);
9  ndata.TL = NMath<short, real>::normalizedData(MINTRIGGER, MAXTRIGGER, data
    .TL);
10 ndata.TR = NMath<short, real>::normalizedData(MINTRIGGER, MAXTRIGGER, data
    .TR);
11 }
12 double XPad::smoothInput(double v, double alpha)
13 {
14     //data is expected to be normalized.
15     if (std::abs(v) < alpha)
16         return make_pair(0, 0);
17     double u = NMath<real>::sgn(v) * (std::abs(v) - alpha) / ( 1.0f - alpha
        );
18     return std::make_pair(u);
19 }

```

### 4.1.7 Multi-Touch

There are multiple forms used to detect the multi-touch input at the hardware level, as explained in Chapter 2. Once the touches are detected, the data is communicated to the system using different types of drivers or protocols. For example, in vision touch detection, Tangible User Interface Objects (TUIO) are quite popular<sup>11</sup>. The protocol is defined in [105], and is illustrated with a real user case in the ReactiVision system [104]. In capacitive systems, it is common to use the drivers provided by the manufacturer or the operating system. For example, the 3M M2256PW multi-touch display comes with its own drivers provided by the manufacturer. In addition, this multi-touch (and others) can work using the WINAPI<sup>12</sup> multi-touch interface. Most languages and API have support for multi-touch. This is true in objective-C for iPad and iPhone, C# for Windows (version 7 or greater)

---

<sup>11</sup>See <http://tuio.org>.

<sup>12</sup>Available in Windows 7 and 8.

desktop, tablets, phones, and Qt<sup>13</sup> framework. Some of those have support for raw data (points and traces), other for specific types of gestures, and some of them, support both options (raw data and gesture mode). For the particular case of this dissertation, this author tested with 3M API, Qt touch libraries, GWC by ideaum, and WINAPI. The latter provided the best results when using raw data.

It is common for raw data to provide trace id, as well as X and Y coordinates. Additional data is obtained depending on the system. For example, in WINAPI, it is possible to obtain a time stamp, as well as the contact size for the X and Y coordinates. It is important to note that the contact size does not provide a lot of information. It fluctuates between low and high levels for the contact size, at least when tested with the 3M display. The gesture mode provides defined gestures to work with. For example, the WINAPI used in Windows 7 provided a very small set of available gestures, which are the most common ones. GWC provides richer set of gestures; however, it does create a lot of ambiguous. There is custom gesture recognition that has been tried, as explained in Section 1.6, as well as Chapters 2 and 3. For this dissertation, the raw touch event data was used with the WINAPI. This provided the opportunity to have custom-made recognition methods, such as FETOUCH. The gesture recognition was performed with gesture works because it provides a larger set of gestures, and it allowed the experiment to be performed in an unbiased way, using a popular library. Yield was created to remove the ambiguity of GWC, as described in Chapter 3.

### **Overview of Windows 7 Raw Multi-Touch**

The implementation for raw data made use of the WINAPI using Windows 7 and Microsoft Visual Studio 2012. Most of the coding was done in C++, while the FETOUCH was tested using C#. The primary reason for the decision to use WINAPI was because

---

<sup>13</sup>See Qt-project.org

of its performance, the understanding of WINAPI by the author and the vast amount of information available in the Microsoft Software Developer Network (MSDN) site<sup>14</sup>, and resources such as [113, 165]. Originally, the 3M M2256PW multi-touch display API was used, but the results were not as satisfactory as those obtained with the WINAPI. Also, the WINAPI offered the flexibility to use any multi-touch desktop, regardless of the brand, in a Windows environment, including laptops, tablets, and phones. Finally, the maturity of the Windows drivers provided an additional incentive to use the WINAPI.

### **Implementation: Multi-Touch Using Windows 7**

When working with multi-touch using the WINAPI, the initialization, the event loop, and the touch event (down, move, and up) are important to explain. To implement a solution with multi-touch using the native calls of Microsoft requires some understanding of the WINAPI. A detailed explanation of this topic is outside the scope of this dissertation. The reader is suggested to review the classic *Windows Programming* book (fifth edition<sup>15</sup>) by Charles Petzold [165] and the book by Kiriaty et al. [113]. Windows 8 has some additional features, which can be accessed using either the WINAPI or the Windows run-time (WinRT) libraries [166].

Listing 4.19 provides the basic initialization of a Windows process and the Multi-Touch functionality. In this listing, lines 8-17 check all the possible direct input interactions available in Windows 7. For example, these lines check if a pen is available to the system. The one that is of concern to this dissertation is the multi-touch functionality. This is checked in line 13. This means that the system must have a multi-touch display attached at the moment of this check. Otherwise, the application will exit. Then, like any typical WINAPI application, the initialization obtains calls in *WinMain*, as shown in Listing 4.20 (line 16).

---

<sup>14</sup>See <http://msdn.microsoft.com>

<sup>15</sup>Note that the next edition does not cover WINAPI.

Once the multi-touch has been initialized, the next step is to receive the messages in the Windows event loop. The *GetMessage(...)* method is seen in Listing 4.20. An alternate way of receiving the messages is using *PeekMessage(...)*. The difference is that by receiving messages, messages are removed. By peeking at a message, the messages are left in the loop. This is very important when working with 3D graphics, as the main event loop may be controlled by another driver. For example, in OGRE, if the developer was to call the *GetMessage(...)* method, it will remove other important messages from Windows. In games, *PeekMessage(...)* is the preferred method [137].

The common way to handle messages in WINAPI is shown in Listing 4.21. With regards to multi-touch events, Listing 4.22 shows the correct form to handle multi-touch data events. If there is multi-touch input data to be processed, then a for loop is executed in lines 15-29. This means that each input is fired independently. Furthermore, each data point in this cycle may have different event modes, which includes down, move, and up. As stated earlier in this dissertation, the down event happens when the finger is pressed for the first time into the display, the move event generates the continuous set of data points while the point has stayed on the touch surface, (regardless of if it is moving or not), and the up event marks the finger's removal from the surface. Each trace is denoted by an id number, in this case, provided by the Windows system. Finally, it is important to deallocate the multi-touch display resource, which had been previously allocated (in the initialization phase), as shown in line 37.

Finally, the events help to handle the multi-touch interaction. The basic interaction generates 2D points. For example, a painting application can be used with this information. The utilization of the 2D point information received depends on the application. 3DNav stores the points in a hash table (map), where each trace becomes the key, and the value of the map becomes a list of points (in order of arrival). In FETOUCH, this is later broken in half to compute the gesture. This is really application specific. Listing 4.23 displays



an example of using (thread-safe) data structures to store the data points, during the down, move, and up events. Two important aspects about the data touch points must be taken into account. First, the data structures shown are thread-safe, meaning that it is safe to run them in parallel if needed<sup>16</sup>. Second, during the up event, no points are removed from the hash table. This is because it is expected to have another process deal with the points and delete them. Finally, the **Trace** class, in this context, contains information about a data point that belongs to a certain trace.

Listing 4.19: WINAPI (Multi-Touch init)

```
1 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
2 {
3     g_hInst = hInstance;
4     g_hWnd = CreateWindow(g_wszWindowClass, g_wszTitle, WS_OVERLAPPEDWINDOW
5         ,
6         CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);
7     if (!hWnd)
8         return FALSE;
9     if (value & NID_READY){ /* stack ready */}
10    if (value & NID_INTEGRATED_TOUCH) { /* integrated touch device in the PC
11        enclosure */}
12    if (value & NID_EXTERNAL_TOUCH) { /* Touch device is not integrated in
13        the PC enclosure */ }
14    if (value & NID_INTEGRATED_PEN){ /* Integrated pan support is in the PC
15        enclosure */ }
16    if (value & NID_EXTERNAL_PEN){ /* Pan is supported but not as part of
17        the PC enclosure */ }
18    if (((value & NID_MULTI_INPUT) == NID_MULTI_INPUT))
19    {
20        if(!RegisterTouchWindow(hWnd, 0))
21            return FALSE;
22    }
23    ...
24    ASSERT(IsTouchWindow(hWnd, NULL));
25    ShowWindow(hWnd, nCmdShow);
26    UpdateWindow(hWnd);
27    ...
28    return TRUE;
29 }
```

Listing 4.20: WINAPI (Multi-Touch WinMain)

<sup>16</sup>There still a synchronization aspect between down, move, and up events.

---

```

1  int APIENTRY wWinMain(HINSTANCE hInstance,
2                          HINSTANCE hPrevInstance,
3                          LPWSTR    lpCmdLine,
4                          int        nCmdShow)
5  {
6      ...
7      GdiplusStartupInput gdiplusStartupInput;
8      ULONG_PTR          gdiplusToken;
9      MSG msg;
10     HACCEL hAccelTable;
11     //Initialize GDI+
12     GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
13
14     ...
15     MyRegisterClass(hInstance);
16     if (!InitInstance (hInstance, nCmdShow))
17         return FALSE;
18     ...
19     // Main message loop:
20     while (GetMessage(&msg, NULL, 0, 0))
21     {
22         if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
23         {
24             TranslateMessage(&msg);
25             DispatchMessage(&msg);
26         }
27     }
28     ...
29     GdiplusShutdown(gdiplusToken);
30     return (int) msg.wParam;
31 }

```

Listing 4.21: WINAPI (Events)

---

```

1  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
    lParam)
2  {
3      int wmId, wmEvent;
4      PAINTSTRUCT ps;
5      HDC hdc;
6      switch (message)
7      {
8          case WM_COMMAND:
9              ...
10             break;
11             case WM_PAINT:
12                 ...

```

```

13         break;
14     case WM_TOUCH:
15         ...
16         break;
17     case WM_DESTROY:
18         ...
19         PostQuitMessage(0);
20         break;
21     default:
22         return DefWindowProc(hWnd, message, wParam, lParam);
23     }
24     return 0;
25 }

```

Listing 4.22: WINAPI (Multi-Touch Touch Events)

```

1  switch (message)
2  {
3  ...
4  case WM_TOUCH:
5      UINT numInputs = (UINT) wParam;
6      TOUCHINPUT* pTIArry = new TOUCHINPUT[numInputs];
7      if(NULL == pTIArry )
8      {
9          CloseTouchInputHandle((HTOUCHINPUT)lParam);
10         break;
11     }
12
13     if(GetTouchInputInfo((HTOUCHINPUT)lParam, numInputs, pTIArry,
14         sizeof(TOUCHINPUT))
15     {
16         for(UINT i=0; i<numInputs; ++i)
17         {
18             if(TOUCHEVENTF_DOWN == (pTIArry[i].dwFlags &
19                 TOUCHEVENTF_DOWN))
20             {
21                 OnTouchDownHandler(hWnd, pTIArry[i]);
22             }
23             else if(TOUCHEVENTF_MOVE == (pTIArry[i].dwFlags &
24                 TOUCHEVENTF_MOVE))
25             {
26                 OnTouchMoveHandler(hWnd, pTIArry[i]);
27             }
28             else if(TOUCHEVENTF_UP == (pTIArry[i].dwFlags &
29                 TOUCHEVENTF_UP))
30             {
31                 OnTouchUpHandler(hWnd, pTIArry[i]);
32             }
33         }
34     }
35 }

```

```

29     }
30 }
31
32     CloseTouchInputHandle((HTOUCHINPUT)lParam);
33     delete [] pTIArray;
34     break;
35
36     case WM_DESTROY:
37         if(!UnregisterTouchWindow(hWnd))
38             ...
39         break;
40     ...
41 }
42 ...

```

Listing 4.23: WINAPI (Multi-Touch down,move,up)

```

1 void onTouchDownHandler(HWND hWnd, const TOUCHINPUT& ti)
2 {
3     POINT pt = GetTouchPoint(hWnd, ti);
4     unsigned long iCursorId = GetTouchContactID(ti);
5     Trace trace(iCursorId,ti,hWnd);
6     trace.requestTimeStamp();
7     concurrent_vector<Trace> vtrace;
8     vtrace.push_back(trace);
9     mapTraces.insert(pair<unsigned long,concurrent_vector<Trace>>(iCursorId
10     ,vtrace));
11 }
12 void onTouchMoveHandler(HWND hWnd, const TOUCHINPUT& ti)
13 {
14     unsigned long iCursorId = GetTouchContactID(ti);
15     POINT pt;
16     pt = GetTouchPoint(hWnd, ti);
17     concurrent_unordered_map<unsigned long,concurrent_vector<Trace>>::
18     iterator p = mapTraces.find(iCursorId);
19     Trace trace(iCursorId,ti,hWnd);
20     trace.requestTimeStamp();
21     int touchCount = mapTraces.size();
22     int traceSize = p->second.size();
23     Trace lastTrace = p->second[traceSize - 1];
24     ...
25     if (moving)
26         p->second.push_back(trace);
27     else //not moving
28         p->second[traceSize - 1].IncCount();
29 }

```

```

30 void OnTouchUpHandler(HWND hWnd, const TOUCHINPUT& ti)
31 {
32     //Delete point only if not handled by a external
33     // resource such as gesture recognition method.
34 }

```

### Implementation: Gesture Data

Besides FETOUCH, 3DNav implemented GWC, which was used for the experiment detailed in Chapter 5. A more detailed explanation of why it was selected for the experiment is given on that chapter. GWC API comes with a C binding of its Dynamic Link Library (DLL)<sup>17</sup> and a few C++ classes. Everything else is a black box, since it is contained in their DLL. The 3DNav contains its own C++ classes. Listings 4.24 and 4.25, show the classes to handle GWC in 3DNav.

Listing 4.24: GWC (GWTouch.h)

---

```

1 #include <GestureWorksCore.h>
2 #include <GWCUtils.h>
3 ...
4 #ifdef _WIN64
5     #define GWDLL "GestureworksCore64.dll"
6 #elif
7     #define GWDLL "GestureworksCore32.dll"
8 #endif
9 class GwTouch
10 {
11 public:
12     GwTouch();
13     bool Register(HWND hWnd);
14     bool RegisterByWindowName(const string & windowName);
15     void init(int screenWidth, int screenHeight, string & pathGmlFile);
16     void resize(int screenWidth, int screenHeight);
17     void registerTouchObject(const string & objectName);
18     bool deregisterTouchObject(const string & objectName);
19     bool addGesture(const string & touchObjectName, const string &
20         gestureId);
21     bool removeGesture(const string & touchObjectName, const string &
22         gestureId);

```

---

<sup>17</sup>Early 2014 version of GWC.

```

21  bool addGestureSet(const string & touchObjectName, const string &
    setName);
22  bool removeGestureSet(const string & touchObjectName, const string &
    setName);
23  bool enableGesture(const string & touchObjectName, const string &
    gestureId);
24  bool DisableGesture(const string & touchObjectName, const string &
    gestureId);
25  void processFrame();
26  std::vector<gwc::PointEvent> consumePointEvents();
27  std::vector<gwc::GestureEvent> consumeGestureEvents();
28  bool assignTouchPoint(const string & touchObjectName, int pointId);
29  void addTouchEvent(TouchPoint touchEvent);
30  int getScreenWidth() const;
31  int getScreenHeight() const;
32  HWND getHWND() const;
33  string getWindowName() const;
34  bool loadGMLFile(string & pathGmlFile);
35 private:
36  int width;
37  int height;
38  HWND hWnd;
39  string wName;
40 };

```

Listing 4.25: GWC (GWTouch.cpp)

---

```

1  ...
2  bool GwTouch::Register(HWND hwnd)
3  {
4      return gwc::registerWindowForTouch(hwnd);
5  }
6  bool GwTouch::RegisterByWindowName(const string & windowName)
7  {
8      return gwc::registerWindowForTouchByName(windowName);
9  }
10 bool GwTouch::loadGMLFile(string & pathGmlFile)
11 {
12     return gwc::loadGML(pathGmlFile);
13 }
14
15 void GwTouch::init(int screenWidth, int screenHeight, string & pathGmlFile
    )
16 {
17     int lgw = gwc::loadGestureWorks(GWDL);
18     bool lgml = loadGMLFile(pathGmlFile);
19     width = screenWidth;
20     height = screenHeight;

```

```

21   gwc::initializeGestureWorks(width,height);
22 }
23
24 void GwTouch::resize(int screenWidth, int screenHeight)
25 {
26   gwc::resizeScreen(screenWidth,screnHeight);
27 }
28 void GwTouch::registerTouchObject(const string & objectName)
29 {
30   gwc::registerTouchObject(objectName);
31 }
32 bool GwTouch::deregisterTouchObject(const string & objectName)
33 {
34   return gwc::deregisterTouchObject(objectName);
35 }
36 ...
37 std::vector<gwc::PointEvent> GwTouch::consumePointEvents()
38 {
39   return gwc::consumePointEvents();
40 }
41 std::vector<gwc::GestureEvent> GwTouch::consumeGestureEvents()
42 {
43   return gwc::consumeGestureEvents();
44 }

```

## 4.2 OGRE

Given that the objective is to perform 3D navigation tests, there was a need to use some type of 3D engine. Originally, OpenGL was tested. Given the nature of OpenGL, which has a close relationship with the GPU, it was a very likely candidate. For example, a big cube of color spheres was tested in OpenGL, as shown in Figure 4.9. While OpenGL provided a great way to work with graphics, there was a need to be able to define virtual worlds. Without using a full-blown game engine, there was a need to have a graphics engine that provided a bit more help, in terms of regular housekeeping tasks, which in game engines are taken for granted. For example, it was desirable to have a framework that would provide a scene graph (as described in Chapter 2) and collision detection, while still keeping the

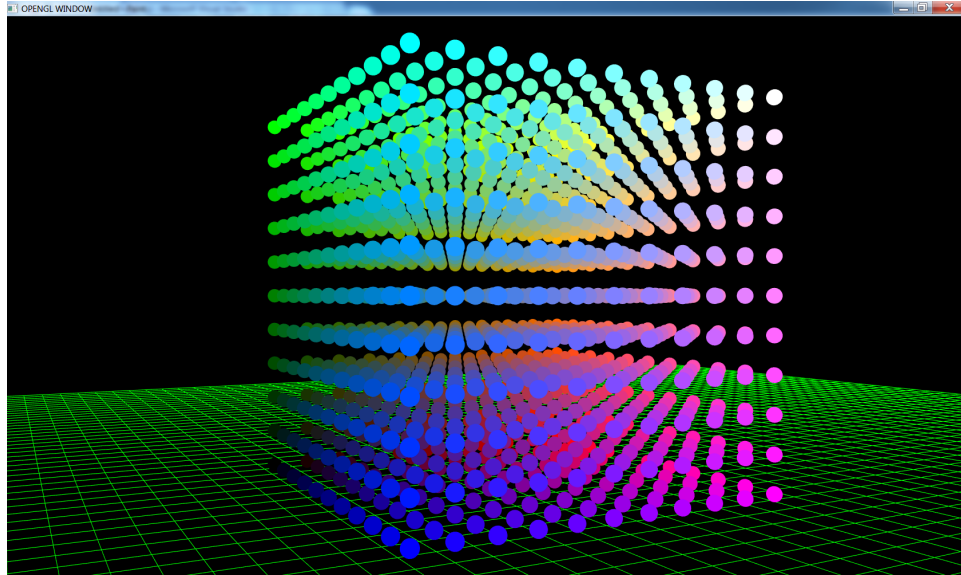


Figure 4.9: OpenGL Cube of Spheres

freedom to code in C++. OGRE was selected because it provided a very capable graphics engine that could run under OpenGL or DirectX. The engine contains a scene controller, as well as other tools. In addition, it used a third-party exporter, called OgreMax<sup>18</sup>, which allows exporting complete scenes from 3DS Max and Maya. This allows a scene to be built with one of those types of software and then exported into a **.scene** (see Listing 4.26) file created by OgreMax. Figure 4.10 shows 3Ds Max scene, which can be exported to a **.scene** file using OgreMax. At the point that the development started for 3DNav, OGRE provided all the components needed for 3D navigation, except advanced collision detection (but it can be provided using the Bullet physics engine).

### **OGRE: The Implementation**

This section outlines the essential issues pertaining to the implementation of 3DNav using OGRE. The OGRE engine is a very well-documented graphics engine. Most of the

---

<sup>18</sup>See <http://www.ogremax.com>.



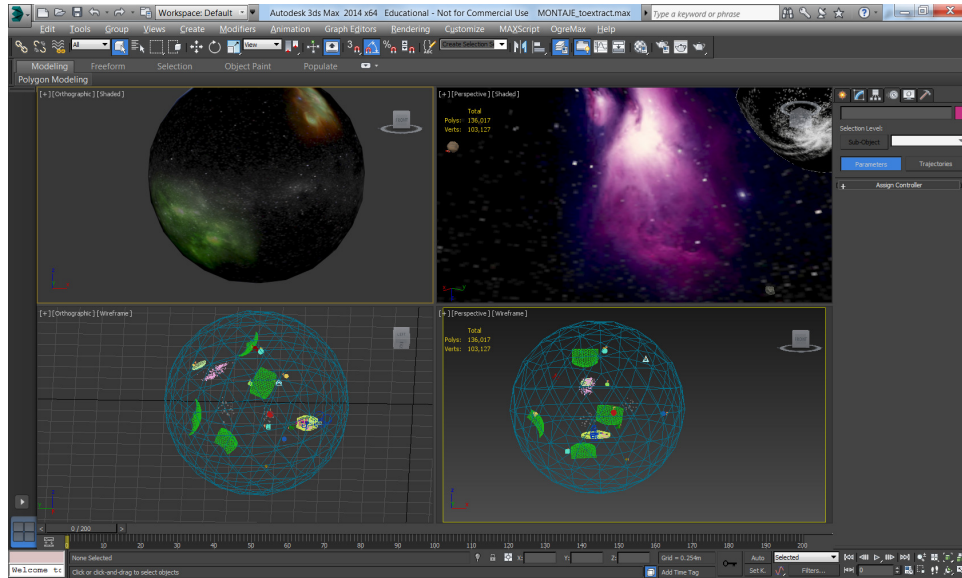


Figure 4.10: 3DS Max Scene

documentation is found in its online wiki<sup>19</sup>, with some additional information found in [65, 100, 109]. 3DNav used the Advanced Ogre Framework<sup>20</sup> to provide a simplified solution to different states of the experiment. This framework facilitated the use of the OGRE engine for things such as start-up for the engine, and different stages of the experiment, among other functionalities. The Advanced Ogre Framework provided a state machine to control the different levels of the process. In the case of the experiment, these were menu state (initial state) and game state (experiment mode). A partial listing is provided for the advanced framework definition, as shown in 4.27, and the definition of the menu and game states, shown in Listings 4.28 and 4.29, respectively.

The applications states (menu and game), shown in Listings 4.28 and 4.29, have a few items in common. First, they have access to an advanced framework singleton [56] object. In addition, the states share a common interface as shown in Listing 4.30. This provides developers with enter, exit, pause, resume, and update methods. The most important meth-

<sup>19</sup>See <http://www.ogre3d.org/tikiwiki/tiki-index.php>.

<sup>20</sup>See tutorial in <http://www.ogre3d.org/tikiwiki>.

ods here are the enter, which provides with the initial configuration needed for the given state (e.g., experiment setup), the exit, to clean up the state, and the update method. The update method is what fires for every frame that is rendered. The input listeners are also shared across states whenever it makes sense. For example, the keyboard is useful in any state (menu and experiment modes).

3DNav used the Object-Oriented Input System (OIS)<sup>21</sup>, which provides basic input functionalities, to enable the keyboard and mouse. However, the rest of the input, as already described in this chapter, used the WINAPI (or external libraries). A partial listing showing the wiiMote (lines 7-8), and the 3D mouse (lines 13-22), is shown in listing 4.31. This is executed in the **OgreFramework** class (from the Ogre Advanced Framework), in a method called *initOgre(...)*. Once OGRE is rendering, for each frame, there needs to be a check if new input event data is available. If it is, then the system fires the appropriate events, as described in previous sections. This happens in the **AppStateManager** class, in a method called *start(...)*. Partial code is shown in Listing 4.32.

A few additional items are important in reference to OGRE for the 3DNav prototype. Collision detection was achieved using bounding volumes provided by the SDK. An example is shown in Listing 4.33. Additional testing was performed using the Bullet physics engine, but it was not used during the experiment. For more information about collision detection, refer to Chapter 2. Finally, to achieve GUI buttons and windows in OGRE, the library called MyGui<sup>22</sup> was used.

Listing 4.26: Scene Nodes (Partial XML file)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <scene formatVersion="1.0" upAxis="y" unitsPerMeter="1" unitType="meters"
   minOgreVersion="1.8" ogreMaxVersion="2.6.1" author="OgreMax Scene
   Exporter (www.ogremax.com)" application="3DS Max">
3   <environment>
```

<sup>21</sup>See <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=OIS>.

<sup>22</sup>See <http://mygui.info>.

```

4     <colourAmbient r="0.333333" g="0.333333" b="0.333333"/>
5     <colourBackground r="0" g="0" b="0"/>
6     <clipping near="0" far="2540"/>
7 </environment>
8 <nodes>
9     <node name="SPACE">
10        <position x="0" y="0" z="0"/>
11        <scale x="0.5" y="0.5" z="0.5"/>
12        <rotation qx="0" qy="0" qz="0" qw="1"/>
13        <entity name="SPACE" castShadows="true" receiveShadows="true"
14            meshFile="SPACE.mesh">
15            <subentities>
16                <subentity index="0" materialName="Material#0"/>
17            </subentities>
18        </entity>
19    </node>
20    <node name="planet_with_craters">
21        <position x="8.23713" y="69.6509" z="-171.328"/>
22        <scale x="0.210295" y="0.210295" z="0.210295"/>
23        <rotation qx="0" qy="0" qz="0" qw="1"/>
24        <entity name="planet_with_craters" castShadows="true"
25            receiveShadows="true" meshFile="planet_with_craters.mesh"
26        >
27            <subentities>
28                <subentity index="0" materialName="
29                    planet_with_craters"/>
30            </subentities>
31        </entity>
32    </node>
33 </nodes>
34 </scene>

```

Listing 4.27: OGRE (Advanced Framework (h))

```

1 class OgreFramework : public Ogre::Singleton<OgreFramework>, OIS::
2     KeyListener, OIS::MouseListener, Ogre::FrameListener, Ogre::
3     WindowEventListener, Win3DXOgreListener
4 {
5     public:
6     OgreFramework();
7     ~OgreFramework();
8     bool initOgre(Ogre::String wndTitle, OIS::KeyListener *pKeyListener =
9         0, OIS::MouseListener *pMouseListener = 0);
10    void updateOgre(double timeSinceLastFrame);
11    bool keyPressed(const OIS::KeyEvent &keyEventRef);
12    bool keyReleased(const OIS::KeyEvent &keyEventRef);
13    bool mouseMoved(const OIS::MouseEvent &evt);
14    bool mousePressed(const OIS::MouseEvent &evt, OIS::MouseButtonID id);

```

```

12  bool mouseReleased(const OIS::MouseEvent &evt, OIS::MouseButtonID id);
13  bool handleNavigator(int message,DOFSix & D, BData & B);
14 private:
15  void windowResized(Ogre::RenderWindow* rw);
16  bool frameStarted(const Ogre::FrameEvent& evt);
17  bool frameEnded(const Ogre::FrameEvent& evt);
18 public:
19  Ogre::Root*          m_pRoot;
20  Ogre::RenderWindow* m_pRenderWnd;
21  Ogre::Viewport*     m_pViewport;
22  ...
23 private:
24  ...
25  BOOL InitRawDevices(void);
26  void FreeRawInputDevices(void);
27  OgreFramework(const OgreFramework&);
28  OgreFramework& operator= (const OgreFramework&);
29 };

```

Listing 4.28: OGRE (Menu State (h))

---

```

1  class MenuState : public AppState
2  {
3  public:
4  MenuState();
5  DECLARE_APPSTATE_CLASS(MenuState)
6  void enter();
7  void createScene();
8  void exit();
9  bool keyPressed(const OIS::KeyEvent &keyEventRef);
10 bool keyReleased(const OIS::KeyEvent &keyEventRef);
11 bool mouseMoved(const OIS::MouseEvent &evt);
12 bool mousePressed(const OIS::MouseEvent &evt, OIS::MouseButtonID id);
13 bool mouseReleased(const OIS::MouseEvent &evt, OIS::MouseButtonID id);
14 bool handleNavigator(int message,DOFSix & D, BData & B);
15 void notifyMouseButtonClick(MyGUI::Widget* _sender);
16 void notifyComboAccept(MyGUI::ComboBox* _sender, size_t _index);
17 void notifyMessageBoxResult(MyGUI::Message* _sender, MyGUI::
    MessageBoxStyle result);
18 void notifyQuestionBoxResult(QuestionPanel* _sender, QuestionPanelStyle
    result);
19 void update(double timeSinceLastFrame);
20 private:
21 bool m_bQuit;
22 };

```

Listing 4.29: OGRE (Game State (h))

---

```

1 class GameState : public AppState
2 {
3 public:
4     GameState();
5     DECLARE_APPSTATE_CLASS(GameState)
6     void enter();
7     void exit();
8     bool pause();
9     void resume();
10    void moveCamera();
11    void getInput(const double timeSinceLastFrame, const unsigned long long
        cCount);
12    void getWiiMoteInput();
13    bool getGamePad();
14    bool keyPressed(const OIS::KeyEvent &keyEventRef);
15    bool keyReleased(const OIS::KeyEvent &keyEventRef);
16    bool mouseMoved(const OIS::MouseEvent &arg);
17    bool mousePressed(const OIS::MouseEvent &arg, OIS::MouseButtonID id);
18    bool mouseReleased(const OIS::MouseEvent &arg, OIS::MouseButtonID id);
19    bool handleNavigator(int message, DOFSix & D, BData & B);
20    void onLeftPressed(const OIS::MouseEvent &evt);
21    void onRightPressed(const OIS::MouseEvent &evt);
22    void update(double timeSinceLastFrame);
23 private:
24    void createScene();
25    void createCamera();
26    void wiiMotionPlus(const wiimote const & remote);
27    double vPrime(double v, double lambda, double vmax);
28    void notifySentenceQuestionPanelResult(SentenceQuestionPanel* _sender,
        QuestionPanelStyle result);
29    ...
30 };

```

Listing 4.30: OGRE (App State Interface)

```

1 class AppState : public OIS::KeyListener, public OIS::MouseListener,
        public Win3DXOgreListener
2 {
3 public:
4     static void create(AppStateListener* parent, const Ogre::String name)
        {};
5     void destroy() { delete this;}
6     virtual void enter() = 0;
7     virtual void exit() = 0;
8     virtual bool pause(){return true;}
9     virtual void resume(){};
10    virtual void update(double timeSinceLastFrame) = 0;
11 protected:

```

```

12 AppState(){};
13 AppState* findByName(Ogre::String stateName){return m_pParent->
    findByName(stateName);}
14 void changeAppState(AppState* state){m_pParent->changeAppState(state);}
15 bool pushAppState(AppState* state){return m_pParent->pushAppState(state
    );}
16 void popAppState(){m_pParent->popAppState();}
17 void shutdown(){m_pParent->shutdown();}
18 void popAllAndPushAppState(AppState* state){m_pParent->
    popAllAndPushAppState(state);}
19
20 ...
21 AppStateListener* m_pParent;
22 Ogre::Camera* m_pCamera;
23 Ogre::SceneManager* m_pSceneMgr;
24 };

```

Listing 4.31: OGRE (InitOgre)

```

1  Ogre::LogManager* logMgr = new Ogre::LogManager();
2  m_pLog = Ogre::LogManager::getSingleton().createLog("OgreLogfile.log",
    true,true,false);
3  m_pLog->setDebugOutputEnabled(true);
4  m_pRoot = new Ogre::Root(mPluginsCfg);
5  if (!m_pRoot->showConfigDialog())
6      return false;
7  m_wiiMote.ChangedCallback = on_wiimote_state_change;
8  m_wiiMote.CallbackTriggerFlags = (state_change_flags)(CONNECTED |
    EXTENSION_CHANGED | MOTIONPLUS_CHANGED);
9  m_pRenderWnd = m_pRoot->initialise(true, wndTitle);
10 size_t hWnd = 0;
11 OIS::ParamList paramList;
12 m_pRenderWnd->getCustomAttribute("WINDOW",&hWnd);
13 if (!InitRawDevices())
14 {
15     OgreFramework::getSingletonPtr()->m_pLog->logMessage("[3DX] Error
        InitRawDevices()");
16     m_bRawDevicesOn = false;
17 }
18 else
19 {
20     m_bRawDevicesOn = true;
21     Win3DXOgreEventRegistry::registerListener(Win3DXOgreEventRegistry::
        LISTEN_ALL,this);
22 }
23 ...
24 m_pRoot->addFrameListener(this);
25 Ogre::WindowEventUtilities::addWindowEventListener(m_pRenderWnd, this);

```

```
26 ...
27 return true;
```

Listing 4.32: OGRE (Check Input)

```
1 ...
2 while (!m_bShutdown)
3 {
4     if(OgreFramework::getSingletonPtr()->m_pRenderWnd->isClosed())
5         m_bShutdown = true;
6     if(OgreFramework::getSingletonPtr()->m_pRenderWnd->isActive())
7     {
8         MSG msg;
9         HWND hwnd;
10        bool navOn = OgreFramework::getSingletonPtr()->m_bRawDevicesOn;
11        OgreFramework::getSingletonPtr()->m_pRenderWnd->getCustomAttribute("
            WINDOW", (void*)&hwnd);
12        if( navOn && PeekMessage( &msg, hwnd, 0, 0, PM_REMOVE ) )
13        {
14            if(msg.message == WM_INPUT)
15            {
16                OgreFramework::getSingletonPtr()->ProcessWM_INPUTEvent(msg.lParam
                    );
17            }
18            else
19            {
20                TranslateMessage( &msg );
21                DispatchMessage( &msg );
22            }
23        }
24    }
25    Ogre::WindowEventUtilities::messagePump();
26
27    if(OgreFramework::getSingletonPtr()->m_pRenderWnd->isActive())
28    {
29        ... //D0 OGRE work.
30    }
31    else
32    {
33        Sleep(1000);
34        sleep(1);
35    }
36 }
37 ...
```

Listing 4.33: OGRE (Collision)

```
1 void CollisionTrigger::onUpdate()
```

```

2 {
3   if(!_nodeA || !_nodeB)
4     return;
5   updateActions();
6   if(_nodeA->_getWorldAABB().intersects(_nodeB->_getWorldAABB()))
7     onCollision();
8   else
9     cancelActions();
10 }
11 void CameraCollisionTrigger::onUpdate()
12 {
13   if(!_camera || !_nodeB)
14     return;
15   updateActions();
16   if(_nodeB->_getWorldAABB().intersects( Ogre::Sphere(_camera->
17     getDerivedPosition(), _cameraRadius) ))
18     onCollision();
19   else
20     cancelActions();

```

### 4.3 ECHoSS: Experiment Module

The experiment module, called Experiment Controller Human Subject System (ECHoSS), contained a series of C++ classes and additional configuration files, designed for 3DNav, to make the environment suitable for human-subject testing in the area of 3D navigation. This section describes the components that make 3DNav into an experiment system. The experiment module is composed of a few sub-modules. Those are the experiment controller, experiment device, experiment task, and experiment search object. With this, it is enough to specify a running experiment.

The experiment controller, shown in Listing 4.34, controls a human-subject test. It is composed of devices (e.g., multi-touch) and tasks (e.g., objects to find). The experiment controller can be built by adding devices using *AddExperiment(...)* or by adding training devices by using *AddTraining(...)*. Actually, a device can be added by using *AddDevice(...)*, since the device can specify if it is a training device or treatment device.



Another important method that must be called after starting the experiment (*start(...)*) is the *processNext()* method, which allows the system to go from one device to the other. Depending on the initialization of the experiment controller, the queue holding the devices is determined as follows:

- Training devices will be the first devices to be placed in the queue.
- If controller is not responsible of randomizing the devices<sup>23</sup>, the order of devices will be set by their insertion.
- If controller is responsible of randomizing the devices, then it will perform a random sorting of devices, sorting separately the training devices from the treatment devices.

The experiment device class, as shown in listing 4.35, allows the control of each treatment (e.g., input device) that will handle a series of tasks. For example, in the case of the experiment, the user must search for five objects. This means that each device contains a series of objectives, defined in the experiment tasks class. This class, as shown in Listing 4.36, provides the definition for a task. In particular, the tasks used in this example add the keyboard task, in addition to the object to be found. However, this is an example for this particular case, since the developer can modify the settings. The actual concrete implementation of the task is defined in the search object class, as shown in Listing 4.37. The specific search object made use of the entities found in OGRE to keep track of the object in question and the marker (e.g., flag) that provides visual feedback to the user. An important part of the task is that it also has knowledge where to place the object to be searched. This allows the experimenter to move the objects, depending on the treatment and requirements of the experiment.

ECHoSS provides a way to create generic human-subject tests, while keeping the requirements separate from the actual 3D implementation. It is the objective of this part

---

<sup>23</sup>The author chose this option for the experiment.

of 3DNav to allow for further studies, providing the developer with more flexibility when using a 3D navigation task.

Listing 4.34: Experiment Controller

```
1 class ExperimentController : public ...
2 {
3 public:
4     typedef boost::system::error_code error_code;
5     typedef enum {Init,Started,Device_Running,Device_Exited,
6         Device_Exit_By_Time_Expired,Stopped} experiment_state;
7     typedef enum {None,Training,Treatment} experiment_mode;
8     ...
9     ExperimentController(const string & subjectName,const string &
10         experimentName, bool shuffleDevices=true, bool, shuffleTraining=
11         true);
12     ~ExperimentController();
13     bool AddExperiment(const ExperimentDevice & experimentDevice);
14     bool AddTraining(const ExperimentDevice & experimentDevice);
15     bool AddDevice(ExperimentDevice & experimentDevice);
16     bool start();
17     bool stop();
18     bool pause();
19     bool resume();
20     bool abort();
21     void update(const double timeElapased,const unsigned long long cCount);
22     virtual bool handleExperiment(...);
23     bool isExperimentStarted(){return _guard.mStarted;}
24     bool isDeviceRunning();
25     bool isPaused() { return _guard.mPaused;}
26     bool processNext();
27     bool isNextAvailable();
28     bool getCurrentDevice(ExperimentDevice & dev);
29     string getExperimentName() { return mExperimentName;}
30     inline size_t getTotalObjectCount() { return mTotalObjectCount;}
31     inline bool hasStarted() { return _guard.mStarted;}
32 private:
33     ...
34 };
```

Listing 4.35: Experiment Device

```
1 class ExperimentDevice : public ...
2 public:
3     ...
4     typedef enum {PreInitialized,Initialized,Running,Completed,
5         CompletedByExpiredTime,Aborted,Paused} device_state;
```

```

5   typedef enum {MultiTouch,GamePad,Mouse,Keyboard,Mouse3D,WiiMote,
      LeapMotion,Generic} device_type;
6   ExperimentDevice(ExperimentTimerPtr timer,string & deviceName,const
      size_t order,bool isTrainingDevice=false);
7   ExperimentDevice(string & deviceName,const size_t order, bool
      isTrainingDevice=false);
8   typedef vector<ExperimentTask> tasks;
9   ...
10  ExperimentDevice();
11  ~ExperimentDevice();
12  bool start();
13  bool isInitialized();
14  bool addTask(ExperimentTask & task);
15  bool exit();
16  bool pause();
17  bool abort();
18  virtual void update(double timeElapsed,unsigned long cycleCount);
19  bool isTrainingDevice() { return mIsTrainingDevice;}
20  //define copy constructor and equal operator
21  ...
22  virtual bool handleExperiment(...);
23  inline const size_t getOrder() { return mOrder;}
24  inline const size_t getFoundObjectCount() { return mFoundObjectCount;}
25  inline const size_t getFoundObjectWithSentenceCount() { return
      mFoundObjectWithSentenceCount;}
26  inline string getDeviceType() { return mDeviceType;}
27  inline void setDeviceType(const string & deviceType){mDeviceType =
      mDeviceType;}
28  ...
29  };

```

Listing 4.36: Experiment Task

```

1  class ExperimentTask : public ...
2  {
3  public:
4      typedef struct view_data
5      {
6          ...
7          Ogre::Vector3 position;
8          Ogre::Quaternion orientation;
9          Ogre::Vector3 direction;
10     }view_data;
11     typedef struct location_data
12     {
13         ...
14         Ogre::Vector3 position;
15         Ogre::Quaternion orientation;

```

```

16     }location_data;
17     typedef struct object_data
18     {
19         ...
20         view_data camera;
21         location_data object;
22         location_data object_flag;
23         bool enabled;
24     }object_data;
25     ExperimentTask(SearchObject * searchObject,const std::string &
26         taskName, const object_data & objectData=object_data());
27     virtual ~ExperimentTask();
28     typedef enum task_state {Init,Started,Stopped,Aborted,Unkown};
29     void start();
30     void stop();
31     void abort();
32     bool isTaskFinished();
33     bool isHit(){ return mHit;}
34     bool isFound(){ return mFound;}
35     task_state getTaskState() { return mTaskState;}
36     bool isEnabled(){return mIsEnabled;}
37     void setEnabled(bool enabled){enabled = mIsEnabled;}
38     virtual void searchEvent(SearchObject::SearchObjectListener::EventType
39         type, SearchObject* sender, map<string,string> userData);
40     SearchObject * getSearchObject();
41     void update(double TimeSinceLastFrame,unsigned long cycleCount);
42     string getTaskName(){return mTaskName;}
43     string getKeyboardSentence() { return mKeyboardSentence;}
44     void setKeyboardSentence(const std::string & str) { mKeyboardSentence=
45         str;}
46     void reset();
47     ...
48 };

```

Listing 4.37: Experiment Search Object

```

1 class SearchObject
2 {
3 public:
4     typedef struct view_data
5     {
6         ...
7         Ogre::Vector3 position;
8         Ogre::Quaternion orientation;
9         Ogre::Vector3 direction;
10    }view_data;
11    typedef struct score_data
12    {

```

```

13     ...
14     int score;
15     bool objectFound;
16     bool objectHit;
17     bool flagFound;
18     bool flagHit;
19 }score_data;
20
21 SearchObject(Ogre::SceneNode* node, Ogre::SceneNode* flagNode);
22
23 virtual ~SearchObject();
24
25     Ogre::SceneNode* getNode();
26     Ogre::SceneNode* getFlagNode();
27
28
29     bool isFound();
30     bool isHit();
31 void update(double timeSinceLastFrame);
32     ...
33 };
34 extern SearchObjects* SearchObjectsCollection;

```

## 4.4 Overview

This chapter provided a description of the important implementations for 3DNav. It presented enough information about the implementation, while leaving the concrete experiment settings for Chapter 5. 3DNav has enough flexibility built-in that it could be customized in the future for different versions. Furthermore, it established a path for how to create a full-fledged 3D input UI experiment API.

## CHAPTER 5

### DESIGN OF EXPERIMENT: 3D NAVIGATION

This section details the design of experiment for 3D navigation using the multi-touch display. This includes the choices of input devices, the subject population, and the different decisions made before performing the experiment. The results, statistical analysis, and discussion are explored in the following chapters. Therefore, this chapter concentrates on the actual experimental design and procedures. Finally, it is important to note that the experiment was approved by Florida International University (FIU) IRB. The corresponding approved memos are shown in Appendix C.

#### 5.1 Experiment Objective

Defining the goals of the experiment was imperative to have a well-designed human-subject test. The primary objective was to see if there are any significant differences when using the 3M M2256PW multi-touch display and the Xbox 360 GamePad for 3D navigation (with 6-DOF). Intrinsic to this question, there was the need to know if any co-factors could influence the results for these two devices. The following list summarizes the questions to be answered through the experiment:

1. Is there any statistical difference between multi-touch display and the GamePad controller when searching for objects in a virtual 3D world?
2. Are there any co-factors that may show a statical difference in reference to item 1?
3. Is there any statical difference when switching from multi-touch to keyboard, versus GamePad to keyboard?
4. Do subjective questionnaires validate the finding of the objective data?

ID	Type of User	Visits
A	Regular iPhone and iPad User. Little experience with video game console	5
B	Regular PC user. No video game console experience. iPad user	3
C	Game developer and experienced game player	2
D	Regular iPhone and MAC user. Little experience with video game console	2
E	Multi-touch user. iPad and iPhone User. Experienced game user.	6

Table 5.1: Pre-Trial Users.

## 5.2 Pre-Trials

Before proceeding with the experiment, this author asked a few subjects with different skills to test a preliminary version of the experimental protocol. The pre-trial users were five, as shown in Table 5.1. This table describes each user (identified by a letter). Each user spent between 30 minutes to up to 4 hours per visit. The pre-trials were very useful to finalize the design of experiment. The decisions for multi-touch gestures, multi-touch and GamePad mappings, visual cues, and the reset button, among others, were derived from by pre-trial feedback. Later in the chapter, those decisions are discussed.

## 5.3 Device Selection

After testing various devices (see Chapters 3 and 4), the GamePad controller was selected as the device to compare to the multi-touch display, for the 3D navigation tasks. Originally, the device selected was one that provided a full 6-DOF, such as the 3D mouse. The problem

was that the mouse did not provide a simple means of navigation in the early test of 3DNav. After various iterations of devices, the GamePad, which has been perfected for more than 30 years, was selected. A possible alternative would have been a vision-based solution, such as the Microsoft Kinect. However, given that this dissertation was not focusing on computer vision, and the Kinect pre-trials we had were not as successful as first expected, the GamePad was chosen instead.

The other decision that needed to be made was how to detect the gesture when using the multi-touch display. While FETOUCH worked well for the gestures that were originally built into it, such as swipe, scale, and rotate, it didn't have enough gestures 3D navigation. The possibility of extending it was an option, but it was also considered that this author was looking for a more unbiased experiment when comparing devices. This meant that using a standard, commercial gesture recognition software would provide a better comparison. Another factor that helped this decision is that early work in a new prototype suffered from similar problems as the one exhibited by GWC (describe later). Therefore, GWC API by ideum was selected. This, in this author's opinion, would provide the best method for the experiment, because GWC is a mainstream framework for multi-touch recognition. Furthermore, GWC would require a contribution by this author to remove gesture detection conflicts (using Yield).

## **5.4 Experimental Subjects**

During the initial design of the experiment, the only requirement was the age of the subjects, who were required to be between 18 and 65 years old. This allowed the range of possible subjects available to the experiment to be quite large. No prior experience was required and a high level of experience with games was not a disqualifying factor.





Figure 5.1: 3M M2256PW multi-touch display

## 5.5 Experiment Apparatus

The experiment apparatus consisted of various pieces of hardware and software. This section describes the actual pieces used for the experiment. This included the actual configuration of 3DNav and details important to reproduce the experiment if needed.

### 5.5.1 Hardware Setup

The equipment used for the experiment consisted of a few pieces of hardware. This included a PC, multi-touch display (Figure 5.1), keyboard, mouse, GamePad (Figure 5.2), higher-quality GPU, stereo speakers, and numeric keypad. The following list details each component:

- Dell Precision T3500 PC.
  - Intel Xeon central processing unit (CPU) W3530 2.8 giga Hertz (GHz).
  - Four-core CPU.
  - 12 gigabyte (GB) random-access memory (CPU), 1333 mega Hertz (MHz).
  - Two 500 GB hard drives.
- AMD ATI FirePro V7800 (FireGL).



Figure 5.2: Xbox 360 GamePad

- GPU 1440 processor.
- 4096 megabyte (MB) RAM.
- GMYLE Super Slim USB 2.0 Mini Keyboard, as shown in Figure 5.4a.
- Standard PC Mouse.
- 3M M2256PW multi-touch display.
  - 22-Inch monitor.
  - 20 independent touches<sup>1</sup>.
  - Maximum Resolution 1680x1050 at 60 Hz.
- Perixx PERIPAD-201B, Numeric Keypad.
- Microsoft Xbox 360 Wireless Controller for Windows.
  - This includes USB wireless receiver.

---

<sup>1</sup>However, Windows 7 reports 30 independent touches.

## 5.5.2 Software Setup

The software utilized for the design of experiment included Windows 7, third-party libraries, Microsoft Visual Studio 2012, and solutions developed by this author. The following is a detailed list of the software required<sup>2</sup> to replicate this experiment:

- Microsoft Windows 7 64 bit (Service Pack 1).
- Microsoft Visual Studio Ultimate 2012 (Update 4).
- OGRE engine, version 1.9.
- GWC API.
- Research applications.
  - 3DNav.
  - Yield.
  - ECHoSS.
  - FaNS.
- DirectX 11 and OpenGL 4.1.
- DirectX SDK June 2010.
- MyGui API, version 3.2.
- OpenAL (audio library).

The GWC and the research modules have been described in Chapters 3 and 4. There are a few aspects that must be expanded, which relate to the design of the experiment. In specific, the setup required to have the experiment running with ECHoSS. The other research modules worked as described in the previous chapters. The gesture selection in GWC is detailed later in 5.6.

---

<sup>2</sup>Higher version of the software listed should also work but may require small changes.

## Setting ECHoSS in 3DNav

To set up ECHoSS in 3DNav, a configuration file (`game.cfg`) is provided. The configuration file is processed in the `GameLogicController` class. However, there are some important aspects that are relevant to this experiment, which shows how the experiment was configured.

As stated before, the system determines if the randomization is done at the programming level or at the configuration level. In the case of this experiment, the randomization is done at the configuration levels by setting the **UseRandom** variable to 0, as shown in Listing 5.1, line 2.

In the **Devices** section in Listing 5.1, line 3, the actual input devices were configured. For this particular experiment, two devices for training and two for treatment were created using the gamepad and multi-touch input systems. Later, in the **Tasks** sections in line 9, all the the tasks were configured. This included the training tasks and the treatment tasks. Note that the second parameter for the task is a name for a set, which allowed to group the tasks for each device.

This makes more sense when the section **Experiment\_Permutations** is explained. This starts in Listing 5.1, line 18, where three important variables are defined. The first one, **DevicePermutation**, defines the name of the permutation, and then how the permutation should behave. For example, for permutation  $y$  in line 2, the set contains  $y = \{\text{gamepad training, multi-touch training, multi-touch treatment, gamepad treatment}\}$ . The **TaskPermutations** follows a similar logic, but uses the set of tasks. Finally, the **UserPermutation-Count** is the number of how many permutations are required to have a complete set. The reason for the 16 permutations is because when there are 2 devices, 2 training devices, 2 sets of tasks, and 2 sets of training tasks, the numbers of possible permutations is 16.

The tasks, defined starting line 9 in Listing 5.1, show a few examples of the actual tasks given in the experiment. In general, and it is true for this configuration file, besides the

normal key/value assignment in a typical initialization (INI)<sup>3</sup> file, a hash table is produced for values that are repeated. This is true for the **tasks** section. The following description provides the specifications for a task defined in game.cfg, in order of values separated by commas:

1. Object Name.
2. Set Name.
3. Object Position. 3D vector.
4. Orientation. Quaternion.
5. Position of marker (flag). 3D vector.
6. Orientation of marker (flag). Quaternion.
7. Position of camera for automatic viewing. 3D vector.
8. Orientation of camera for automatic viewing. Quaternion.
9. Direction of camera for automatic viewing. 3D vector.
10. Keyboard sentence to be typed by subject.

Listing 5.1: 3DNAV game.cfg (Experiment Setup)

---

```
1 [Experiment_Default]
2 UseRandom = 0
3 [Devices]UseRandom = 0
4 DeviceTraining = 0, "Multi-Touch Training" , 0 , 300, 60 , "MT" , "
    TouchGesturePad" , "Multi-Touch"
5 DeviceTraining = 1, "GamePad Training", 0, 300, 60, "PAD", "XPAD", "Multi
    -Touch"
6 Device = 2, "Multi-Touch Treatment", 0, 300, 60, "MT", "TouchGesturePad",
    "GamePad"
7 Device = 3, "GamePad Treatment", 0, 300, 60, "PAD", "XPAD", "GamePad"
8 [Tasks]
9 UseRandom = 0
```

---

<sup>3</sup>See [http://en.wikipedia.org/wiki/INI\\_file](http://en.wikipedia.org/wiki/INI_file).

```

10 TaskTraining="hypercube", "X", -37.9252 -52.4117 -163.951, 1 0 0 0, -33.0
    -52.4 -164.0, 1 0 0 0, -34.3721 -51.0749 -146.844, 1 0 -0 -0 , 0 0
    -1, "Soccer is the greatest sport of the world."
11 #...
12 #set A
13 Task="creature_barrier", "A", 276.044 -302.008 -221.937, 1 0 0 0, 262.044
    -290 -220, 0.991445 0.130526 0 0, 308.293 -200.245 -58.013, 0.885552
    -0.340648 -0.301152 -0.095582, 0.468252 -0.660892 -0.586533, "I'm
    looking forward to Brazil 2014."
14 Task="BALL", "A", 64.5 99.5 -100.54, 1 0 0 0, 59 100 -100.54, 1 0 0 0,
    60.8526 99.8465 -80.827, 1 0 0 0, 0 0 -1, "I'm having to type short
    sentences."
15 #...
16 #set B
17 #...
18 [Experiment_Permutations]
19 DevicePermutation = "w", 0 , 1 , 2 , 3
20 DevicePermutation = "x", 0 , 1 , 3 , 2
21 DevicePermutation = "y", 1 , 0 , 2 , 3
22 DevicePermutation = "z", 1 , 0 , 3 , 2
23 TaskPermutation = "a", "X", "Y", "A", "B"
24 TaskPermutation = "b", "Y", "X", "A", "B"
25 TaskPermutation = "c", "X", "Y", "B", "A"
26 TaskPermutation = "d", "Y", "X", "B", "A"
27 UserPermutationCount=16

```

## 5.6 Multi-Touch Gesture Design

This section provides the definition of a subset of gestures that were considered, the gesture mapping for the 3D navigation, and explains the decision process for the selected gestures with their mappings.

### 5.6.1 Gesture Definition

Table 5.2 provides definitions for multi-touch gestures that were considered for the experiment. The definitions here are considered to be the base-line gestures, given that the

number of fingers required varies. For example, a more specific gesture could be a two-finger versus a three-finger swipe.

## 5.6.2 Gesture Selection

Using the definitions of Table 5.2, during the pre-trial phase (see 5.2), a series of test and informal questions were asked to find a set of gestures that would work best with 3D navigation. For the translation, which was used more frequently than rotations, users preferred to use a one-finger swipe gesture. For the rotations, the most intuitive use of the rotate gesture was for the roll<sup>4</sup> action. For the remaining rotations, a two-finger vertical swipe was mapped as the yaw action, and a two-finger horizontal swipe was mapped to the pitch action. This made sense to the users, as well as this author, during the pre-trials. The scale gesture was a candidate for translating on the Z axis. However, while this gesture is pervasive, the experimenter decided to use a bi-manual gesture, called **Hold-and-Roll** (described in Table 5.2). This decision was made for various reasons. First, the scale gesture seems to indicate to the user a zoom-in/zoom-out effect, which is common when using smart phones or tablets (e.g., iPhone). This was not the mapping required for this experiment, since translating is the action of moving in the positive or negative direction on the z axis. Second, the scale gesture could be more appropriate in 3D navigation for either scaling a single object required for visualization, or changing the viewing angle of the camera. The latter allows the frustum (see 2.1.1) to be altered, providing an additional DOF. Finally, the bi-manual interaction was an interesting approach, which will be explored further in the discussion part of this dissertation, in Chapter 7.

---

<sup>4</sup>See 2.1.2 for a review about yaw, pitch, and roll.

Name	Definition	Pervasive	Mode
Swipe	This gesture involves one or more fingers going in the same direction.	Yes	UP
Drag	Similar to swipe. Some may define this gesture a bit slower than swipe, as if it is dragging an object.	Yes	UP
Rotate	A gesture that requires two or more fingers in a circular direction.	Yes	UP
Scale	This gesture, also called zoom, is defined as the movement of two or more fingers, with at least one of them going in opposite direction (zoom out) or towards each other (zoom in).	Yes	BI
Tap	This gestures is similar to a mouse click. There may be single tap, double tap, or triple tap. It is defined as a touch within a time constraint.	Yes	UM
Hold	This gesture is similar to the tap, but the user does not lift the finger from the display.	Yes	UM
Flicker	This gesture involves one or more (usually two or three) fingers moving in the same direction for a short time, and moving all of them in the opposite direction, repeating this pattern n times. This creates a flickering effect.	No	UP
Tilt	This gesture involves two or more fingers. At least one of the fingers is moving to create a tilt effect. The gesture is meant to stay in place.	No	UP
Hold-and-Roll	This gesture, designed by this author, provides a bi-manual interaction to hold with the non-dominant hand (at least one finger) and roll with the dominant hand. The roll movement is considered to emulate the scroll wheel of a mouse.	No	BM

Table 5.2: Gesture Definitions.

	BM	Bi-Manual Only
	UM	Uni-Manual Only
Interaction Mode Legend	UP	Uni-Manual Preferred
	BP	Bi-Manual Preferred
	BI	Either BM or UM



Fingers	Gesture	Action
1	Swipe horizontal	Translate X
1	Swipe vertical	Translate Y
3	Hold-and-Roll bi-manual	Translate Z
1	Swipe diagonal	Translate X & Y.
2	Swipe horizontal	Yaw
2	Swipe vertical	Pitch
2	Rotate	Roll
2	Swipe diagonal	Disabled
n	Scale gesture	Disabled

Table 5.3: Gesture Mappings.

### 5.6.3 Gesture Mapping

Once the gestures were selected for this experiment, the mapping associating gestures to actions, as shown in Table 5.3, was created. This provided the human subjects with a 3D navigation using multi-touch enabling 6-DOF. Some constraints were established. First, diagonal translations were enabled for the X and Y axes. This was possible because the swipe gesture allowed for the user to move diagonally. For the Z axis, the **Hold-and-Roll** gesture was meant to be independent. The same case applied to the rotate gesture, because it provided an action for only one type of rotation. In the case of the pitch and yaw rotations, there was a possibility to provide dual rotation, since it used a two-finger swipe gesture. However, during the pre-trials, this created confusion in the users. Therefore, this was disabled from the interaction. To overcome the possibility of the user creating a small diagonal when he/she meant to do a horizontal or vertical swipe, in a rotation action, the system provided a way to adjust the swipe to the closest match (either vertical or horizontal.) In addition, different thresholds and noise reduction techniques were applied, which are part of Yield (see 3.4).

Control	Direction	Action	Analog
Left thumb-stick	Up/down	Translate y axis	Yes
Left thumb-stick	Left/right	Translate X axis	Yes
Left thumb-stick	Diagonal	Translate Y axis	Yes
Left/Right trigger	-	Translate Z axis	Yes
Right thumb-stick	Up/down	Pitch	Yes
Right thumb-stick	Left/right	Yaw	Yes
Left/Right shoulder	-	Roll	No

Table 5.4: Controller Mappings.

## 5.7 GamePad Design

The GamePad design was also tested during trials. In specific, one of the pre-trial users is a game developer, leader of the Miami Game Developer Guild<sup>5</sup>, and an experienced game player. He, along with the other users, helped to test the GamePad implementation for 3D navigation. This helped to make the design decisions for the experiment.

The Xbox 360 controller comes with two thumb-sticks. A thumb-stick is a small joystick that is designed for the use of a thumb. By providing two analog thumb-sticks (this has been standard, since the introduction of the Sony Dual Analog Controller and DualShock [129]), this type of dual-thumb-stick GamePad can provide a more accurate movement with 4-DOF<sup>6</sup>. In gaming, it is customary to prevent the character from looking up or down past a given angle [233, pp. 561–572]<sup>7</sup>. In some domain-specific scenarios, having less than 6-DOF is very suitable [201]. In the case for this experiment, 6-DOF were needed. Therefore, additional mapping was required, which is shown in Table 5.4. For an expanded picture of the Xbox 360 controller, with a description for each of the thumb-sticks, triggers and buttons, see Appendix D.

---

<sup>5</sup>See <http://www.gamedevelopersguild.com>.

<sup>6</sup>Using the additional buttons, is possible to have 6-DOF navigation.

<sup>7</sup>See also [131, Chapter 14] and [144, pp. 47–49].



Figure 5.3: Multi-Touch Reset Button

## 5.8 Additional Controller Design

In addition to the design in Sections 5.6 and 5.7, there are some additional settings required for the experiment. First, the reset button was created to come back to the original starting point, for the multi-touch and the GamePad controllers. This button was the red button (button B) on the XBox controller and a similar red button on top of the screen, shown in Figure 5.3, for the multi-touch case. Second, the mouse cursor needed to be disabled for the experiment. Human subjects were not given a mouse. Third, the experiment included the use of a keyboard. A notebook-like keyboard was used, which has become common because of notebooks<sup>8</sup>. The keyboard, as shown in Figure 5.4a , allowed users to type sentences when prompted by the experiment. Finally, the experimenter had a numeric keypad to control certain aspects of the experiment, as shown in Figure 5.4b.

---

<sup>8</sup>The Apple wireless keyboard has also become pervasive among Mac users.



(a) Experiment Keyboard

(b) Experiment Keypad

Figure 5.4: Additional Keyboard Input

## 5.9 Techniques

This section describes the techniques for these experiments and why they were chosen. In particular, the techniques for: finding objects in a 3D world (primed search), device switching, and visual cues. More details about the actual techniques are provided in Chapter 2.

### 5.9.1 Primed Search

Search was the task chosen for this experiment, as reviewed in Chapter 2 and described in [17]. The search technique was selected to measure the time taken with each device when finding the targets. This would have proven difficult under the exploration technique. Once the search technique was chosen, the options of using naïve and primed searches were evaluated. It is hard to say where the naïve interaction ends and the primed search begins (see Chapter 2).

If the user did know where the objects were located, which was an option tested in pre-trial, this would be a primed search. In the actual 3DNav, the functionality to show

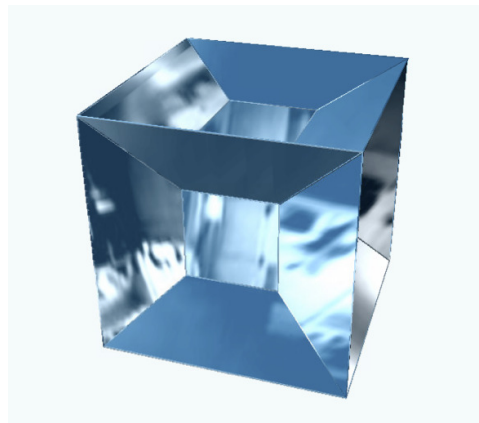


Figure 5.5: Hyper Cube for Training

where the objects were was provided by pressing the **V** option of the keypad<sup>9</sup> (see Figure 5.4b). Instead, the appearances of the five objects that the user needed to find were shown on a piece of paper, and the user was given a written clue about the location of one of the objects. However, to reduce a possible frustration factor, two objects were always very easy to spot, one less easy to spot, and two more difficult to find. During the training, the user did not see the actual objects, but was presented with a hypercube, as shown in Figure 5.5. The virtual world with the static objects was the same during the training and treatment phases. This meant that the user would know the world around it, without the actual targets. Therefore, this can be viewed as a naïve search with some primed search features (see Chapter 2).

### 5.9.2 Visual Cues

The visual cues that were provided were minimal. First, the subjects were told that the universe was surrounded by a big sphere, called the inner sphere. They were told that all the objects would be found inside the inner sphere. The users were also told that an outer sphere surrounded the inner sphere. Exiting the inner sphere would be allowed, given

---

<sup>9</sup>The actual key was keypad 7.



Figure 5.6: Search Object Marker (Flag)

that there was a space between both spheres. However, if the users found themselves too close to the outer sphere, the environment would trap them, forcing them to press the reset button. Pressing the reset button would produce a large penalty in distance traversed. It is important to note that while all the objects were inside the inner sphere, the user could not see them right away. This allowed the users to try to stay within the boundaries of the inner sphere. The decision to let them out of the traversal space (in the inner sphere) was done to keep navigation smooth, while users were in the boundaries.

The users were also told that the targets would have a flag, as shown in Figure 5.6. This flag would indicate that the object next to it was a target, as there were other objects in the virtual world that were not targets. Once a target was reached, its flag would disappear. For each treatment, all the search objects would start with their flags showing, and as they subject reach them, the flag would disappear. In addition to the flag, in the top right corner, the name of the targets would appear, written in red, once they were reached.

Another visual cue given to the users was a paper handout, which included the search objects. In here, the handout also included also a tip, indicating that one of the objects would be found near a purple nebula. The users were not told about the other three nebulas in the virtual world, which were: yellow, green, and blue (see Figure B.1 in Appendix B).

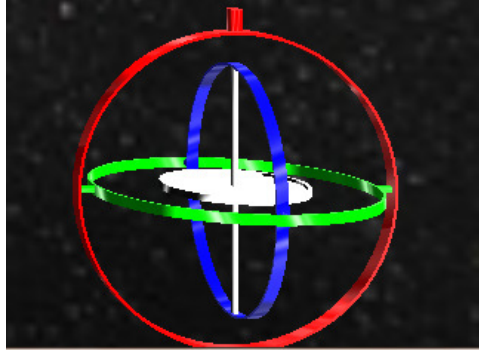


Figure 5.7: Three-Ring Sphere

Note that the user did not know the location of the object. Finally, the other non-target objects provided a cue for navigation.

Additionally, the users were provided with a position indicator, in the bottom center of the screen. This provided the X, Y, and Z coordinates. The users were not provided with the rotation angles. However, 3DNav provided a three-ring sphere for reference. The three-ring sphere looked similar to a gimbal<sup>10</sup> [139, pp. 314-315].

### **Three-ring Sphere**

The three-ring sphere, which appears visually like a gimbal, was envisioned to provide a rotational indicator. This was composed of four components: an outer ring (red), a middle ring (green), an inner ring (blue), and a center disk connecting the inner ring. This three-ring sphere is shown in Figure 5.7

### **5.9.3 Device Switching**

The device switching, also known as users' access time [46], is the measurement of time for a user to go from device A to device B. For example, this could mean that someone using a GamePad to navigate, may be required to put down this device before using the keyboard.

---

<sup>10</sup>See <http://en.wikipedia.org/wiki/Gimbal>.

In the case of the experiment, the objective was to see if there was any difference when subjects performed the switch between the two devices tested (GamePad and multi-touch) and the keyboard. This meant the time the user took between the target collision and the successful sentence completion using the keyboard. Similar approaches to the switch between two devices have been studied [194]. The users' access time is called homing, when working with the Keystroke-Level Model (KLM) [25]. The KLM is related to Goals, Operators, Methods, and Selection (GOMS) [37] because it tries to break down complex tasks. The KLM and GOMS are important models, but they are outside the scope of this dissertation, given that none of these methods were used or applied.

## 5.10 Questionnaires

Subjects were given entry and exit questionnaires. A subset of subjects were given additional questions for the entry and exit surveys. Given the length of the survey, the actual surveys are shown in Appendix B. Before listing the questions in each survey, the legend that accompanied the survey is shown in Table 5.5. Some questions will be simplified for sake of space in this chapter. All the relevant questions in the survey are discussed in Chapter 6.

The entry survey, shown in Table 5.6, provided a way to classify the subject game expertise level (casual or experienced). This measured a ranking for the 3D navigation using a GamePad controller. The criteria designed to quantify the user's expertise is discussed later. An additional questionnaire was given to a selected number of subjects. This helped validate the previous question's objective, which it was to find the expertise of the subjects in relation to game playing. The additional questions are shown in Table 5.7.

The exit survey provided a subjective evaluation of the system. Besides understanding what each subject internalized during the process, the survey looked to validate the objec-



Symbol	Choices
¶	6 months; 1 year; 2-4 years; 4-6 years; 5-10 years; 10 or more years
§	Never; Rarely; Daily; Weekly; Once a month; Once every 3 months; Once in 6 months; Once a Year.
†	5:Extremely well skilled; 4: Very good; 3: Good; 2: Not very skilled; 1: Not skilled at all.
‡	Choose either for GamePad or multi-touch, each of the following operations: Rotation: Yaw, Roll, Pitch Translations: Up/Down, Left/Right, Forward/Back See Appendix B for more information.

Table 5.5: Multiple Choice Legend.

tive data, or explain the discrepancies observed in them. The questions are shown in Table 5.12. Additional questions were given to a selected number of subjects to understand the interaction with the Hold-and-Roll bi-manual gesture. This is shown in Table 5.8.

## 5.11 Gamers' Experience

Ahead of time, before the experiment started, there was a valid concern that some users may have extensive experience with the GamePad when navigating 3D games. The entry questionnaire shown in Table 5.6 provided a way to classify users in this regard. Similar methods to gamers' classification have been used before in [120]. Equation 5.1 shows the game experience calculation, which is the weighted sum of some questions divided by the

#	Question	Type
1	Have you ever played PC Games? If Yes, list a few of them and when you played them.	Open
2	Have you ever played Console Games (XBOX, PlayStation, Nintendo)? If yes, list a few of them and when you played them	Open
3	Have you ever played smart phone or tablet games? If yes, list a few of them, and list the device and when you played them	Open
4	If you play video games rarely or don't play video games, can you tell us why? Is it cost, low interest, lack of time, lack of skills or another set of reasons?	Open
5	How long have you been playing video games? Please circle one option.	Range <sup>¶</sup>
6	How often (approximately) do you currently play video games? Please circle one.	Range <sup>§</sup>
7	How would you describe your skill level at playing video games in a scale of 1-5, with 5 the being the most skilled and 1 the least skilled?	Range <sup>†</sup>
8	What gaming systems do you own or have you owned in the past? Please list them and specify if you still own them. Also, include if there are any systems you would like to own in the next year.	Open
9	Please list your favorite video games. List at least a couple, if possible, and tell us why.	Open
10	Please tell us what other devices besides multi-touch or GamePad have you used to play video games. Have you used the Nintendo WiiMote or PlayStation Move? You can describe any device that you have used to play games in this question.	Open
11	Have you heard about the Oculus Rift (experimenter will show you one) or similar devices? Can you tell us what you think about those devices and playing video games with them, if you have an opinion? Have you ever use them?	Open
12	Please feel free to write any other opinions you want to express about video games below.	Open

Table 5.6: Entry Questionnaire.

#	Question	Type
13	How often do you use GamePad to play video games (currently)?	Range <sup>§</sup>
14	If you don't use it as often now, describe how often you used to use it before.	Range <sup>§</sup>
14b	Describe when (about 14).	Open
15	Rate how easy you find the GamePad (very easy = 10, very hard = 1).	Scale 1-10
16	Do you have a preference for any type of GamePad (e.g., XBOX ONE, Logitech, Playstation)? List them in order of preference, if you have more than one	Open
17	Please describe what you think of GamePads.	Open

Table 5.7: Additional Entry Questionnaire.

#	Question	Type
23	Please rate the Hold-and-Roll gesture you used during the experiment (10 = very useful , 1= not useful at all)	Scale 1-10
24	Would you like to see this gesture in new games? Please explain.	Open
25	Would you like to see this gesture in new applications? Please explain.	Open
26	What is your opinion about Hold-and-Roll?	Open
27	Please describe what benefits the multi-touch interaction gave you during this experiment. How about for your daily use?	Open
28	Please describe what benefits did the GamePad gave you during the interaction. How about for your daily use?	Open

Table 5.8: Additional Exit Questionnaire.

maximum score (13.25), providing a normalized value from 0 to 1 (0% to 100%). This formula contains  $Q_n$ , where  $Q$  stands for question and the index  $n$  stands for the question number, a weight for each question (reflecting its importance), and the constant of 13.25 to normalized the results into  $L$ , which is the game-level experience. The actual classifications for each question are shown in Table 5.9. Table 5.10 describes the game levels in detail.

Studies have looked at finding measurements for game levels (see [97, 206]). In this particular approach, Equation 5.1 was validated by selecting a subset of subjects and performing an additional interview. The second validation was provided by the additional questions in Table 5.8. By no means is this a general model, and further study is needed to determine a general approach to game-level classification. Nevertheless, for this experiment, this approach gave correct results and provided a way to define a game-level factor for the experiment.

$$L = \frac{(3 * Q_1 + 6 * Q_2 + 0.25 * Q_5 + 1.5 * Q_6 + 1.5 * Q_7 + 0.5 * Q_8 + 0.5 * Q_9)}{13.25} \quad (5.1)$$

## 5.12 Objective Measurements

A set of measurements were recorded, which included: travel time, switching time, and sentence error rate. The travel time was defined as the time from the start of the treatment to the the final sentence typed. The switching time was defined as the time from the question prompt to the successful sentence completion using the keyboard. A third time can be derived, that is the treatment time minus the keyboard time. The sentence error rate is the number of incorrect sentences typed after pressing the enter key.

#	Level 6	Level 5	Level 4	Level 3	Level 2	Level 1	Level 0	None
$Q_1$	1.00	0.75	0.50	0.40	0.35	0.30	0.20	0.00
$Q_2$	1.00	0.75	0.50	0.40	0.35	0.30	0.20	0.00
$Q_9$	1.00	0.75	0.50	0.40	0.35	0.30	0.20	0.00
	6 mo.	1 yr.	2-4 yrs.	4-6 yrs.	6-10 yrs.	10+ yrs.	-	-
$Q_5$	0.05	0.10	0.30	0.50	0.80	1.00	-	-
	Never	Rarely	Daily	Monthly	Every 3 mos.	Every 6 mos.	Yearly	
$Q_6$	0.05	0.09	1.00	0.70	0.40	0.20	0.12	0.08
	Extremely well	Very good	Good	Not very Skilled	Not-skilled	-	-	-
$Q_7$	1.00	0.80	0.60	0.40	0.20	-	-	-
	5+	3-5	2	1	none	-	-	-
$Q_8$	1.00	0.70	0.50	0.25	0.00	-	-	-

Table 5.9: Game Classification.

Level	Classification	Description
6	Very Experienced Gamer	Also refereed as to Hard-Core gamer, This is a gamer that plays games very frequently and those games consist of 3D games
5	Experienced Gamer	Similar to Level 6, but plays less frequent
4	Semi-Experience Gamer	Similar to Level 6, but their game play is more recent
3	Classic	This is a person that plays many 2D games. It is called a classic gamer, because involves games that are designed using the 1980s type of games. This is different from casual games, since it involves a joystick or similar devices.
2	Strategy	This is a gamer that plays mostly strategy games. Most of this games required a different set of skills
1	Classic Casual	This is similar to Level 3, but plays less frequent and it plays more casual games
0	Casual	This is a gamer that plays casual games. This users will play touch games like angry-birds or card games
-	None	In some instances, the user may have no experience or almost no experience.

Table 5.10: Game Classification Description

### **5.13 Experiment Procedure**

The procedure for the subject included the entry and exit questionnaires, a video tutorial, two training sessions, and two treatment sessions. The following list describes a brief step-by-step procedure for each of the subjects:

1. Read and sign FIU IRB user's consent for this experiment.
2. Request a user to draw a number, which will provide his or her identification number. This will provide the permutations as described in 5.5.2.
3. Briefly explained to the user the logistics of the experiment (2 minutes).
4. Ask the subject to respond to entry survey, shown in Table 5.6.
  - (a) additional questions were provided if subject was selected (see Table 5.7).
5. Have the subject view the tutorial video.
6. Train subject with both devices, in the order provided by the identification number (drawn in 2).
7. The subject performs the navigation with both devices, in the order provided by the identification number (drawn in 2).
8. Ask the subject to respond the exit survey, shown in Table 5.12.
  - (a) additional questions were provided if subject was selected (see Table 5.8).
9. Conclude Experiment.

### **5.14 3D Navigation Experiment Tour**

The experiment provided the entry and exit survey, as well as a training video. There were five objects to find. In the case of the training, the hypercube (Figure 5.5) was used to

search. The five objects for the search task (in both treatments) were a hypersphere (Figure 5.8), a spaceship (Figure 5.10), a green creature (Figure 5.11a), a space satellite (Figure 5.11b), and a tetrahedron (Figure 5.11c). A series of static non-target objects were also placed in the virtual world, which included a red creature (Figure 5.12a), and a green space ship (Figure 5.12b), among others. A screenshot of the actual experiment display (on the multi-touch screen) is shown in Figure 5.9.

For the actual treatment, the subjects were asked to use a multi-touch or GamePad device (in random order) to be used for the treatment session. The user advised when he/she was ready to start the experiment. Once ready, the experimenter pressed the special function key N<sup>11</sup> (see Figure 5.4b). When a target was found, the user was required to collide with the target. Once the collision was detected, an input text box message appeared in the middle of the screen, as shown in Figure 5.13, asking the user to type a sentence. The user was only allowed to use the keyboard at this point. The sentence had to be typed correctly to move to the next phase. This would iterate for each object. The next device treatment was the same, but the objects were swapped between them. The actual sentences for the experiment are found in Table 5.11. Note that one word used British spelling purposely (“travelled”).

A set of figures are provided to visualize the experiment trials. A subject is shown in Figures 5.14 and 5.15. The subject is performing different actions. In Figures 5.14a and 5.14b, a subject is performing a one-hand two-finger rotation to perform a yaw. The subject is also typing once an object has collided, shown in figure 5.14c. In Figures 5.15a, 5.15b, and 5.15c, the subject is performing the Hold-and-Roll (bi-manual) gesture to acquire a target (by moving forward/back).

---

<sup>11</sup>This correspond to the keypad 9.

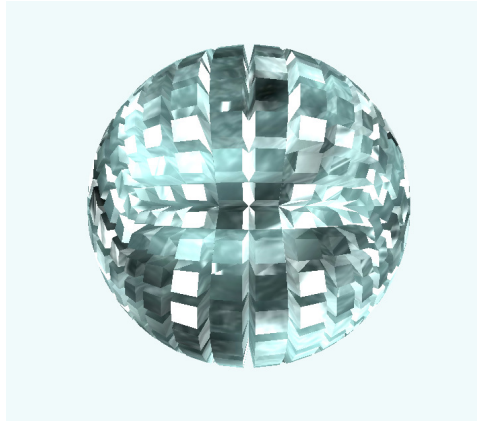


Figure 5.8: Hyper sphere (Target Object)

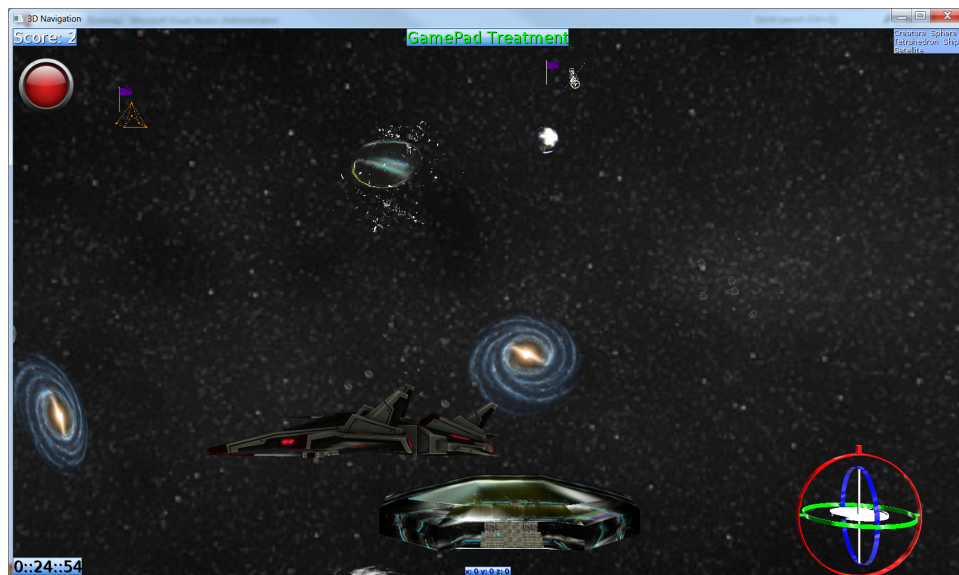


Figure 5.9: 3D Navigation Experiment Display

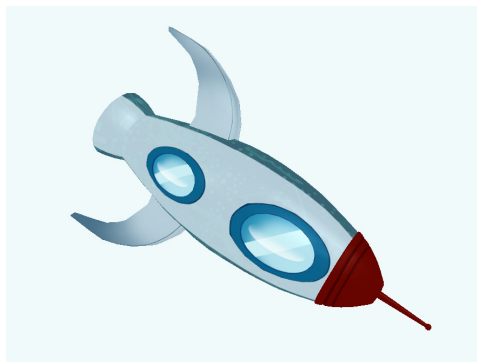


Figure 5.10: Space Ship (Target Object)



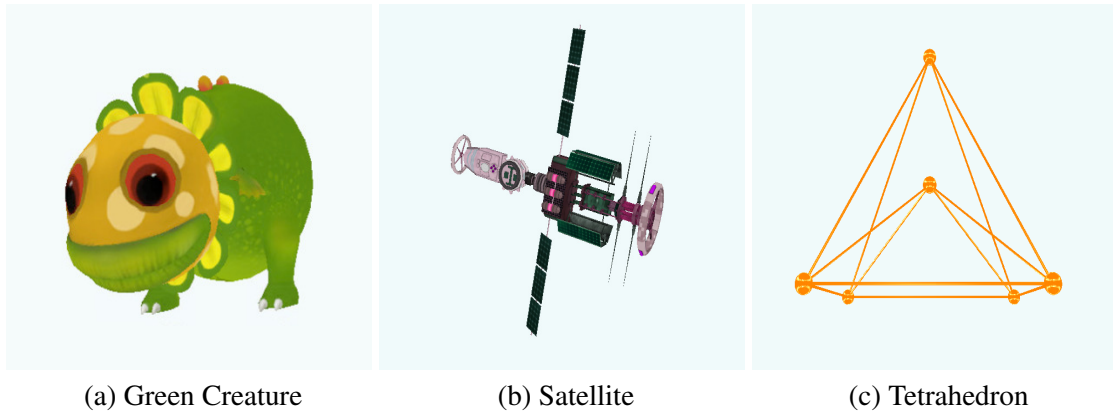


Figure 5.11: Target Objects

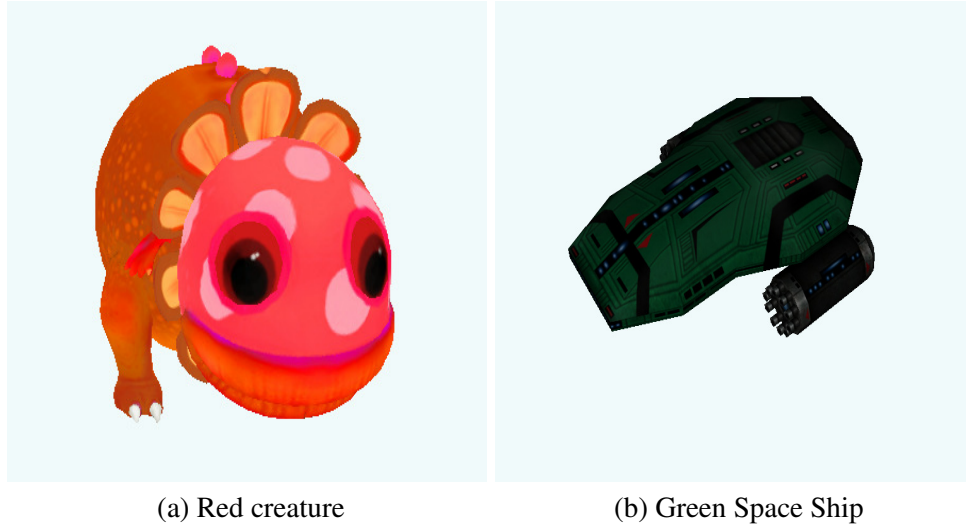


Figure 5.12: Non-Target Objects

Sentence	Mode
Soccer is the greatest sport of the world.	Training
I'm having to type short sentences.	Treatment
The greatest coach of all time has travelled <sup>†</sup> to France.	Treatment
I dream with a big library full of books.	Treatment
I have been told not to write with CAPS (Uppercase).	Treatment
I'm having to type short sentences.	Treatment

Table 5.11: Sentences.

<sup>†</sup> British spelling.

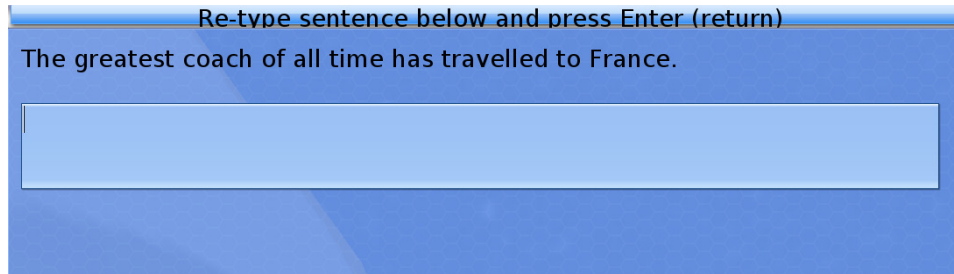


Figure 5.13: Keyboard Pop Up



Figure 5.14: Subject

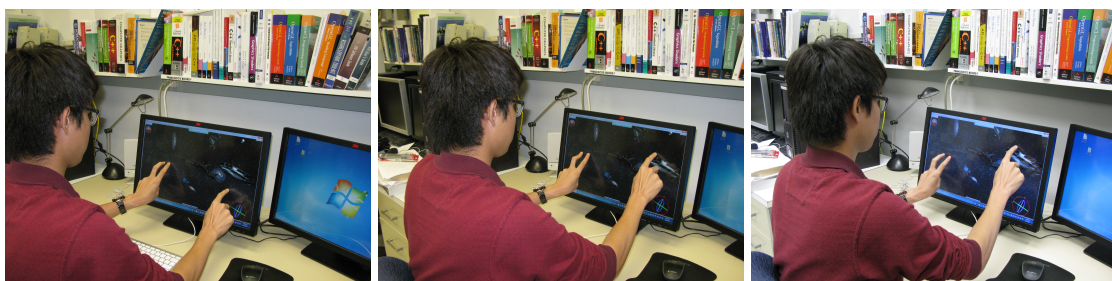


Figure 5.15: Hold-and-Roll

Table 5.12: Exit Questionnaire.

#	Question	Type
1	On a scale of 1 to 10, please rank how much easier you found the multi-touch display compared to the GamePad for 3D navigation. The higher you rank (10), the easier you found the multi-touch display versus the GamePad.	Scale 1-10
2	On a scale of 1 to 10, please rank how much easier you found the GamePad device compared to the multi-touch display. The higher you rank (10), the easier you found the GamePad device versus the multi-touch display.	Scale 1-10
3	On a scale of 1 to 10, please rank how intuitive you found the multi-touch display. The higher you rank (10), the more intuitive you found it	Scale 1-10
4	For the task given during the experiment: how do you rank the interaction to perform the search with the multi-touch display? The higher you rank (10), the better you found the experience with the device to perform the assigned task during the experiment.	Scale 1-10
5	For the task given during the experiment: how do you rank the interaction to perform the search with the GamePad? The higher you rank (10), the better you found the experience with the device to perform the assigned task during the experiment.	Scale 1-10
6	Given the time you took with multi-touch display, rank how likely you are to use this device for daily use if you had access to it. The higher you rank, the more you expect to use if it was available to you.	Scale 1-10
7	Given the time you took with the GamePad device, rank how likely you are to use this device for daily use if you had access to it. The higher you rank, the more you expect to use if it was available to you.	Scale 1-10

*Continued on next page*

Table 5.12 – *Continued from previous page*

#	Question	Type
8	Please rank how the multi-touch display compared to the GamePad device for rotating the camera. The higher you rank (10), the better you found the multi-touch display for rotations	Scale 1-10
9	Please rank how the GamePad device compared to the multi-touch display for rotating the camera. The higher you rank (10), the better you found the GamePad device for rotations.	Scale 1-10
10	Please rank how the multi-touch display compared to the GamePad device for translation (up, down, left, right, forward, back). The higher you rank (10), the better you found the multi-touch display for translation movements.	Scale 1-10
11	Please rank how the GamePad compared to the multi-touch display for translation (up, down, left, right, forward, back). The higher you rank (10), the better you found GamePad for translation movements.	Scale 1-10
12	Which device do you prefer: GamePad, multi-touch, No Difference (both). (please circle one)	multiple-choice
13	Which device did you find better to switch to the keyboard (when asked to type)? GamePad, multi-touch, No Difference (both). (Please circle one).	multiple-choice
14	Please select which Rotation or Translation you found better for the experiment you tested. Please mark with X for each of the categories.	multiple-choice <sup>‡</sup>
15	Please tell us what you thought about the experiment?	Open
16	Tells us why you prefer one device over the other for the experiment.	Open
17	Tell us why you prefer one device over the other when you have to type, as the experimented demanded.	Open
18	Please tell us your overall opinion about the design of the multi-touch display.	Open

*Continued on next page*

Table 5.12 – *Continued from previous page*

#	Question	Type
19	Please tell us your overall opinion about the design of the GamePad device.	Open
20	Please tell us how we did in the experiment. Is there anything in the experiment that can be done better next time?	Open
21	What do you think about the three-ring sphere that gave you a sense of rotation in space?	Open
22	You can use the rest of this page to write any comments before starting the experiment about the questions asked. Please feel free to write anything about video games below.	Open

## CHAPTER 6

### EXPERIMENT ANALYSIS

This chapter presents the statistical data analysis for the 3DNav experiment described in Chapter 5. The objective of this chapter is to provide the quantitative and qualitative analysis of the data. Several statistical tests were performed, including t-tests and Analysis of variance (ANOVA) tests. The software used to perform the data analysis included IBM SPSS version 19 and Microsoft Excel. The results will be interpreted in Chapter 7.

#### **6.1 Data Outliers**

When looking at the primary measurement factor (time), two subjects were considered outliers. One of the subjects (Row #7, ID #6, female, age 29) showed a huge difference in time when using the GamePad compared to the rest of the subjects, as shown in Figure 6.1a. This subject was interviewed at a later time to see if the discrepancy could be understood. The user said that she gave up at times when using the GamePad out of frustration. The subject also reported that she had trouble using the controller for all types of navigation purposes. The other subject (Row #9, ID #8, male, age 33) showed a difference from the rest of the subjects when using the multi-touch display, as shown in Figure 6.1b. The difference was apparent to the experimenter during the trial. The user had physical problems that made it difficult for him to use the multi-touch display. Therefore, both subjects were removed from the analysis, bringing the total pool of subjects to 28. It is important to note that the removal of the outliers mentioned did not make a difference in the significance of the comparison of means. Furthermore, a possible third outlier, described later, was not removed from the dataset.

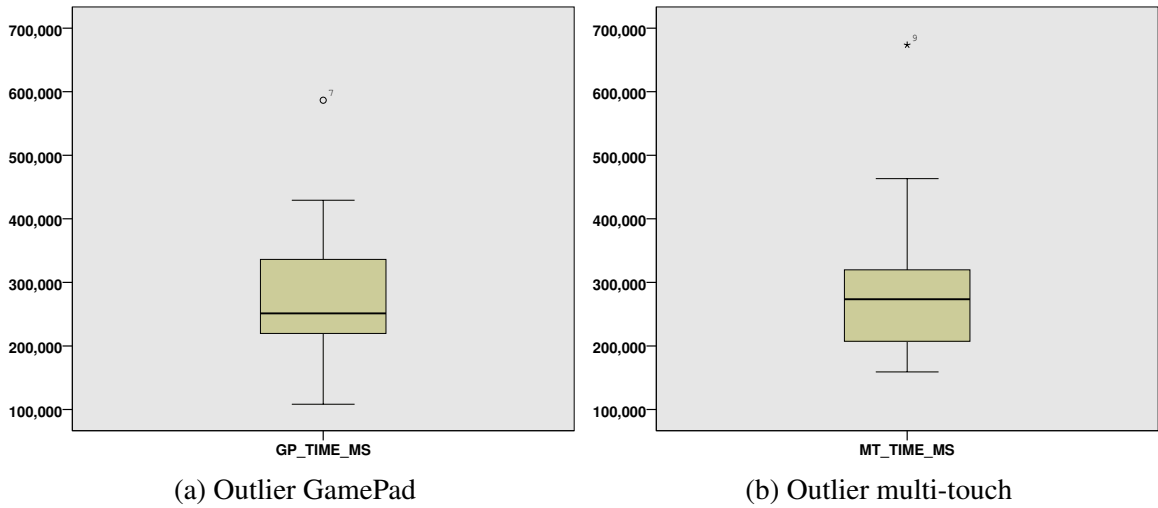


Figure 6.1: Outliers

## 6.2 The Dataset

The data collected consisted of quantitative (objective) data and qualitative (subjective) data) for 30 participants. The final pool was reduced to 28 subjects for reasons explained already in 6.1. The actual data items collected are described extensively in Chapter 5.

The subject pool analysis consisted of 28 subjects, of which 18 were male and 12 were female. The average age was 27 years old (27.2857143), with the oldest subject having 42 years of age and the youngest subject having 19 years of age (the mode was 23). The population was primarily from the School of Computing and Information Sciences and the Engineering College. This consisted of 15 undergraduate students (mostly junior and seniors), 8 graduate students (master's and PhD), and 5 professional subjects who had at least a bachelor's degree.

## 6.3 Quantitative Data

The quantitative data provided an objective understanding of the interaction of the user with each device. The most crucial measurements were the time taken in total for each device

(referred to as  $T_d$ ), the time taken to type sentences with the keyboard for each treatment (refer as  $T_k$ ), and the difference between  $T_d$  and  $T_k$  ( $T_{dnk} = T_d - T_k$ ). Additionally, the other measurement taken is the number of tries per device when typing on the keyboard, with the lowest count expected to be five (for a total of 5 sentences per device). When analyzing some of the objective data, a co-factor was taken into consideration. This is the Experience Level<sup>1</sup>, which is divided into two categories: Casual gamers, which is category 1 (60% or lower), and experienced gamers, which is category 2 (above 60%). This is discussed in 5.11. It is important to note that this game's expertise level refers to the person's skills in video game consoles or games that required the use of GamePad. The reason for this co-factor is that it was expected for regular GamePad users to be able to manipulate this controller better than regular users. Finally, the time variables were recorded in milliseconds. Therefore, unless otherwise stated, the default unit of measurement of time in this experiment is milliseconds, or the  $\log(x)$  transformation of such time.

### 6.3.1 Time: GamePad and Multi-Touch

The time considered for each treatment was from the start of treatment to the completion of the final objective of the trial. This measurement ( $T_d$ ) provides an unbiased look at the completion time for each device. Table 6.1 includes: mean ( $\bar{x}$ ), standard error of mean (SEM), median ( $\tilde{x}$ ), mode (MO), standard deviation (SD), variance (Var), minimum (Min), maximum (Max), and population size (N), among others. The variables MT\_TIME\_MS and GP\_TIME\_MS describe the time elapsed for the multi-touch treatment (in milliseconds) and GamePad, respectively. The mean for multi-touch is  $\bar{x} = 262.28$  seconds and for the GamePad is  $\bar{x} = 270.29$  seconds. This means that the multi-touch mean has a slight advantage of 8 seconds.

---

<sup>1</sup>Also referred to as Hard-Core Gamers.



	MT_TIME_MS	GP_TIME_MS
<b>Mean</b>	262288.29	270298.96
Std. Error of Mean	14289.806	14516.009
Median	271057.00	241555.50
Mode	159130 <sup>¶</sup>	108266 <sup>¶</sup>
<b>Std. Deviation</b>	75614.544	76811.499
Variance	5717559249.619	5900006358.628
Skewness	.613	.109
Std. Error of Skewness	.441	.441
Kurtosis	.232	-.562
Std. Error of Kurtosis	.858	.858
Range	304142	320982
Minimum	159130	108266
Maximum	463272	429248
Percentiles		
25	200434.25	216771.00
50	271057.00	241555.50
75	312495.75	335238.50

Table 6.1: Descriptive Statistics for  $T_d$

Table Legend: ¶ Multiple modes exist. The smallest value is shown

Table 6.2 shows the normality tests (i.e., the Kolmogorov-Smirnov and Shapiro-Wilk tests). The latter shows that the data is normal for both treatment variables. The former test (Kolmogorov-Smirnov) shows that is only normal for multi-touch. While this may be concerning, when looking at the QQ plots, the normality can be visualized in Figures 6.4b and 6.4a. Nevertheless, the data can be converted using a log transformation function ( $\log(x)$ ) [48], which passed all the normality tests, as shown in Table 6.3. This transformed data is called  $T_{dx}$ . The QQ plots for the transformations are shown in Figures 6.5a and 6.5b. The histograms with the Gaussian curve are shown for non-transformed and transformed cases, in Figures 6.7 and 6.6. Finally, the descriptive information for the transformed data is shown in Table 6.4

	Kolmogorov-Smirnov †			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
MT_TIME_MS	.108	28	.200 <sup>¶</sup>	.948	28	.175
GP_TIME_MS	.180	28	.020	.961	28	.372

Table 6.2: Normality Test for  $T_d$

Table Legend: † Lilliefors Significance Correction.  
 ¶ This is a lower bound of the true significance.

	Kolmogorov-Smirnov †			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
MT_TIME_LOG	.129	28	.200 <sup>¶</sup>	.963	28	.411
GP_TIME_LOG	.141	28	.163	.940	28	.114

Table 6.3: Normality Test for  $T_{dx}$

Table Legend: † Lilliefors Significance Correction.  
 ¶ This is a lower bound of the true significance.

After the Log transformation, Figure 6.3a displayed an outlier on the graph (Row #16, ID #16, male, age 24). However, after careful analysis, this subject was kept in the pool of data used for analysis for the following two reasons: First, the subject was not an outlier in the data before the transformation. Second, his low time for both devices was just part of his vast game experience, as this subject is in the hard-core category, with a hard-core level of 94%. The actual times for this subject were 159 seconds for the multi-touch device, and 108 seconds for the GamePad. Notice that the other outliers had a large difference between the mean and each of their times. For example, subject ID #6 had a total of 586.59 seconds for the GamePad device, yielding over 300 second difference. Subject ID #8 had a total of 673.52 seconds, yielding over 400 second difference. This was not the case for ID #16, whose difference to the mean was less than 90 seconds for each of the devices. Therefore, the conclusion was to leave the subject in the dataset.

	MT_TIME_Log	GP_TIME_Log
<b>Mean</b>	5.4016	5.4134
Std. Error of Mean	.02356	.02519
Median	5.4331	5.3830
Mode	5.20 <sup>¶</sup>	5.03 <sup>¶</sup>
Std. Deviation	.12466	.13330
Variance	.016	.018
Skewness	.011	-.704
Std. Error of Skewness	.441	.441
Kurtosis	-.719	.947
Std. Error of Kurtosis	.858	.858
Range	.46	.60
Minimum	5.20	5.03
Maximum	5.67	5.63
Percentiles		
25	5.3019	5.3360
50	5.4331	5.3830
75	5.4947	5.5253

Table 6.4: Descriptive Statistics for  $T_{dx} = \text{Log}(T_d)$

Table Legend: ¶ Multiple modes exist. The smallest value is shown

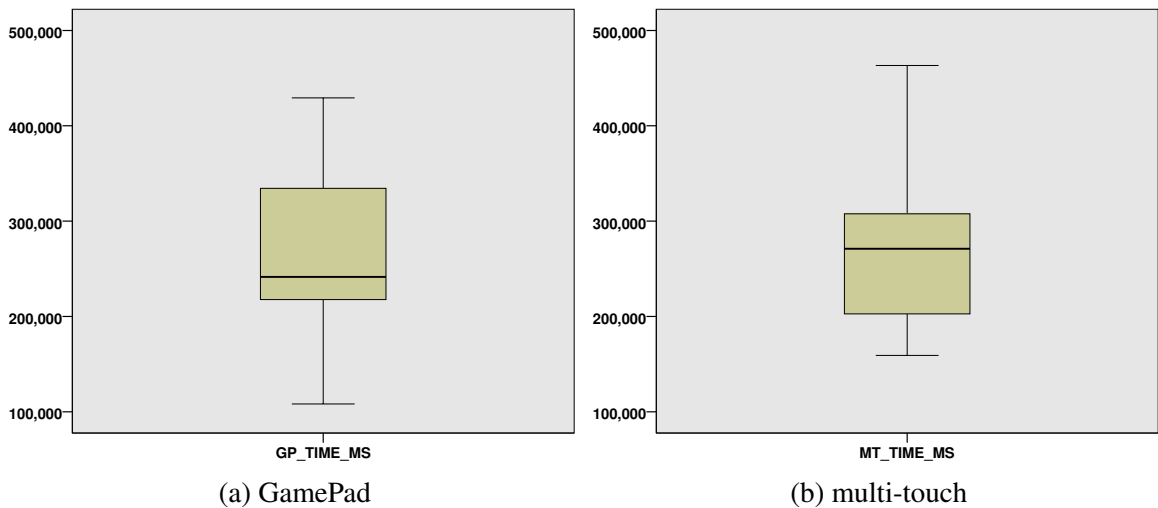


Figure 6.2: BB Plot

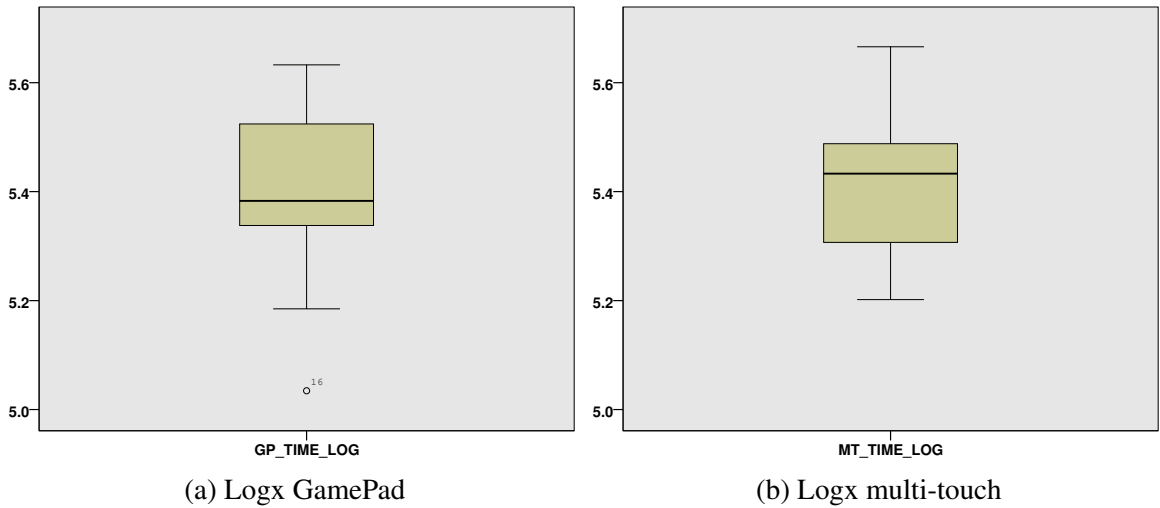


Figure 6.3: BB Plot (Transform Data)

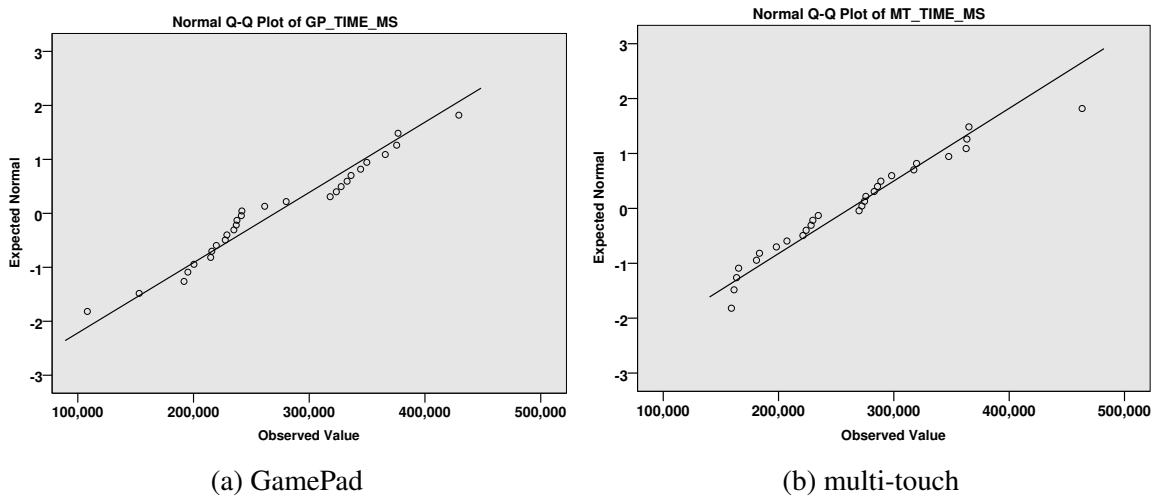
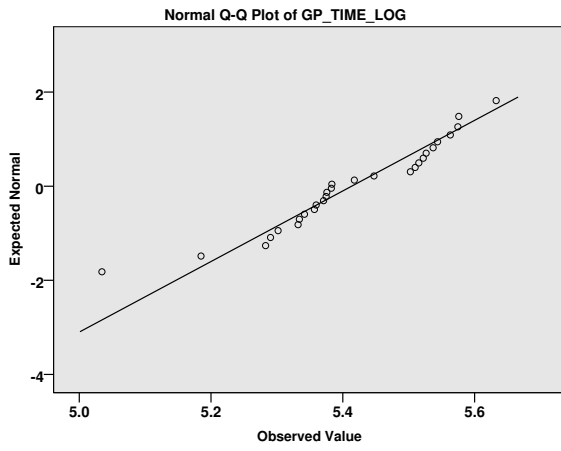
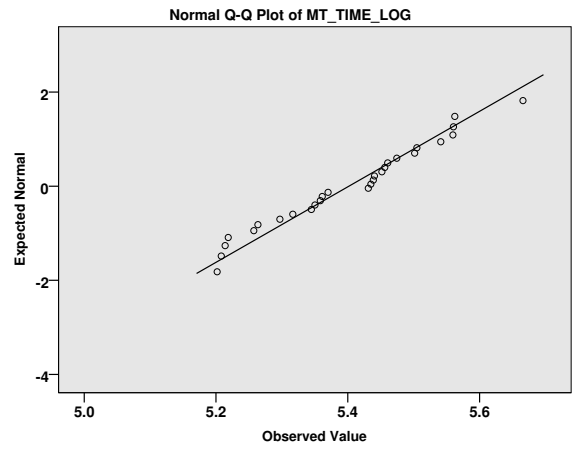


Figure 6.4: QQ Plot

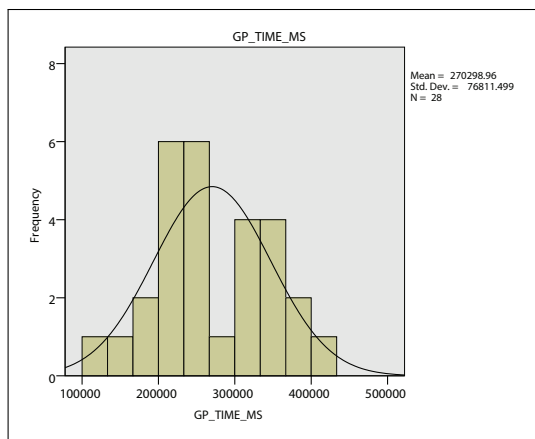


(a) Logx GamePad

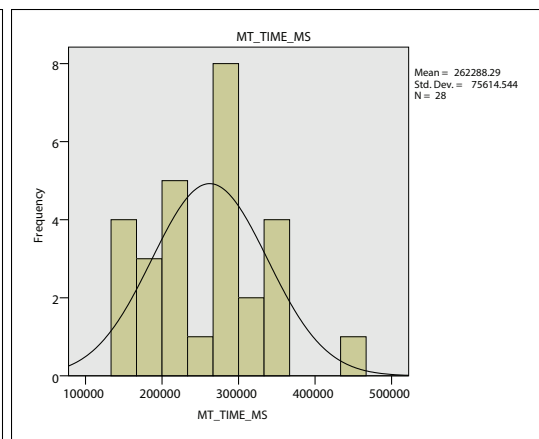


(b) Logx multi-touch

Figure 6.5: QQ Plot (Transform Data)



(a) GamePad



(b) multi-touch

Figure 6.6: Histograms

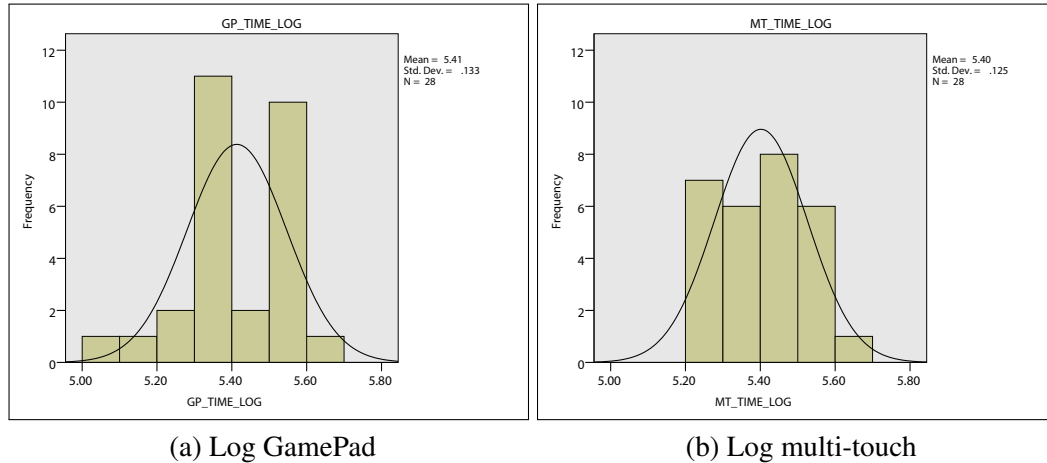


Figure 6.7: Histograms (Transformed Data)

### Comparison of Means

After analyzing the descriptive statistics and showing that the mean for the multi-touch device (4 minutes and 37 seconds) was smaller than the GamePad device (4 minutes and 50 seconds), the analysis needed to determine if the difference was significant. The t-test showed that the difference of the means was not statically significant. Therefore, the difference between the multi-touch device and the GamePad, yielded  $t(27) = -.490$ ,  $p > .05$ . The same was the case for the transformed data (using  $\log(x)$ ), that yielded  $t(27) = -.432$ ,  $p > .05$ .

### One-way ANOVA: Repeated Measures

The analysis of the data is performed with the transformed data. The premise is that the data complies with the assumptions required for ANOVA [48]. Nevertheless, the non-transformed data, while failing one of the normality tests, does appear to be normally distributed (see Figures 6.4a and 6.4a). When running ANOVA, it is expected that a similar result from the previous t-test will also yield a non-significant difference for  $T_{dx}$ . This was the case as well, with ANOVA results of  $F(1,26) = .1$ ,  $p > 0.5$ .

Factor	Input Time	Mean	Std. Error
Casual	Multi-touch	5.437	0.031
	GamePad	5.495	0.026
Experienced	Multi-touch	5.361	0.033
	GamePad	5.319	0.028

Table 6.5: Co-Factor Means  $T_{dx}$

An additional test was performed using the game-level factor (see 5.11). In the subject pool, 13 subjects were classified as experienced gamers and 15 subjects were classified as casual users. The study sought to see if there was any difference between the experience gamers and the casual players when using the GamePad and the multi-touch device. In this test, the hypothesis was that experienced gamers would pull the average completion time down, for the GamePad controller. This hypothesis showed to be significant. In other words, for experienced gamers only the difference of mean completion times between both devices was significant, with  $p < 0.1$ . This was not the case for casual gamers, showing that the difference of means was not significant,  $p > 0.5$ . Looking at the non-transformed data that provides the actual time, it is possible to see the differences between both groups. The mean for the experienced gamers was 214.8 seconds with the GamePad and 239.4 seconds with the multi-touch device. For the casual gamers, the mean was 318 seconds with the GamePad and 282 seconds with the multi-touch device. Table 6.5 provides the means of the transformed data and the analysis of the interaction of the factor with each device is reported in Table 6.6.

### **Time: Breaking the groups**

The casual gamer group is composed of 8 males and 7 females, with an average age of 29.80 (SD=7.153), a mode of 23, and minimum/maximum of 23 and 38, respectively. The

Input Time	(I) Factor	(J) Factor	Mean Difference (I – J)	Std. Error	Sig. <sup>†</sup>
Multi-touch	Casual	Experienced	0.76	0.46	0.108
	Experienced	Casual	-0.76	0.46	0.108
GamePad	Casual	Experienced	0.176 <sup>¶</sup>	0.38	0.00
	Experienced	Casual	-0.176 <sup>¶</sup>	0.38	0.00

Table 6.6: Co-Factor Analysis  $T_{dx}$

Table Legend: † Adjustment for multiple comparisons:  
Least Significant Difference (equivalent to no adjustments).  
¶ The mean difference is significant at the .05 level.

experienced gamer group was composed of 11 males and 2 females, with an average age of 24.38 (SD=3.61), a mode of 23, and a minimum/maximum of 19 and 35, respectively. The data is normal for both groups as shown in Table 6.7. For the homogeneity (Levene) test, the variance of the  $T_{dx}^2$  for both game factors (casual and experienced) was equal for the GamePad  $F(1,26) = 2.362$ ,  $p > 0.05$  and the multi-touch  $F(1,26) = 0.64$ . Therefore, the homogeneity test passed.

After running the t-tests for each group<sup>3</sup>, the expected outcome showed that it was significant for the casual gamer category. In other words, casual gamers completed their tasks in less time when using the multi-touch device ( $\bar{x} = 5.44$ ,  $SD = 0.11$ ) compared to the GamePad device ( $\bar{x} = 5.51$ ,  $SD = 0.87$ ), with  $t(15) = -1.942$ , one-tailed significance of  $p < 0.05$  (Two-tailed significance was equal to 0.073, divided by 2 gives one-tailed significance of 0.036),  $r = 0.46$ . In the experienced gamer category, there was no significance, with a mean of 5.36 for the multi-touch and 5.32 for the GamePad. To place the results of the first category (casual gamers) into context, the actual average time for the multi-touch device was 282 seconds, and for the GamePad device, 318.6 seconds. For the latter category (experienced gamer), the average time to complete the tasks for the multi-touch device was 239.4 seconds and for the GamePad device was 214.8 seconds. There is a clear tendency

<sup>2</sup>Using the transformed data with  $Log(x)$ .

<sup>3</sup>Using the split file function of SPSS.



Game Group		Kolmogorov-Smirnov †			Shapiro-Wilk		
		Statistic	df	Sig.	Statistic	df	Sig.
Casual	MT_TIME	.165	15	.200 <sup>¶</sup>	.958	15	.655
	GP_TIME	.201	15	.104	.937	15	.345
Experienced	MT_TIME	.171	13	.200 <sup>¶</sup>	.948	13	.090
	GP_TIME	.222	13	.081	.884	13	.081

Table 6.7: Normality Test for  $T_{dx}$  by Group

Table Legend: † Lilliefors Significance Correction.  
 ¶ This is a lower bound of the true significance.

for experienced gamers to perform faster (without statistical significance) when using the GamePad controller. This will be discussed in detail in Chapter 7.

### 6.3.2 Homing: Switching Devices

Participants were required to switch from either the GamePad or the multi-touch to use the keyboard. In the case of this study, the time is Homing plus the time to complete a sentence successfully. This can be formulated as the sum of homing ( $H$ ) plus the time that it takes to successfully complete the sentence requested by the experiment using the keyboard ( $K$ ), as shown in Equation 6.1. The data analysis used untransformed data (in milliseconds) because it passed all the normality tests and the homogeneity of variance test when looking at the game category groups (casual and experienced).

In addition to the time  $S_t$ , the error rate for the keyboard ( $E_k$ ) is also taken into account. For each device, the user was required to type five sentences correctly per device. Any additional tries after that are considered user errors. For example, a user with 6 tries for the multi-touch would have a 0.20 (20%) error rate. This is calculated using Equation 6.2.

$$S_t = H_t + K_t \tag{6.1}$$

$$E_k = (TRIES/5) - 1 \quad (6.2)$$

### **Sentence Completion Time**

The average time for the multi-touch display to keyboard, the completion time, was 81.6 seconds, and for the GamePad to keyboard, the completion time was 85.8 seconds. The data passed the normality test. When looking at the entire population, without regard for the game-category factor, there was no significant difference between either device when switching to the keyboard and completing a correct sentence. The result yielded  $t(27) = -.490$ , with  $p > 0.05$ . When looking at the separate game experience groups, there was also no significant difference.

### **Error Rate**

The error rate is determined by how many times the user hit enter with an incorrect answer, as described in Equation 6.2. However, the error rate is not normally distributed. For this reason, a non-parametric test, the Wilcoxon-signed-rank test was used to see if the assumption that users would have a more fluid interaction between two devices in favor of the the multi-touch and keyboard switch. This yielded a one-tailed significant difference.

The Wilcoxon-signed-rank test allowed the analysis to find if there is any difference between the error rates for each device. As expected, there was a difference when users transitioned to the keyboard from each device. In particular, users had a higher error rate when using the GamePad and the keyboard (mean = 0.1857, SD = 0.335), compared to the multi-touch and the keyboard (mean = 0.0643, SD = 0.163). In other words,  $T=0$ ,  $p < 0.05$  (One tailed significance),  $r = -0.352$  for the GamePad. Users tended to make more errors when using the transition between the GamePad and keyboard, as shown in Tables 6.8 and 6.9.

Rank Type	N	Mean Rank	Sum of Ranks
Negative	3 <sup>†</sup>	5.17	15.50
Positive	9 <sup>‡</sup>	6.94	62.50
Ties	16 <sup>*</sup>		
Total	28		

Table 6.8: Wilcoxon-signed-rank test: GP\_Keyboard - MT\_Keyboard

Table Legend: † GP\_KEYERROR < MT\_KEYERROR  
‡ GP\_KEYERROR > MT\_KEYERROR  
\* GP\_KEYERROR = MT\_KEYERROR

Z	-1.861 <sup>†</sup>
Sig. (1-tailed)	0.0315
r	-0.352

Table 6.9: Wilcoxon-signed-rank statistics: GP\_Keyboard - MT\_Keyboard

Table Legend: † Based on negative ranks.

## 6.4 Qualitative Data

Section 6.3 provides the analysis of the objective data. This is an important aspect of user study. However, there are aspects that quantitative data cannot easily measure, which qualitative data can complement. This section describes the relevant questions from the exit survey described in Chapter 5 that ties into the objective of the experiment. Detailed discussion about this section will be covered in Chapter 7.

### 6.4.1 Paired Questions

Two questions were asked to the subjects, which are meant to be treated as pairs. These were questions  $Q_1$  and  $Q_2$  in 5.10 (see Table 5.12). These questions tried to get feedback

from the user in regards to the multi-touch versus the GamePad, and vice versa, using a scale of 1-10. The purpose was to try to remove some bias and see if the answers were coherent, within each pair. For example, if the user found the multi-touch very easy to use, it would be expected that the user would rank in the following question (that asked the reverse), the GamePad not easy at all. Therefore, for this set of questions, the data analysis would include looking at each of them independently (e.g., mean), the bivariate correlation between them, and the ranking between both.

The GamePad scored higher ( $Q_2$ ), with a mean of 7.86 (mode = 10.0, median = 8.0) versus the multi-touch ( $Q_1$ ), with a mean of 5.46 (mode = 4.0, median = 5.00). The normality tests were not passed for  $Q_2$ , with both the Kolmogorov-Smirnov and Shapiro-Wilk tests showing a significant value of  $p < 0.05$ . Therefore, any correlation tests would need to use non-parametric methods. The Spearman's rho ( $r_s$ ) and Kendall's tau ( $\tau$ ) tests were performed. Both of them show a (2-tailed) significant difference, with  $r_s = -0.485$ ,  $p < 0.01$  and  $\tau = -0.376$ ,  $p < 0.05$ . This relationship is negative because as  $Q_1$  increases,  $Q_2$  decreases, and vice-versa. When looking at the Wilcoxon-signed-rank non-parametric test, it also shows a significance, where  $Q_2 - Q_1$  is (2-tailed) significant, with a negative ranking, with  $p < 0.01$ ,  $Z = -2.739$ , and  $r = -0.376$ , as shown in Tables 6.10 and 6.11. This means that users found the GamePad controller to be easier over the multi-touch device for 3D navigation.

When analyzing  $Q_2$  and  $Q_1$  by game groups (casual or experienced), there were some different results. When looking at the correlation, both tests (Spearman's rho and Kendall's tau ) were used. The test showed a significant correlation for  $Q_1$  and  $Q_2$  in both tests (while Kendall's tau has more importance since it works better with smaller samples) for the experienced gamers. The tests yielded  $r_s = -0.592$ ,  $p < 0.05$  and  $\tau = -0.491$ ,  $p < 0.05$ . This showed a negative relationship, which meant that as  $Q_2$  increases,  $Q_1$  decreases.

Rank Type	N	Mean Rank	Sum of Ranks
Negative	6 <sup>†</sup>	9.08	54.50
Positive	18 <sup>‡</sup>	13.64	245.50
Ties	4 <sup>*</sup>		
Total	28		

Table 6.10: Wilcoxon-signed-rank test:  $Q_2 - Q_1$

Table Legend: †  $Q_2 < Q_1$   
‡  $Q_2 > Q_1$   
\*  $Q_2 = Q_1$

Z	-2.739 <sup>†</sup>
Sig. (2-tailed)	0.006
r	-0.376

Table 6.11: Wilcoxon-signed-rank statistics:  $Q_2 - Q_1$

Table Legend: † Based on negative ranks.

For the Wilcoxon-signed-rank test, it only yielded (2-tailed) statistical significance for the experienced category. The mean for  $Q_1$  was 5.15 ( $SD = 2.478$ ) and the mean for  $Q_2$  was 8.62 ( $SD = 1.66$ ) when looking at the experienced gamer group, for a total  $N = 13$ . There were 2 negative rankings, with mean rank of 3.50, 10 positive ranks with mean ranking of 7.10, and one tie ranking. The Wilcoxon-signed-rank test yielded  $Z = -2.547$ ,  $p < 0.05$ ,  $r = -0.68$  (large effect size), with a negative ranking. This meant that the experienced gamer found the GamePad controller easier than the multi-touch display.

## 6.4.2 Additional Pairs of Questions

Additional paired questions were asked. However, these questions did not made a comparison between GamePad and multi-touch. The questions were only asked about one device,

but the type of questions were the same. The questions were  $Q_4$  to  $Q_{11}$  in pairs (e.g., 4-5, 5-6, and so forth), from the exit survey, shown in Chapter 5 (see Table 5.12). Most question did not pass the normality tests (Kolmogorov-Smirnov and Shapiro-Wilk), as shown in Table 6.12. Therefore, the analysis is better suited for non-parametric methods. The analysis is similar to the previous pair of questions (see 6.4.1). The descriptive statistics are shown in Table 6.13.

When looking at the bivariate correlation between each pair of questions, it must be interpreted a bit differently (and with caution) from the previous comparison in 6.4.1. This is because the questions did not put each device against each other, but they were considered independent. The reason that they are pairs is because the question is the same with just replacing the device name. Nevertheless, the correlation will still help in the discussion chapter.

Only the pairs  $\{Q_6, Q_7\}$  and  $\{Q_8, Q_9\}$  showed a statistical (2-tailed) significance. For  $\{Q_6, Q_7\}$ , Spearman's rho yielded  $r_s = -0.504$ ,  $p < 0.05$  and Kendall's tau yielded  $\tau = -0.386$ ,  $p < 0.01$ , with a negative relationship. This was also the case for  $\{Q_8, Q_9\}$  with  $r_s = -0.388$ ,  $p < 0.05$  and  $\tau = -0.330$ ,  $p < 0.05$ , with a negative relationship. This meant that as  $Q_6$  increased,  $Q_7$  decreased, and when  $Q_8$  increased,  $Q_9$  decreased.

The Wilcoxon-signed-rank test did not show any significant difference between the pair of questions. However, there are statistically significant results when looking at each gamer group category (casual, experienced). When looking at the casual gamers, the ranking for  $Q_7 - Q_6$  yielded a (2-tailed) significant difference based on a positive ranking. This meant that casual users said that based on their experience of the experiment, they were more likely to use the multi-touch display versus the GamePad controller. The mean for  $Q_6$  was 8.87 ( $SD = 1.356$ ) and the mean for  $Q_7$  was 5.00 ( $SD = 2.390$ ), both with  $N = 15$ . The Wilcoxon-signed-rank results where  $Z = -2.940$ ,  $p < 0.01$ , and  $r = -0.56$ . For the experienced group, there was a statistical (2-tailed) significance for  $Q_7 - Q_6$  and  $Q_9 - Q_8$ ,

	Kolmogorov-Smirnov †			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
$Q_4$	0.220	27	0.002	0.882	27	0.005
$Q_5$	0.236	27	0.000	0.883	27	0.006
$Q_6$	0.144	27	0.161	0.912	27	0.026
$Q_7$	0.130	27	0.200*	0.930	27	0.070
$Q_8$	0.144	27	0.155	0.952	27	0.245
$Q_9$	0.230	27	0.001	0.821	27	0.000
$Q_{10}$	0.132	27	0.200*	0.933	27	0.082
$Q_{11}$	0.211	27	0.003	0.852	27	0.001

Table 6.12: Normality Test for  $Q_4$  to  $Q_{11}$

Table Legend: † Lilliefors Significance Correction.  
 \* This is a lower bound of the true significance.

both favoring the GamePad, both with  $p < 0.05$ . The results for the experienced group are shown in Tables 6.14 and 6.15.

### 6.4.3 GamePad or Multi-Touch

Two questions were very specifically asking the user if they preferred the GamePad controller or the multi-touch display. Questions  $Q_{12}$  and  $Q_{13}$  are from the exit survey, shown in Chapter 5 (see Table 5.12). Question  $Q_{12}$  asked the subject to select which device was preferred. Question  $Q_{13}$  asked the subject to select which device was preferred when switching to the keyboard. For both questions, the possible answers were multi-touch (1), GamePad (2), or both<sup>4</sup> devices (3).

---

<sup>4</sup>Which means that subject found them equal.

	$Q_4$	$Q_5$	$Q_6$	$Q_7$	$Q_8$	$Q_9$	$Q_{10}$	$Q_{11}$
<b>Mean</b>	7.07	7.82	7.64	6.48	6.14	7.36	6.75	7.36
<b>SEM<sup>†</sup></b>	0.405	0.337	0.361	0.516	0.495	0.533	0.487	.520
<b>Median</b>	8.00	8.00	8.00	7.00	6.00	8.50	7.00	8.50
<b>Mode</b>	8	9	10	10	8	9 <sup>§</sup>	7 <sup>§</sup>	9 <sup>§</sup>
<b>SD<sup>‡</sup></b>	2.142	1.786	1.909	2.728	2.621	2.818	2.577	2.752
<b>Variance</b>	4.587	3.189	3.646	7.443	6.868	7.942	6.639	7.571
<b>Range</b>	7	7	6	9	9	9	9	9
<b>Minimum</b>	3	3	4	1	1	1	1	1
<b>Maximum</b>	10	10	10	10	10	10	10	10
<b>Percentiles</b>								
25	6.00	7.00	6.00	4.00	4.00	5.25	5.00	5.25
50	8.00	8.00	8.00	7.00	6.00	8.50	7.00	8.50
75	9.00	9.00	9.75	9.00	8.00	9.75	9.00	9.75

Table 6.13: Descriptive Statistics for  $Q_4$  to  $Q_{11}$

Table Legend: † Standard Error of Mean.  
‡ Standard Deviation.  
§ Multiple modes exists.  
The smallest shown.

	Rank Type	N	Mean Rank	Sum of Ranks
$Q_7 - Q_6$	Negative	2 <sup>†</sup>	7.50	15.00
	Positive	10 <sup>‡</sup>	5.67	51.00
	Ties	1 <sup>*</sup>		
	Total	13		
$Q_9 - Q_8$	Negative	3 <sup>¶</sup>	4.33	9.00
	Positive	10 <sup>§</sup>	7.80	78.00
	Ties	0 <sup>⊘</sup>		
	Total	13		

Table 6.14: Wilcoxon-signed-rank tests: Experienced gamer

Table Legend: †  $Q_7 < Q_6$  || ¶  $Q_9 < Q_8$   
‡  $Q_6 > Q_6$  || §  $Q_9 < Q_8$   
\*  $Q_7 = Q_6$  || ⊘  $Q_9 = Q_8$



	$Q_7 - Q_6$	$Q_9 - Q_8$
Z	-2.373 <sup>†</sup>	-2.280 <sup>†</sup>
Sig. (2-tailed)	0.018	0.023
r	-0.448	-0.431

Table 6.15: Wilcoxon-signed-rank statistics: Experienced gamer

Table Legend: † Based on negative ranks.

When looking at the entire sample ( $N = 28$ ), the preferred device ( $Q_{12}$ ) was the GamePad (mean = 1.64), with GamePad = 18, multi-touch = 10, and both = 0. The preferred device when switching to keyboard was the multi-touch display (mean = 1.25), with multi-touch = 23, GamePad = 3, and both = 2.

There was a preference for the multi-touch display when looking only at the casual gamers group. It was only a slight preference for the multi-touch display (mean = 1.53), with 8 users preferring the multi-touch display, and 7 preferring the GamePad.

#### 6.4.4 Rotation and Translations Questions

A table was provided, which asked the user to select which device was preferred for a specific rotation or translation (see example in Figure 6.8). It is important to remember that users were provided with a figure of an airplane. Additional help was provided by the experimenter if the user had questions about the rotations and translation. The questions were very specific, asking the user if they preferred the GamePad controller or the multi-touch display for a given operation. This question ( $Q_{14}$ ) is broken down into 6 categories, with  $Q_{14a}$  to  $Q_{14c}$  for the rotations and  $Q_{14d}$  to  $Q_{14f}$  for the translations, as shown in Table 6.16.

	GamePad Device	Multi-Touch Display
Rotation: Yaw		X
Rotation: Roll		X
Rotation: Pitch		X
Up – Down	X	
Left – Right	X	
Forward – Back	X	

Figure 6.8: Experiment Table

Question #	Type
$Q_{14a}$	Rotation: Yaw
$Q_{14b}$	Rotation: Roll
$Q_{14c}$	Rotation: Pitch
$Q_{14d}$	Translation: Up-Down
$Q_{14e}$	Translation: Left-Right
$Q_{14f}$	Translation: Forward-Back

Table 6.16: Question 14: GamePad or multi-touch

### 6.4.5 Other Questions

Several open questions were asked to the user. Some examples are shown in the next chapter. For three of the additional questions, the answers were grouped in categories by the experimenter. These were  $Q_{16}$ ,  $Q_{17}$ , and  $Q_{21}$ . The results are shown in Tables 6.17 and 6.18. From the perspective of the user, they preferred the GamePad for 3D navigation ( $Q_{16}$ ). When asked which device they preferred when switching to keyboard, the majority preferred the multi-touch display ( $Q_{17}$ ). For the three-ring sphere, the answers were generally positive ( $Q_{21}$ ).

### 6.4.6 Hold-And-Roll Questions

A subset of subjects of the entire sample ( $N = 28$ ), were asked to answer questions about the Hold-and-Roll gesture. The total number of subjects asked was twenty. In particular, two questions can be quantified to understand the preference of the users ( $Q_{23}, Q_{24}$ , and

Type	Frequency Count	
	$Q_{16}$	$Q_{17}$
No response	3	5
Multi-touch	8	19
Game-Pad	16	2
Both Devices	1	2

Table 6.17: Questions 16 and 17

Type #	Frequency
No response	3
Very negative	1
Negative	1
Neutral	5
Positive	13
Very Positive	5

Table 6.18: Question 21

$Q_{25}$ , see Table 5.8).  $Q_{23}$  had a scale from 1 to 10, with 10 being the highest ranking. The mean for  $Q_{23}$  was 7.70 ( $SEM = 0.31, SD = 1.526$ ), median of 8.0, mode of 7.0, and minimum/maximum of 5 and 10, respectively.  $Q_{24}$  asked the user if they would like to see this gesture incorporated into new games. From the 21 subjects asked, 13 of them said yes, 1 of them said no, and 7 of them said maybe.  $Q_{25}$  asked the user if they would like to see this gesture incorporated into new applications. From the 21 subjects asked, 14 of them said yes, 3 of them said no, and 4 of them said maybe. Finally, when looking at subjects that performed better with one device or the other, and their answer to  $Q_{24}$ , there was no statistical significance. There was also no statistical significance for users who preferred one device over the other (from  $Q_{12}$ ) in relation to  $Q_{24}$ . The same was found for  $Q_{25}$ .

## CHAPTER 7

### EXPERIMENT DISCUSSION

This chapters provides a discussion about the data analysis performed in Chapter 6, which includes quantitative and qualitative results. It also provides a discussion about open questions in the exit survey, the observations from the experimenters during the experiment trials, and presents a view of how all of this helps to understand 3D navigation using multi-touch desktop displays.

#### 7.1 Assimilating Experimental Results

It is common in HCI to experiment on different prototypes and techniques. This is very important in HCI. However, they are a few pointers that are important to have in mind. One of them is the bias towards objective, quantifiable measurements, treated as facts, without the incentive of re-testing by others [63]. Also, Greenberg and Buxton provided compelling arguments about one myth held by some people in the community. Some researchers may think that to have a successful experiment, the new device must outperform the old one. A counter example offered by the authors in [63] is Marconi's wireless radio (1901). This radio would not have passed any usability test and it was severely criticized at the time. This leads to the question whether a device is usable or useful (or both)? It is the opinion of this author (and the authors in [63]), the usefulness at early stages is far more important. Usability comes later in the process, as has been the experience in the field [20, 63].

$H_t$	$H_u$	$H_v$	$H_w$	$H_x$	$H_y$	$H_z$
False	False	True	False	True	False	True

Table 7.1: Hypothesis Results

## 7.2 3D Navigation

Chapter 1 provided a series of questions ( $Q$ ) with their hypotheses ( $H$ ) in Table 1.1. The first question  $Q_s$ , helps to place the experiment in context. This was the first question when this project started. Can a user navigate in 3D using a 2D multi-touch display? Given that the users were able to navigate, it does help to validate the hypothesis. It was clear that the subjects, all of them, were able to use a multi-touch display to complete the primed search tasks while navigating in 3D. The other questions helped to find the validity of specific hypotheses. The results from Chapter 6 are shown in Table 7.1.

The primary aim was to know if, when given a task (primed search) while navigating in 3D, there would be any significant difference (either way) between the multi-touch display and the GamePad controller. This was not the case. The significance value ( $p$ ) was not within the range that would make  $H_t$  true. The question remains: Why was it not significant? Is the interaction of both devices comparable? This author believes that there are a few factors that made a difference. The first factor is that there is a difference between experienced gamers and casual gamers. This is apparent when looking at hypothesis  $H_v$ , which resulted to be true. This meant that there was a significant difference in the time average of both groups when they used the GamePad. This was not the case for hypothesis  $H_u$  (when both groups used the multi-touch), where there was no significant difference. This meant that experienced gamers had exposure to the GamePad and time to train, in comparison to the other subjects. As noted in 6.3.1, experienced gamers did significantly better with the GamePad than with the multi-touch, 214.8 seconds and 239.4 seconds, respectively. Casual gamers did better with the multi-touch. The second factor, which may explain why  $H_t$  was not supported, is that experienced gamers may be better when navigating in 3D because of previous exposures. The other two hypotheses, that required looking at the groups separately (and running t-tests), showed that casual gamers ( $H_x$ ) did have a statistically significant difference in their average task time when both devices were con-

sidered. This meant that the multi-touch display allowed the casual gamers to finish the task in less time than the GamePad, as predicted in  $H_x$ . This was not the case for  $H_w$ , which showed no significant difference for experience gamers when using the GamePad.

Finally, the other question that this dissertation had was about the switching of devices. In hypothesis  $H_y$ , there was a prediction that the sentence completion would take less time when using the multi-touch display. However,  $H_y$  could not be supported. Is it possible that there is no difference? Twenty-three subjects out of 28 preferred the multi-touch display (with 2 subjects having no preference). This leads the author to think that further studies, with additional variables, can show a difference in time. Furthermore, this is corroborated when looking at the error rate hypothesis  $H_z$ . The result was statistically significant, stating that users would have a higher error rate when switching from the GamePad to the keyboard. This underlines the belief that a larger sample of subjects may lead to show that the assumption made in  $H_y$  was correct after all.

The qualitative data (see 6.4) provided some important insight, from the perspective of the subject. The assumption that  $Q_1$  and  $Q_2$  (see 6.4.1) would be correlated, with a negative relationship, proved to be correct. This meant that not only did users preferred the GamePad versus the multi-touch display, but when asked the question in reverse, they were consistent. It also showed that the preference of the GamePad versus the multi-touch for  $Q_2$  were statistically significant. Why did users preferred the GamePad, when there was no significant difference between them? Could this indicate that in a larger sample the GamePad performed significantly better than a multi-touch device? The possible answer for the former question, is that user did perceived the GamePad as a better device. This was probably due to the indirect nature of the device for the rotations. When users were performing rotations with the multi-touch displays, the rotation did not always match with the visual understanding of the action. When the user performed a few rotations, and then came back to rotate using the roll, for example, the subject expected to see the display

image to move in the same manner as before. However, this is not possible, since the roll movement will reflect the current state of the camera rotation. In the case of the GamePad, the user could adjust the device to match the interaction. The latter question, would seem to imply that that the GamePad would outperform the multi-touch display with a larger N. However, the results did not support this for all the groups. It may be a possibility with experienced gamers, but not for the casual gamers. For the other pairs of questions,  $Q_4$  to  $Q_{11}$  (see 6.4.2), there were only some pairs that were correlated, with a negative relationship. In the questions that were not correlated, it could be the case that subjects' answers were about the same for both devices. The questions that showed to be statistically significant for casual gamers were  $Q_6$  and  $Q_7$ . This showed a clear preference by users for the multi-touch desktop display in day-to-day use, regardless of the task at hand. Finally, users did prefer the GamePad over the multi-touch display. This is a subjective response that must be taken into consideration. This can be attributed to the fact that users are more accustomed to the GamePad interaction (at least the experienced gamers), or that the multi-touch interaction is still somewhat challenging for the users. This author believes in the latter. There still more work to do with multi-touch.

### **7.2.1 Open Questions**

Open questions found in 5.10 (see Tables 5.12 and 5.8) provided comments to improve the interaction in the near future. Some of the answers did not provide much information. Hence, they were ignored. Table 7.2 provides some of the comments given by the subjects. Some of the comments were extrapolated into categories for questions  $Q_{16}$ ,  $Q_{17}$ , and  $Q_{21}$ , as described in 6.4.5. Open questions are useful for future design, but it is the opinion of this author that their anecdotal nature does not help to see a trend, in most instances. It was clear that  $Q_{15}$  and  $Q_{20}$  were answered with positive statements or neutral statements, without

much thought. However, some of the other questions provide an insight into future design. Six major points were obtained with some of the anecdotal comments by the subjects:

1. Some users requested the pinch gestures to move forward and back.
2. Subjects found the multi-touch device easier to transition to the keyboard and back.
3. The nature of the direct interaction created cognitive disconnect with the rotations.
4. Users have different ideas of what the ideal gestures for 3D navigation are.
5. Some subjects found the GamePad to be easier.
6. Orientation indicators, such as the three-ring sphere, do provide orientation help.

### **7.2.2 Hold-and-Roll**

The Hold-and-Roll gesture provided a bi-manual interaction. The response was positive by most users asked about this gesture, as described in 6.4.6. While this is encouraging, users also requested the pinch gesture to be used for moving back and forth. This does need further study and it will be discussed in Chapter 8. There are some comments from question  $Q_{22}$ , which are very interesting to share. Some of the typical comments are shown in Table 7.3. A very detailed comment was provided in the last entry of the table.

## **7.3 Lessons learned**

The work in this dissertation and the experiment conducted provided some interesting insight into the interaction of multi-touch displays for 3D navigation. The hope is that it can be applied in future work, which is discussed in Chapter 8. The observations have also provided additional guidelines that may be applied for better interactions.

The most important lesson learned is the nature of direct interaction. In this type of interaction when using a multi-touch display, rotations can become problematic, as already



described in this chapter. Another important lesson is that users want to control the speed differently when using multi-touch displays. While this was provided, it seems to be more intuitive to think that the more you push a joystick, the more speed the user will have. In the case of the multi-touch display, the speed was given by the speed of the movement of the gesture. However, this must be studied further. The following recommendations are based on the experiences during the experiment, and the comments by the users:

- Rotation gestures must be dynamic. As the camera moves, the option for the gesture must also change.
- The mapping of gestures is important for each action. Further study is merited to find the most optimal gestures.
- Orientation gizmos are important for the interaction in 6-DOF.

## **7.4 Limitations of the Study**

The study has limitations. The first limitation is that it worked with a subset of gestures that were highly optimized for the environment. This is true for the GamePad as well. If the same set of gestures were used into another environment, while it is possible that the behavior would be comparable, it is unknown at this point. Also, due to the fact that the study was performed with a desktop multi-touch display, it made assumptions, which are not always true with tablets or phones. This is the case for the hold-and-roll gesture which is a bi-manual gesture developed for a desktop or tabletop surface. Finally, the objects in the universe were spread with significant distances between each other. Results may be different for a different type of 3D navigation where the objects are found very close to each other (in a dense environment). With this said, I find that our universe applies to many (but not all) environments. In the next sub-sections, the internal and external validity of the experiment are discussed.

### 7.4.1 Internal Validity

Internal Validity<sup>1</sup> “has to do with defending against sources of bias arising in research design, typically by affecting the process being studied by introducing covert variables” [58]. In other words, other variables may be responsible for the observed effect.

The experiment bias (similar to Hawthorne effect) was not shown during the experiment. Furthermore, the experimenter used a commercial multi-touch gesture API to avoid any bias toward his approach. It is also important to note that subjects were not aware that the experimenter preferred one device over the other. With this said, there is always a possibility that a subject may have assumed that the multi-touch or the GamePad was the preferred device. However, is the opinion of this author, that this was not an internal threat. This also helped to avoid as much as possible the “look good” effect, where the subject has a tendency to answer positive answers in favor of the experimenter. Having said this, it is always possible for some subjects to want to impress the experimenter. This was avoided by using objective and subjective measurements.

Avoiding selection bias was important during the experiment design phase. Subjects were recruited via emails sent to the School of Computer Information and Sciences and requesting participation from different classes in the Engineering College. We also received a few other subjects which were from different majors. The experimenter tried to avoid selection bias but a bigger universe of subjects could always help.

The experiment had two groups: casual and experienced gamers. However, the subjects were not aware that they were classified into those categories nor which category they belonged to. This made rivalry threat non-existent. The selection of the subjects into the groups was made in a post-hoc analysis with questions that the subjects didn't know where meant for categorization. The categorization also helped to minimize the test expe-

---

<sup>1</sup>See Wikipedia: Internal Validity for an overview.

rience. While the subjects were presented with the experiment only once, it was expected for some users to performed better with the GamePad because of their previous exposure to this device and the type of environments where the device was used (e.g., 3D games). This is very important because some confounding variables where suspected before the experiment, avoiding typical confounding threats. Nevertheless, it is possible that another confounding variable may had an effect. However, this author believes that the designed experiment was carefully planned to avoid this type of internal threats.

There are other threats to validity (e.g, maturation), however those where not applicable to this test. In summary, while there are some possible internal threats to be aware, the design was created with this in mind.

#### **7.4.2 External Validity**

External validity<sup>2</sup> “has to do with possible bias in the process of generalizing conclusions from a sample to a population, to other subject populations, to other settings, and/or to other time periods” [58]. This refers to the potential of this experiment to be generalized across situations and across people. In the case of the experiment found in this dissertation, the external validity plays an important role and sets a path for future experiments.

Already mentioned in the limitations of this study is the fact that this 3D universe is not dense in objects. This makes it hard to generalize the experiment for all types of 3D domains. This author found that one possible optimal way to test 6-DOF was to have a pseudo-universe where objects performed a search task. This is because navigation is a secondary task to achieve the primary task, which in this case was search. However, there are other possible scenarios. One example is navigating a real 3D city of Manhattan. It is not clear if the experiment found here may be generalized for that type of scenario.

---

<sup>2</sup>See Wikipedia: External Validity for an overview.

Another threat to external validity has to do with generalization of our categorization equation. It has already been mentioned that the author does not claim the equation to be a general model but a path forward to it. This also leads to the aptitude-treatment interaction effect. It is unclear that the results would be the same if other subjects, with different skills, had performed the experiment? The author finds that the subject categorization for 3D environments must be studied further and in detail to be able to achieve results that could be generalized.

Q#	Answer
15	It was a good experiment. I enjoyed looking for the items in the nebula.
15	Good experiment for user input.
16	I prefer the GamePad because I grew up with playing on physical controllers and find it easier.
16	Multi-touch is excellent for navigation in space. Not so, for FPS games.
16	I prefer the GamePad because I can make multiple inputs quickly, whereas as with the display, it took more effort.
16	Both, but the approach in GamePad provides more control. The multi-touch, too, but I wish to have the possibility to move my fingers in such a way that the program can interpret exactly what I want. Both are good, but a combination of both would be better. Something similar to the touch system used by Tony Stars in Ironman.
16	It's easier to find objects by using multi-touch.
16	I prefer multi-touch because the controls are more realistic than those of the gamepad.
16	I prefer multi-touch because I don't feel tired and also I can switch quickly between devices to the keyboard.
17	Not sure why, but I found typing a bit easier when doing multi-touch.
17	Game-Pad has to be set down. In multi-touch, all you have to do is begin typing.
17	I preferred the multi-touch because I didn't have to let go of a remote to type.
17	Typing was quicker if I did not have a device (GamePad) in my hand.
18	The zoom must be the "pinch" gesture.
18	I think it is useful, just that it has many options to rotate, which can confuse the user at times. Once you become expert, there will be no problem.
18	The rotation was more difficult to assimilate.
18	It is easy to control the 3D space camera because it's realistic when you control the camera by your hand.
20	I enjoyed the experiment. Perhaps maybe add more items and differentiate them so they have different patterns or colors so you really have to think before you choose the correct one.
20	Need more training time.
21	It was helpful in helping me orient myself, although it would probably be more helpful with experience or maybe it would eventually become so familiar that it would be annoying.
21	Extremely helpful. I would suggest some sort of indicator [for direction], like an arrow.

Table 7.2: Open Question Results<sup>¶</sup>

Table Legend: ¶ Sentences where modified only to correct grammar.

Q#	Answer
22	Needs work, but could be easier than pinch and zoom if you don't care about two hands.
22	Easy to implement, but should have easier speed adjustments.
22	I didn't like it very much.
22	Yes, it can be useful in an application.
22	Yes. It will be better if the "HOLD" part of the gesture can specify the center of rotation for a gesture.
22	I would think it would give people something new and would be great in new apps.
22	At first, it was counter-intuitive in terms of trying of zoom in using a touch screen interface Like most, I was used to holding my index finger and thumb together, then releasing them to enlarge or zoom in. However, after a few trials, I generally found the Hold-and-Roll gesture much simpler to use. It allows the user to generate one fluid forward-moving motion with one roll of the fingers and allows the user to seize motion when necessary, which as opposed to using the index finger and thumb to create the motion would take several movements to travel the idealized distance. Perhaps the Hold-and-Roll gesture is more time consuming but, at least from my experience, I had more time to react and control the movement.

Table 7.3: Hold-and-Roll Comments<sup>¶</sup>

Table Legend: ¶ Sentences where modified only to correct grammar.

## CHAPTER 8

### CONCLUSIONS & FUTURE WORK

#### 8.1 Concluding Remarks

This dissertation presented a novel approach to 3D navigation using multi-touch display devices. The contributions included a novel multi-touch recognition method (FETOUCH), a gyroscope and multi-touch interaction (GyroTouch), a multi-touch model using HLPN (PeNTa), a gesture conflict resolution technique (Yield), and a 3D navigation prototype for human-subject testing (3DNav). The 3D prototype included Yield, FaNS, and ECHoSS in its implementation. Also, the author created a categorization equation to divide the subjects into casual games and experienced gamers. Finally, the author proposed a novel gesture called Hold-and-Roll. This dissertation was concluded with an experiment to answer the questions postulated in Chapter 1.

When looking at the experiment conducted, this dissertation showed that experienced gamers can affect the comparison between GamePad and multi-touch devices. It also showed that casual gamers performed significantly faster when using the multi-touch display. Furthermore, the experiment showed that users performed a significantly higher number of errors when switching from the GamePad to the keyboard. However, the experiment was not able to conclude if there was a significant difference when looking at the entire subject pool between multi-touch and the GamePad, even though the multi-touch average time was lower than the time for the GamePad. This was attributed to the previous use of the GamePad by experienced gamers.

This dissertation described the proposed questions and hypotheses in Chapter 1, the background required to understand the material in Chapter 2, the contributions of the research in Chapters 3 and 4, the experiment design in Chapter 5, and the experiment data

and its conclusions in Chapters 6 and 7. The following list details the primary findings of this dissertation:

1. Casual gamers performed significantly better when they used the multi-touch display (analyzed as a group).
2. The ANOVA co-factor analysis (gamer experience) indicated that when looking at both groups, the experienced gamers performed significantly better when using the GamePad compared to casual gamers.
3. When users switched from the GamePad to the keyboard, they performed significantly worse in typing a target sentence. Each error meant that the user hit enter believing that they had entered a correct sentence, when in fact they had not.
4. While users took less time with the multi-touch display, the data analysis did not yield significant results.
5. In the exit survey, most users reported preferring the GamePad for 3D navigation.
6. In the exit survey, most users reported preferring the multi-touch display when they were required to switch to the keyboard and type target sentences.
7. An observation made by the experimenter was that when users performed rotation movements with the multi-touch display, the users required having a matching rotation for the orientation of their fingers.

In addition to the findings reported in Chapter 6 and the full discussion of Chapter 7, there were additional contributions. Those contributions are listed below:

- Proposed a set of gestures for 6-DOF 3D navigation, which included a new gesture called Hold-and-Roll for the Z axis.
- Proposed a framework for the 3D navigation experiment.



- Designed a feature extraction method for multi-touch gestures.
- Designed a mathematical framework for multi-touch interaction.
- Complemented multi-touch with Gyroscope.
- Designed an algorithm to manage the ambiguity that exists in the results of some gestures classifications methods.

With the above contributions, this dissertation added to the body of knowledge in the fields of 3DUI, HCI, and Computer Science. It is the vision of this author that modern input devices are changing the interaction of computer users, will revolutionized the paradigms of HCI, and will provide the building blocks required for ubiquitous computing.

## **8.2 Future Work**

Multi-touch surfaces have become pervasive and are expected to continue to be important devices in the immediate future. Multi-touch is not the only path to the post-WIMP era, but one of many devices to achieve ubiquitous computing. It is, in this author's opinion, one of the pillars of ubiquitous computing. It is expected that with multi-touch surface, researchers will keep pushing the body of knowledge with bendable surfaces, stereoscopic surfaces with multi-touch, and possibly many new devices that are yet to come. This will bring new challenges and problems to the field of 3DUI. It is the hope of this section to provide pointers for future work that will continue the spirit of this dissertation and beyond.

The most immediate need is to find a simple, standard algorithm that will work with multi-touch. It is this author's belief that the answer lies in previous work from stroke recognition. This author implemented a limited solution, but a full-fledged, general solution is still needed. In addition to this, as demonstrated with GyroTouch, finding ways of how multi-touch can interact with other modern devices is needed. To continue looking at new

problems, stereoscopic multi-touch, which has been worked on by some researchers, needs to continue being in the fore-front of the 3DUI evolution. In addition, as is customary in HCI, a set of experiments and a benchmark that can be used to validate multi-touch interaction for 3D navigation (and manipulation) still need to be expanded and optimized to gain deeper understanding of the interactions performed with these devices. This will also be shaped by the new devices to come.

Another aspect that requires the attention of 3DUI is the modeling of modern devices. In specific, the modeling of multi-touch interaction is also needed, as described by this author's contribution and other contributions. The multi-modal aspect of modern interaction requires a better modeling to test, analyze, and implement solutions. Finally, there needs to be better understanding of the type of users who are tested for 3D navigation. A predictive model that describes the type of user (e.g., the casual gamer) is required to understand the interaction between the users and the interfaces. This should be mathematically sound.

It is the hope of this author that this dissertation, and the continuous progress in the fields of 3DUI and HCI, encourages new and current practitioners and researchers to advance techniques, models, and interactions to new frontiers. This author hopes that Mark Weiser's vision is realized for ubiquitous computing [219]. This dissertation concludes with the words of the pioneer Ivan Sutherland [203]:

“The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal.”

## BIBLIOGRAPHY

- [1] Touch Technology Brief : Projected Capacitive Technology. Technical Report 3M PCT TECH BRIEF-1013, 2013.
- [2] M. J. Abásolo and J. M. Della. Magallanes: 3D navigation for everybody. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, GRAPHITE '07, page 135. ACM, Dec. 2007.
- [3] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering, Third*. A K Peters, Ltd., July 2008.
- [4] L. Anthony and J. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010*, GI'10, Toronto, ON, 2010.
- [5] P. Apostolellis, B. Laha, and D. Bowman. A Gaming Interface Using Body Gestures for Collaborative Navigation. In *IEEE Symposium on 3D User Interfaces (3DUI), 2012*, 3DUI '12, Mar. 2012.
- [6] J. Arvo. *Graphics Gems II*. Morgan Kaufmann, Oct. 1994.
- [7] R. S. Astur, M. L. Ortiz, and R. J. Sutherland. A characterization of performance by men and women in a virtual Morris water task:: A large and reliable sex difference. *Behavioural brain research*, 1998.
- [8] E. Beheshti, A. Van Devender, and M. Horn. Touch, click, navigate: comparing tabletop and desktop interaction for map navigation tasks. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*, ITS '12, pages 205–214. ACM, 2012.
- [9] H. Benko, A. Wilson, and P. Baudisch. Precise selection techniques for multi-touch screens. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 1263–1272. ACM, 2006.
- [10] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Springer, Berlin, Heidelberg, second edition, 2000.
- [11] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 73–80. ACM, 1993.

- [12] M. Blumenstein, B. Verma, and H. Basli. A novel feature extraction technique for the recognition of segmented handwritten characters. In *Seventh International Conference Proceedings on Document Analysis and Recognition, 2003*, pages 137–141. IEEE Computer Society, 2003.
- [13] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Levy. *Polygon Mesh Processing*. CRC Press, Oct. 2010.
- [14] D. Bowman, J. Chen, C. Wingrave, A. Ray, N. Polys, Q. Li, Y. Haciahmetoglu, and J. Kim. New directions in 3d user interfaces. *International Journal of Virtual Reality*, 5(2):3–14, 2006.
- [15] D. A. Bowman, D. B. Johnson, and L. F. Hodges. Testbed evaluation of virtual environment interaction techniques. In *Proceedings of the ACM symposium on Virtual reality software and technology, VRST '99*, pages 26–33. ACM, Dec. 1999.
- [16] D. A. Bowman, D. Koller, and L. F. Hodges. Travel in immersive virtual environments: an evaluation of viewpoint motion control techniques. In *Virtual Reality Annual International Symposium, 1997*, pages 45–52. IEEE Computer Society Press, 1997.
- [17] D. A. Bowman, E. Kruijff, J. J. LaViola, Jr, and I. Poupyrev. *3D user interfaces: theory and practice*. Addison-Wesley Professional, 2004.
- [18] A. Boyali and M. Kavakli. 3D and 6 DOF user input platform for computer vision applications and virtual reality. In *Symposium on Innovations in Intelligent Systems and Applications (INISTA), 2011 International*, INISTA '11, pages 258–263, 2011.
- [19] G. C. Burdea and P. Coiffet. *Virtual Reality Technology*. John Wiley & Sons, June 2003.
- [20] B. Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design*. Focal Press, Morgan Kaufmann, 2010.
- [21] W. Buxton. A three-state model of graphical input. *Human-computer interaction-INTERACT '90*, 90:449–456, 1990.
- [22] W. Buxton and B. Myers. A study in two-handed input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '86*, pages 321–326. ACM, 1986.

- [23] R. Camiciottoli, J. M. Corrifoni, A. d. Bimbo, E. Vicario, and D. Lucarella. 3D navigation of geographic data sets. *MultiMedia, IEEE*, 5(2):29–41, 1998.
- [24] X. Cao, A. Wilson, R. Balakrishnan, K. Hinckley, and S. Hudson. ShapeTouch: Leveraging contact shape on interactive surfaces. *Horizontal Interactive Human Computer Systems, 2008. 3rd IEEE International Workshop on TABLETOP 2008*, pages 129–136, 2008.
- [25] S. K. Card, T. P. Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23(7):396–410, 1980.
- [26] D. Catuhe. *Programming with the Kinect for Windows Software Development Kit*. Microsoft Press, 2012.
- [27] X. J. Chai and L. F. Jacobs. Effects of cue types on sex differences in human spatial memory. *Behavioural brain research*, 2010.
- [28] L. Chang. *A Nested Petri Net Framework for Modeling and Analyzing Multi-Agent Systems*. PhD thesis, Florida International University, 2011.
- [29] M. Chen, S. J. Mountford, and A. Sellen. A study in interactive 3-d rotation using 2-d control devices. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '88*, pages 121–129. ACM, 1988.
- [30] Y.-R. Chen, T. Wang, X. Chen, and X. Bai. Research on navigation method in 3d geographic information system for water conservancy projects of large basin. In *Education Technology and Training, 2008. and 2008 International Workshop on Geoscience and Remote Sensing. ETT and GRS 2008. International Workshop on*, volume 2, pages 521–524, Dec 2008.
- [31] D. Coffey, N. Malbraaten, T. B. Le, I. Borazjani, F. Sotiropoulos, A. G. Erdman, and D. F. Keefe. Interactive Slice WIM: Navigating and Interrogating Volume Data Sets Using a Multisurface, Multitouch VR Interface. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1614–1626, 2012.
- [32] M. Czerwinski, D. S. Tan, and G. G. Robertson. Women take a wider view. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '02*, pages 195–202, New York, New York, USA, Apr. 2002. ACM.

- [33] R. P. Darken and W. P. Banker. Navigating in natural environments: a virtual environment training transfer study. In *Proceedings on Virtual Reality Annual International Symposium*, pages 12–19. IEEE Computer Society, 1998.
- [34] R. P. Darken and J. L. Sibert. Wayfinding strategies and behaviors in large virtual worlds. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '96, pages 142–149, New York, New York, USA, Apr. 1996. ACM.
- [35] R. David and H. Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer, Nov. 2010.
- [36] P. Dietz and D. Leigh. DiamondTouch: a multi-user touch technology. *Proceedings of the 14th annual ACM symposium on User interface software and technology*, page 226, 2001.
- [37] A. Dix, J. Finlay, G. D. Abowd, and R. Beale. *Human-computer Interaction*. Pearson Education, 2004.
- [38] N. Doulamis and C. Yiakoumettis. Personalised 3D navigation and understanding of Geo-referenced Scenes. In *IEEE 14th International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013, WowMoM '13*, pages 1–6, June 2013.
- [39] F. Dunn and I. Parberry. *3D Math Primer for Graphics and Game Development*. A K Peters/CRC Press, second edition, Nov. 2011.
- [40] F. A. Ebeling, R. L. Johnson, and R. S. Goldhor. Infrared Light Beam XY Position Encoder for Display Devices. US Patent 3,775,560, 1973.
- [41] D. H. Eberly. *3D Game Engine Architecture*. Engineering Real-Time Applications with Wild Magic. Elsevier, 2005.
- [42] D. H. Eberly. *3D Game Engine Design*. A Practical Approach to Real-Time Computer Graphics. Gulf Professional Publishing, 2007.
- [43] D. H. Eberly. *Game Physics*. Morgan Kaufmann, Apr. 2010.
- [44] J. Edelmann, A. Schilling, and S. Fleck. The DabR - A multitouch system for intuitive 3D scene navigation. In *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video, 2009*, pages 1–4. IEEE, 2009.

- [45] W. El Oraiby. Scene Management. In D. Astle, editor, *More OpenGL Game Programming*, pages 565–605. Thomson Course Technology, 2004.
- [46] W. K. English, D. C. Engelbart, and M. L. Berman. Display-Selection Techniques for Text Manipulation. *IEEE Transactions on Human Factors in Electronics*, (1):5–15, 1967.
- [47] C. Ericson. *Real-time Collision Detection*. Morgan Kaufmann, 2005.
- [48] A. P. Field. *Discovering Statistics Using SPSS for Windows*. SAGE, third edition, 2009.
- [49] G. Fitzmaurice, J. Matejka, I. Mordatch, A. Khan, and G. Kurtenbach. *Safe 3D navigation*. Proceedings of the 2008 symposium on Interactive 3D graphics and games (I3D '08), Feb 2008.
- [50] G. W. Fitzmaurice, H. Ishii, and W. A. S. Buxton. Bricks: laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95. ACM Press/Addison-Wesley Publishing Co. Request Permissions, May 1995.
- [51] J. D. Foley, A. Van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles & Practice In C*. Pearson Education India, second edition, Sept. 1996.
- [52] D. Fryberger and R. G. Johnson. Touch actuatable data input panel assembly. US Patent 3,673,327, 1972.
- [53] C.-W. Fu, W. B. Goh, and J. A. Ng. Multi-touch techniques for exploring large-scale 3D astrophysical simulations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2213–2222. ACM, Apr. 2010.
- [54] R. Fuchs and H. Hauser. Visualization of Multi-Variate Scientific Data. *Computer Graphics Forum*, 28(6):1670–1690, 2009.
- [55] L. Gallo, G. De Pietro, and I. Marra. 3d interaction with volumetric medical data: Experiencing the wiimote. In *Proceedings of the 1st International Conference on Ambient Media and Systems*, Ambi-Sys '08, pages 14:1–14:6, 2008.
- [56] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Elements of Reusable Object-Oriented Software. Pearson Education, Oct. 1994.

- [57] G. Gan, C. Ma, and J. Wu. *Data Clustering: Theory, Algorithms, and Applications (ASA-SIAM Series on Statistics and Applied Probability)*. SIAM, Society for Industrial and Applied Mathematics, May 2007.
- [58] G. D. Garson. *Validity and Reliability*. Statistical Associates Blue Book Series. Statistical Associates Publishers, Kindle edition, Feb 2013.
- [59] H. J. Genrich. Predicate/transition nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri nets 1986, part I on Petri nets: central models and their properties*. Springer, Jan 1987.
- [60] H. J. Genrich and K. Lautenbach. System modelling with high-level Petri nets. *Theoretical computer science*, 13(1):109–135, 1981.
- [61] A. S. Glassner. *Graphics Gems*. Morgan Kaufmann, June 1993.
- [62] W. D. Gray, B. E. John, and M. E. Atwood. Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance. *Human-Computer Interaction*, 8(3):237–309, 1993.
- [63] S. Greenberg and B. Buxton. Usability evaluation considered harmful (some of the time). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08. ACM, Apr. 2008.
- [64] M. Gregoire, N. A. Solter, and S. J. Kleper. *Professional C++*. John Wiley & Sons, Sept. 2011.
- [65] I. Grinblat and A. Peterson. *OGRE 3D 1.7 Application Development Cookbook*. Packt Publishing Ltd, 2012.
- [66] F. Guéniat, J. Christophe, Y. Gaffary, A. Girard, and M. Ammi. Tangible windows for a free exploration of wide 3D virtual environment. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, VRST '13, page 115, New York, New York, USA, Oct. 2013. ACM.
- [67] Y. Guiard. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Journal of motor behavior*, 19:486–517, 1987.
- [68] M. Hachet, B. Bossavit, A. Cohé, and J.-B. de la Rivière. Toucheo: multitouch and stereo combined in a seamless workspace. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 587–592, New York, New York, USA, Oct. 2011. ACM.



- [69] M. Hachet, F. Declé, and P. Guitton. Z-Goto for efficient navigation in 3D environments from discrete inputs. In *Proceedings of the ACM symposium on Virtual reality software and technology, VRST '06*, pages 236–239, New York, New York, USA, Nov. 2006. ACM.
- [70] M. Hachet, F. Declé, S. Knödel, and P. Guitton. Navidget for 3D interaction: Camera positioning and further uses. *International Journal of Human-Computer Studies*, 67(3):225–236, Mar. 2009.
- [71] B. Hagedorn and J. Döllner. Sketch-Based Navigation in 3D Virtual Environments. In A. Butz, B. Fisher, P. Olivier, and M. Christie, editors, *Proceedings of the 9th international symposium on Smart Graphics, SG '08*, pages 239–246. Springer-Verlag, Aug. 2008.
- [72] P. Haigron, G. Le Berre, and J. L. Coatrieux. 3D navigation in medicine. *Engineering in Medicine and Biology Magazine, IEEE*, 15(2):70–78, 1996.
- [73] R. R. Hainich and O. Bimber. *Displays. Fundamentals and Applications*. CRC Press, July 2011.
- [74] K. S. Hale and K. M. Stanney. *Handbook of Virtual Environments. Design, Implementation, and Applications*. CRC Press, Jan. 2002.
- [75] A. Hamon, P. Palanque, J. L. Silva, Y. Deleris, and E. Barboni. Formal description of multi-touch interactions. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems, EICS '13*, pages 207–216, New York, New York, USA, June 2013. ACM.
- [76] J. Han. Low-cost multi-touch sensing through frustrated total internal reflection. *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118, 2005.
- [77] J. Han. *3D Graphics for Game Programming*. Chapman & Hall, Feb. 2011.
- [78] M. Hancock, S. Carpendale, and A. Cockburn. Shallow-depth 3d interaction: design and evaluation of one-, two- and three-touch techniques. In *Proceedings of the SIGCHI conference on Human Factors in computing systems, CHI '07*, pages 1147–1156. ACM, 2007.
- [79] C. Hand. A survey of 3D interaction techniques. *Computer Graphics Forum*, 16(5):269–281, 1997.

- [80] A. J. Hanson and E. A. Wernert. Constrained 3d navigation with 2d controllers. In *Proceedings of the 8th Conference on Visualization '97, VIS '97*, pages 175–182. IEEE, 1997.
- [81] J. S. Harbour. *Beginning Game Programming*. Cengage Learning, third edition, 2010.
- [82] X. He and T. Murata. High-Level Petri Nets-Extensions, Analysis, and Applications. In W. Chen, editor, *Electrical Engineering Handbook*, pages 459–475. Elsevier Academic Press, 2005.
- [83] P. S. Heckbert. *Graphics Gems IV*. Morgan Kaufmann, 1994.
- [84] M. L. Heilig. Sensorama simulator. US Patent 3,050,870 A, August 1962.
- [85] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, first edition, Mar. 2008.
- [86] C. F. Herot and G. Weinzapfel. One-point touch input of vector information for computer displays. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '78*, pages 210–216. ACM, 1978.
- [87] M. A. Hiltzik. *Dealers of Lightning. Xerox PARC and the Dawn of the Computer Age*. HarperCollins, May 2009.
- [88] K. Hinckley. Input Technologies and Techniques. In A. Sears and J. A. Jacko, editors, *Human-Computer Interaction*, pages 161–176. CRC, New York, 2012.
- [89] K. Hinckley, M. Pahud, and B. Buxton. 38.2: Direct Display Interaction via Simultaneous Pen+ Multi-touch Input. *Society for Information Display SID Symposium Digest of Technical Papers*, 41:537–540, May 2010.
- [90] P. Hong and T. Huang. Constructing finite state machines for fast gesture recognition. In *15th International Conference on Pattern Recognition*, volume 3 of *ICPR'00*, 2000.
- [91] P. Hong, T. Huang, and M. Turk. Gesture modeling and recognition using finite state machines. *IEEE Conference on Face and Gesture Recognition*, Mar. 2000.
- [92] J. F. Hughes, A. Van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley. *Computer Graphics. Principles and Practice*. Addison-Wesley Professional, July 2013.

- [93] A. Hülsmann and J. Maicher. HOUDINI: Introducing Object Tracking and Pen Recognition for LLP Tabletops. In *Human-Computer Interaction. Advanced Interaction Modalities and Techniques*, pages 234–244. Springer International Publishing, Cham, Switzerland, 2014.
- [94] E. C. Ifeachor and B. W. Jervis. *Digital Signal Processing. A Practical Approach*. Pearson Education, 2002.
- [95] B. Jackson, D. Schroeder, and D. F. Keefe. Nailing down multi-touch: anchored above the surface interaction for 3D modeling and navigation. In *Proceedings of Graphics Interface 2012, GI '12*. Canadian Information Processing Society, May 2012.
- [96] R. Jacob, A. Girouard, L. Hirshfield, M. S. Horn, O. Shaer, E. T. Solovey, and J. Zigelbaum. Reality-based interaction: a framework for post-WIMP interfaces. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, CHI '08*, pages 201–210. ACM, 2008.
- [97] C. Jennett, A. L. Cox, P. Cairns, S. Dhoparee, A. Epps, T. Tijs, and A. Walton. Measuring and defining the experience of immersion in games. *International Journal of Human-Computer Studies*, 66(9), Sept. 2008.
- [98] K. Jensen and L. Kristensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Springer, 1996.
- [99] G. Johnson, M. Gross, and J. Hong. Computational support for sketching in design: a review. *Foundations and Trends in Human-Computer Interaction 2*, 2009.
- [100] G. Junker. *Pro OGRE 3D Programming*. Apress, Sept. 2006.
- [101] P. Kabbash, W. Buxton, and A. Sellen. Two-handed input in a compound task. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '94*, pages 417–423. ACM, Apr. 1994.
- [102] P. Kabbash, I. S. MacKenzie, and W. Buxton. Human performance using computer input devices in the preferred and non-preferred hands. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems, CHI '93*, pages 474–481. IOS Press, 1993.
- [103] G. Kaindl. *Exploring multi-touch interaction: An overview of the history, HCI issues and sensor technology of multi-touch appliances*. VDM Verlag, Germany, Apr. 2010.

- [104] M. Kaltenbrunner. reactIVision and TUIO: a tangible tabletop toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, pages 9–16. ACM, Nov. 2009.
- [105] M. Kaltenbrunner, T. Bovermann, and R. Bencina. TUIO: A protocol for tabletop tangible user interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, GW 2005, 2005.
- [106] D. Kammer, J. Wojdziak, M. Keck, R. Groh, and S. Taranko. Towards a formalization of multi-touch gestures. In *International Conference on Interactive Tabletops and Surfaces*, ITS '10. ACM, Nov. 2010.
- [107] L. Kara and T. Stahovich. An image-based, trainable symbol recognizer for hand-drawn sketches. *Computers & Graphics*, 29(4):501–517, 2005.
- [108] J. S. Kelso, D. L. Southard, and D. Goodman. On the coordination of two-handed movements. *Journal of experimental psychology. Human perception and performance*, 5(2):229–238, 1979.
- [109] F. Kerger. *Ogre 3D 1.7 Beginner's Guide*. Packt Publishing Ltd, 2010.
- [110] A. Khan, B. Komalo, J. Stam, G. Fitzmaurice, and G. Kurtenbach. Hovercam: Interactive 3d navigation for proximal object inspection. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 73–80. ACM, 2005.
- [111] K. Kin, M. Agrawala, and T. DeRose. Determining the benefits of direct-touch, bimanual, and multifinger input on a multitouch workstation. In *Proceedings of Graphics Interface 2009*, GI '09, pages 119–124. Canadian Information Processing Society, may 2009.
- [112] K. Kin, B. Hartmann, T. DeRose, and M. Agrawala. Proton++: a customizable declarative multitouch framework. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12. ACM, Oct. 2012.
- [113] Y. Kiriaty, L. Moroney, S. Goldshtein, and A. Fliess. *Introducing Windows 7 for Developers*. Microsoft Press, Sept. 2009.
- [114] R. Kosara, H. Hauser, and D. L. Gresh. An interaction view on information visualization. *Proceedings EuroGraphics 2003: State-of-the-Art Report*, 2003.

- [115] S. Kratz and M. Rohs. A \$3 gesture recognizer: simple gesture recognition for devices equipped with 3D acceleration sensors. In *Proceedings of the 15th international conference on Intelligent user interfaces*, IUI '10, pages 341–344, New York, New York, USA, Feb. 2010. ACM.
- [116] S. G. Kratz and M. Rohs. Protractor3D: a closed-form solution to rotation-invariant 3D gestures. In *Proceedings of the 16th international conference on Intelligent user interfaces*, IUI '11, pages 371–374, 2011.
- [117] P.-O. Kristensson and S. Zhai. SHARK2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST '04, page 43, New York, New York, USA, Oct. 2004. ACM.
- [118] M. W. Krueger, T. Gionfriddo, and K. Hinrichsen. VIDEOPLACE—an artificial reality. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '85, pages 35–40, New York, New York, USA, Apr. 1985. ACM.
- [119] R. Kruger, S. Carpendale, S. Scott, and A. Tang. Fluid integration of rotation and translation. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '05, pages 601–610. ACM, 2005.
- [120] A. Kulshreshth, J. J. LaViola, and Jr. Evaluating performance benefits of head tracking in modern video games. In *Proceedings of the 1st symposium on Spatial user interaction*, SUI '13, pages 53–60. ACM, July 2013.
- [121] E. Langetepe and G. Zachmann. *Geometric data structures for computer graphics*. A K Peters, Ltd., 2006.
- [122] S. Lao, X. Heng, G. Zhang, Y. Ling, and P. Wang. A gestural interaction design model for multi-touch displays. *Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology (BCS-HCI '09)*, pages 440–446, 2009.
- [123] J. F. Lapointe, P. Savard, and N. G. Vinson. A comparative study of four input devices for desktop virtual walkthroughs. *Computers in Human Behavior*, 27(6):2186–2191, Nov. 2011.
- [124] C. A. Lawton. Gender differences in way-finding strategies: Relationship to spatial ability and spatial anxiety. *Sex Roles*, 1994.

- [125] S. Lee, W. Buxton, and K. C. Smith. A multi-touch three dimensional touch-sensitive tablet. pages 21–25, 1985.
- [126] A. Leganchuk, S. Zhai, and W. Buxton. Manual and cognitive benefits of two-handed input: an experimental study. *Transactions on Computer-Human Interaction (TOCHI)*, 5(4):326–359, Dec. 1998.
- [127] Y. Li. Protractor: a fast and accurate gesture recognizer. In *Proceedings of the 28th international conference on Human factors in computing systems, CHI '10*. ACM, 2010.
- [128] S. Liu, R. Zeng, and X. He. PIPE-A Modeling Tool for High Level Petri Nets. 2011.
- [129] B. Loguidice and M. Barton. *Vintage Game Consoles. An Inside Look at Apple, Atari, Commodore, Nintendo, and the Greatest Gaming Platforms of All Time*. CRC Press, Feb. 2014.
- [130] H. Lü and Y. Li. Gesture coder: a tool for programming multi-touch gestures by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '12*, pages 2875–2884. ACM, 2012.
- [131] Luna. *Introduction to 3d Game Programming With DirectX 11*. Jones & Bartlett Publishers, July 2011.
- [132] C. Lundstrom, T. Rydell, C. Forsell, A. Persson, and A. Ynnerman. Multi-Touch Table System for Medical Visualization: Application to Orthopedic Surgery Planning. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1775–1784, 2011.
- [133] I. S. Mackenzie. *Human-Computer Interaction: An Empirical Research Perspective*. Morgan Kaufmann;, Dec. 2012.
- [134] S. MacLean and G. Labahn. Elastic matching in linear time and constant space. In *International Workshop on Document Analysis Systems 2010, DAS '10*, 2010.
- [135] D. B. Makofske, M. J. Donahoo, and K. L. Calvert. *TCP/IP Sockets in C#. Practical Guide for Programmers*. Morgan Kaufmann, 2004.
- [136] J. McCrae, I. Mordatch, M. Glueck, and A. Khan. Multiscale 3D Navigation. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D '09*, pages 7–14. ACM, 2009.

- [137] M. McShaffry. *Game Coding Complete, Fourth*. Cengage Learning, fourth edition, 2013.
- [138] J. C. Meng and M. Halle. Using a 2D colon to guide 3D navigation in virtual colonoscopy. In *Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, APGV '04, page 179, New York, New York, USA, Aug. 2004. ACM.
- [139] F. C. Moon. *The Machines of Leonardo Da Vinci and Franz Reuleaux: kinematics of machines from the Renaissance to the 20th Century*. Springer, 2007.
- [140] K. H. Mortensen. Efficient data-structures and algorithms for a coloured Petri nets simulator. pages 57–74, 2001.
- [141] M. E. Mortenson. *Geometric modeling*. Industrial Press Inc., New York, third edition, 2006.
- [142] T. Moscovich and J. Hughes. Indirect mappings of multi-touch input using one and two hands. CHI '08, pages 1275–1284. ACM, 2008.
- [143] T. Moscovich and J. F. Hughes. Indirect mappings of multi-touch input using one and two hands. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1275–1284, New York, New York, USA, Apr. 2008. ACM.
- [144] R. Mukundan. *Advanced Methods in Computer Graphics. With Examples in OpenGL*. Springer Science & Business Media, London, Feb. 2012.
- [145] C. Müller-Tomfelde. *Tabletops: Horizontal Interactive Displays*, 2010.
- [146] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
- [147] B. A. Myers. A new model for handling input. *ACM Transactions on Information Systems (TOIS)*, 8(3):289–320, 1990.
- [148] M. A. Nacenta, P. Baudisch, H. Benko, and A. Wilson. Separability of spatial manipulations in multi-touch interfaces. In *Proceedings of Graphics Interface 2009*, GI '09. Canadian Information Processing Society, May 2009.

- [149] M. Naef and E. Ferranti. Multi-touch 3D navigation for a building energy management system. *IEEE Symposium on 3D User Interfaces (3DUI), 2011*, pages 113–114, 2011.
- [150] L. H. Nakatani and J. A. Rohrlich. Soft machines: A philosophy of user-computer interface design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '83*, pages 19–23, New York, New York, USA, Dec. 1983. ACM.
- [151] Y. Nam, N. Wohn, and H. Lee-Kwang. Modeling and recognition of hand gesture using colored Petri nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 29(5):514–521, 1999.
- [152] H.-I. P. Nets-Concepts. Definitions and graphical notation. *Final Draft International Standard ISO/IEC, 15909*, 2000.
- [153] W. M. Newman. A system for interactive graphical programming. pages 47–54, 1968.
- [154] J. Nielsen. *Usability Engineering*. Elsevier, Nov. 1994.
- [155] G. Nielson and D. Olsen Jr. Direct manipulation techniques for 3D objects using 2D locator devices. *Proceedings of the 1986 workshop on Interactive 3D graphics*, pages 175–182, 1987.
- [156] F. Ortega, A. Barreto, N. Rishe, M. Adjouadi, and F. Abyarjoo. Poster: Real-Time Gesture Detection for Multi-Touch Devices. In *IEEE 8th Symposium on 3D User Interfaces, 3DUI '13*, pages 167–168. IEEE, 2013.
- [157] F. R. Ortega, A. Barreto, and N. Rishe. Augmenting multi-touch with commodity devices. In *Proceedings of the 1st symposium on Spatial user interaction, SUI '13*, page 95, New York, New York, USA, July 2013. ACM.
- [158] F. R. Ortega, A. Barreto, N. D. Rishe, M. Adjouadi, and F. Abyarjoo. GyroTouch: Complementing the Multi-Touch Display. In *ACM Richard Tapia Celebration of Diversity in Computing*, Seattle, Feb. 2014.
- [159] F. R. Ortega, F. Hernandez, A. Barreto, N. D. Rishe, M. Adjouadi, and S. Liu. Exploring modeling language for multi-touch systems using petri nets. In *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces, ITS '13*. ACM, Oct. 2013.



- [160] F. R. Ortega, S. Liu, F. Hernandez, A. Barreto, N. Rische, and M. Adjouadi. PeNTa: Formal Modeling for Multi-touch Systems Using Petri Net. In *Human-Computer Interaction. Theories, Methods, and Tools*, pages 361–372. Springer International Publishing, Cham, Switzerland, Jan. 2014.
- [161] F. R. Ortega, N. Rische, A. Barreto, F. Abyarjoo, and M. Adjouadi. Multi-Touch Gesture Recognition using Feature Extraction. *Innovations and Advances in Computer, Information, Systems Sciences, and Engineering. Lecture Notes in Electrical Engineering*, 152, 2013.
- [162] R. Parent. *Computer Animation. Algorithms and Techniques*. Newnes, second edition, 2008.
- [163] J.-H. Park and T. Han. LLP+: multi-touch sensing using cross plane infrared laser light for interactive based displays. *SIGGRAPH 2010 Posters*, page 1, July 2010.
- [164] J. L. Peterson. *Petri net theory and the modeling of systems*. Prentice Hall, 1981.
- [165] C. Petzold. *Programming Windows*. Microsoft Press, fifth edition, 1998.
- [166] C. Petzold. *Programming Windows. Writing Windows 8 Apps With C# and XAML*. Microsoft Press, sixth edition, 2013.
- [167] R. W. Pew and S. Baron. Perspectives on human performance modelling. *Automatica*, 19(6):663–676, 1983.
- [168] E. Piphoo. *Focus on 3D Models*. Thomson Course Technology, 2003.
- [169] J. Pittman. Recognizing handwritten text. In *Human factors in computing systems: Reaching through technology*, CHI '91, pages 271–275. ACM, 1991.
- [170] R. Plamondon and S. N. Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.
- [171] I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa. The Go-go Interaction Technique: Non-linear Mapping for Direct Manipulation in VR. pages 79–80, 1996.
- [172] I. Poupyrev, S. Weghorst, and S. Fels. Non-isomorphic 3D rotational techniques. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '00, pages 540–547. ACM, 2000.

- [173] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes. the art of scientific computing*. Cambridge University Press, Hong Kong, third edition, 2007.
- [174] W. Reisig. *Understanding Petri Nets: Modelins Techniques, Analysis Methods, Case Studies*. Springer, July 2012.
- [175] J. Reisman, P. Davidson, and J. Han. A screen-space formulation for 2D and 3D direct manipulation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, UIST '09, pages 69–78. ACM, 2009.
- [176] A. Remazeilles, F. Chaumette, and P. Gros. 3D navigation based on a visual memory. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006, ICRA 2006*, pages 2719–2725. IEEE, 2006.
- [177] G. Robertson, M. Czerwinski, and M. van Dantzich. Immersion in desktop virtual reality. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97. ACM, Oct. 1997.
- [178] Y. Rogers, H. Sharp, and J. Preece. *Interaction Design. Beyond Human - Computer Interaction*. John Wiley & Sons, June 2011.
- [179] T. Ropinski, F. Steinicke, and K. Hinrichs. A constrained road-based VR navigation technique for travelling in 3D city models. In *Proceedings of the 2005 international conference on Augmented tele-existence, ICAT '05*, page 228. ACM, Dec. 2005.
- [180] D. Rubine. Specifying gestures by example. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques, SIGGRAPH '91*, pages 329–337. ACM, July 1991.
- [181] R. A. Ruddle, S. J. Payne, and D. M. Jones. Navigating large-scale virtual environments: what differences occur between helmet-mounted and desk-top displays? *Precense: Teleoperators and Virtual Environments*, 8(2):157–168, 1999.
- [182] S. Rümelin, E. Rukzio, and R. Hardy. NaviRadar: A Novel Tactile Information Display for Pedestrian Navigation. pages 293–302, 2011.
- [183] C. Russo dos Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud, and J. P. Paris. Metaphor-aware 3D navigation. In *IEEE Symposium on Information Visualization, InfoVis 2000*, pages 155–165, 2000.

- [184] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.
- [185] S. R. Santos, S. R. d. dos Santos, and P. M. Duarte. Supporting Search Navigation by Controlled Camera Animation. *2011 XIII Symposium on Virtual Reality*, pages 207–216, May 2011.
- [186] C. Saona-Vazquez, I. Navazo, and P. Brunet. The visibility octree: a data structure for 3D navigation. *Computers & Graphics*, 23(5):635–643, Oct. 1999.
- [187] C. Scholliers, L. Hoste, B. Signer, and W. De Meuter. Midas: a declarative multi-touch interaction framework. pages 49–56, 2011.
- [188] L. Schomaker and E. Segers. Finding features used in the human reading of cursive handwriting. *International Journal on Document Analysis*, 2(1):13–18, 1999.
- [189] T. Sezgin and R. Davis. HMM-based efficient sketch recognition. *Proceedings of the 10th international conference on Intelligent user interfaces (IUI '05)*, 2005.
- [190] D. Shreiner and B. T. K. O. A. W. Group. *OpenGL Programming Guide*. The Official Guide to Learning OpenGL version 4.3. Pearson Education, Mar. 2013.
- [191] B. Signer, U. Kurmann, and M. C. Norrie. iGesture: A General Gesture Recognition Framework. In *Ninth International Conference on Document Analysis and Recognition, 2007, ICDAR 2007*, pages 954–958. IEEE, 2007.
- [192] M. Sipser. *Introduction to Theory of Computation*. Cengage, second edition, 2006.
- [193] O. Sommer, A. Dietz, and R. Westermann. An interactive visualization and navigation tool for medical volume data. *Computers & Graphics*, Jan. 1999.
- [194] H. Song, H. Benko, F. Guimbretiere, S. Izadi, X. Cao, and K. Hinckley. Grips and gestures on a multi-touch pen. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11*, page 1323, New York, New York, USA, May 2011. ACM.
- [195] B. Sousa Santos, P. Dias, A. Pimentel, J.-W. Baggerman, C. Ferreira, S. Silva, and J. Madeira. Head-mounted display versus desktop for 3D navigation in virtual reality: a user study. *Multimedia Tools and Applications*, 41(1):161–181, Aug. 2008.
- [196] B. Sousa Santos, B. Prada, H. Ribeiro, P. Dias, S. Silva, and C. Ferreira. Wiimote as an Input Device in Google Earth Visualization and Navigation: A User Study Com-

paring Two Alternatives. In *Information Visualisation (IV), 2010 14th International Conference, IV 2010*, pages 473–478, July 2010.

- [197] L. D. Spano. Developing Touchless Interfaces with GestIT. *Ambient Intelligence*, 2012.
- [198] L. D. Spano, A. Cisternino, and F. Paternò. A compositional model for gesture definition. In *Proceedings of the 4th international conference on Human-Centered Software Engineering, HCSE'12*, pages 34–52, Berlin, Heidelberg, Oct. 2012. Springer-Verlag.
- [199] L. D. Spano, A. Cisternino, F. Paternò, and G. Fenu. GestIT: a declarative and compositional framework for multiplatform gesture definition. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems, EICS '13*, pages 187–196. ACM, June 2013.
- [200] S. L. Stoev, D. Schmalstieg, and W. Straßer. Two-handed through-the-lens-techniques for navigation in virtual environments. In *Proceedings of the 7th Eurographics conference on Virtual Environments & 5th Immersive Projection Technology, EGVE'01*. Eurographics Association, Jan. 2001.
- [201] N. Sultanum, E. V. Brazil, and M. C. Sousa. Navigating and annotating 3D geological outcrops through multi-touch interaction. In *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces, ITS '13*. ACM, Oct. 2013.
- [202] I. E. Sutherland. Sketchpad: a man-machine graphical communication system. In *AFIPS '63 (Spring): Proceedings of the May 21-23, 1963, spring joint computer conference*. ACM, May 1963.
- [203] I. E. Sutherland. The Ultimate Display, invited lecture. In *IFIP Congress*, 1965.
- [204] D. S. Tan, G. G. Robertson, and M. Czerwinski. Exploring 3D navigation: combining speed-coupled flying with orbiting. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '01*, pages 418–425. ACM, Mar. 2001.
- [205] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

- [206] M. S. Terlecki and N. S. Newcombe. How important is the digital divide? The relation of computer and videogame usage to gender differences in mental rotation ability. *Sex Roles*, 2005.
- [207] T. Theoharis, G. Papaioannou, N. Platis, and N. M. Patrikalakis. *Graphics and Visualization*. Principles & Algorithms. CRC Press, May 2008.
- [208] D. R. Trindade and A. B. Raposo. Improving 3D navigation in multiscale environments using cubemap-based techniques. In *Proceedings of the 2011 ACM Symposium on Applied Computing, SAC '11*, page 1215, New York, New York, USA, Mar. 2011. ACM.
- [209] B. Ullmer and H. Ishii. The MetaDESK: Models and Prototypes for Tangible User Interfaces. *ACM Symposium on User Interface Software and Technology*, pages 223–232, 1997.
- [210] D. Valkov, F. Steinicke, G. Bruder, and K. Hinrichs. A multi-touch enabled human-transporter metaphor for virtual 3D traveling. In *IEEE Symposium on 3D User Interfaces 2010, 3DUI '10*, pages 79–82, 2010.
- [211] S. Vallance and P. Calder. Context in 3D planar navigation. In *In Proceedings User Interface Conference, 2001. Second Australasian, AUIC 2001*, pages 93–99. IEEE Computer Society, 2001.
- [212] G. van den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann, 2004.
- [213] R. D. Vatavu, L. Anthony, and J. O. Wobbrock. Gestures as point clouds: a \$ P recognizer for user interface prototypes. In *Proceedings of the 14th ACM international conference on Multimodal interaction, ICMI '12*, pages 2875–2884, 2012.
- [214] S. Voelker, K. Nakajima, C. Thoresen, Y. Itoh, K. I. Øvergård, and J. Borchers. PUCs: detecting transparent, passive untouched capacitive widgets on unmodified multi-touch displays. In *UIST '13 Adjunct: Proceedings of the adjunct publication of the 26th annual ACM symposium on User interface software and technology*. ACM, Oct. 2013.
- [215] F. Wang, X. Cao, X. Ren, and P. Irani. Detecting and leveraging finger orientation for interaction with direct-touch surfaces. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology, UIST '09*, pages 23–32. ACM, 2009.

- [216] F. Wang and X. Ren. Empirical evaluation for finger input properties in multi-touch interaction. *CHI '09*, pages 1063–1072. ACM, Apr. 2009.
- [217] A. H. Watt and F. Policarpo. *3D Games*, volume 1 of *Real-time rendering and Software Technology*. Addison-Wesley, 2001.
- [218] A. H. Watt and F. Policarpo. *3D Games*, volume 2 of *Animation and Advanced Real-Time Rendering*. Addison-Wesley, 2003.
- [219] M. Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, 1991.
- [220] P. Wellner. The digitaldesk calculator: Tangible manipulation on a desk top display. In *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology*, UIST '91, pages 27–33. ACM, 1991.
- [221] A. Williams. *C++ Concurrency in Action: Practical Multithreading*. Manning Publications, first edition, Feb. 2012.
- [222] B. Williamson, C. Wingrave, and J. Iavoli. Realnav: Exploring natural user interfaces for locomotion in video games. In *IEEE Symposium on 3D User Interfaces 2010*, 3DUI '10, Jan. 2010.
- [223] A. Wilson, S. Izadi, O. Hilliges, A. Garcia-Mendoza, and D. Kirk. Bringing physics to the surface. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '09, pages 67–76. ACM, 2008.
- [224] A. M. Wing. Timing and co-ordination of repetitive bimanual movements. *The Quarterly Journal of Experimental Psychology*, 34(3):339–348, 1982.
- [225] J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: a \$1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, New York, New York, USA, Oct. 2007. ACM.
- [226] J. A. Wolfeld. *Real time control of a robot tactile sensor*. PhD thesis, University of Pennsylvania, 1981.
- [227] M. Wolter, B. Hentschel, I. Tedjo-Palczynski, and T. Kuhlen. A direct manipulation interface for time navigation in scientific visualizations. In *Proceedings of the 2009 IEEE Symposium on 3D User Interfaces*, 3DUI '09, pages 11–18.

- [228] A. Wu, D. Reilly, A. Tang, and A. Mazalek. Tangible Navigation and Object Manipulation in Virtual Environments. pages 37–44, 2011.
- [229] K. P. Yee. Two-handed interaction on a tablet display. *CHI'04 Extended Abstracts on Human Factors in Computing Systems*, 2004.
- [230] L. Yu, K. Efstathiou, P. Isenberg, and T. Isenberg. Efficient Structure-Aware Selection Techniques for 3D Point Cloud Visualizations with 2DOF Input. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2245–2254, 2012.
- [231] L. Yu, P. Svetachov, P. Isenberg, M. H. Everts, and T. Isenberg. FI3D: Direct-Touch Interaction for the Exploration of 3D Scientific Visualization Spaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1613–1622, 2010.
- [232] P. Zarchman and H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. AIAA, third edition, 2009.
- [233] M. Zechner and R. Green. *Beginning Android 4 Game Development*. Apress, Berkeley, CA, 2011.
- [234] S. Zhai. *Human Performance in Six Degree of Freedom Input Control*. PhD thesis, University of Toronto, 1995.
- [235] U. Zölzer. *Digital Audio Signal Processing*. John Wiley & Sons, Chichester, UK, July 2008.

## APPENDIX A

This appendix explains how to get additional information about the source code that was not included in this dissertation. Note that all the listing and algorithms needed to re-implement the solutions are provided in this dissertation.

3DNav prototype contains more than 30,000 lines of code. In addition, many third-party libraries, data files, configuration files, and many other additional data for 3DNav. Also, there are additional prototypes created for this dissertation, that includes source code and additional files. With this in mind, it is impossible to add all this information in this dissertation. There are a few ways to obtain the source code for the prototype.

- For the Yield essential source code, the reader may retrieve it from the Florida International University Library, as it has been uploaded as additional files to this dissertation. Go to <http://digitalcommons.fiu.edu/etd/>.
- It is possible to go to <http://www.FranciscoRaulOrtega.com>, this author's website. In particular, the page Projects should include information on how to download the source code. The source code may not be available until the end of 2015.
- Some additional code, and newer iterations of the code, may be available at <https://github.com/iblues76>. The source code may not be available until the end of 2015.
- If the source code you are looking for, it is not available, you can write an email to [forte007@fiu.edu](mailto:forte007@fiu.edu) or [FranciscoRaulOrtega@gmail.com](mailto:FranciscoRaulOrtega@gmail.com).



## **APPENDIX B**

This appendix includes sample handouts given to the subjects during the experiment. This appendix include the entry and exit surveys including additional questions that were only asked to a subset of subjects. It also includes the handout with the search objects.



4. If you play video games rarely or don't play video games, can you tell us why? Is it cost, low interest, lack of time, lack of skills or another set of reasons? :

---

---

5. How long have you been playing video games? Please circle one option:

**6 Months** **1 Year**    **2-4 years**    **4-6 years**    **5-10 years**    **10 or more years**

6. How often (approximately) do you currently play video games? Please circle one.

Never	Rarely	Daily
Weekly		
Once a month	Once every 3 months	Once in 6 months
Once a Year		

7. How would you described your skill level at playing video games with a scale of 1 – 5, with 5 the being the most skilled and 1 the least skilled?

5: Extremely well skilled	4: Very good	3: Good
2: Not very skilled	1: Not skilled at all.	

8. What gaming systems do you own or have you owned in the past? Please list them and specify if you still own them. Also, include if there are any systems you would like to own in the next year.

---

---

---

---

9. Please list your favorite video games. List at least a couple, if possible, and tell us why:

---

---

---

10. Please tell us what other devices besides multi-touch or gamepad you have used to play games? Have you used the Nintendo Wii Mote, PlayStation move? You can describe any device that you have used to play games in this question:

---

---

---

---

11. Have you heard about the Oculus Rift (experimenter will show you one) or similar devices? Can you tell us what you think about those devices and playing video games with them, if you have an opinion? Have you ever use them?

---

---

---

Thank you for participating. You can use the rest of this page to write any comments before starting the experiment about the questions asked.

12. Please feel free to write anything about video games below.

Additional questions (for selected subjects)

13 ) How often do you use gamepad to play video games (currently) ? Please circle one.

Never	Rarely	Daily
Weekly		
Once a month	Once every 3 months	Once in 6 months
Once a Year		

14. If you don't use it as often now, describe how often you used to use it before?  
Please circle one.

Never	Rarely	Daily
Weekly		
Once a month	Once every 3 months	Once in 6 months
Once a Year		

14b) Described when : \_\_\_\_\_

15. Rate how easy you find the game pad (very easy = 10, very hard = 1)

1    2    3    4    5    6    7    8    9    10

16. Do you have a preference for any type of game pad (e.g., XBOX ONE, Logitech, Playstation)? List them in order of preference, if you have more than one.

17. Please describe what you think of game pads?

## Natural User Interface Questionnaire Exit-Questionnaire

Please answer the following question. There is no time limit and you can stop at any time.

Gender: (Circle one) Female / Male

Experiment ID: \_\_\_\_\_

Date: \_\_\_\_\_

Age: \_\_\_\_\_

Major: \_\_\_\_\_

Major Completed Date: \_\_\_\_\_

Other Degrees: \_\_\_\_\_

Please circle your answer when appropriate or write free text in open questions.

1. On scale of 1 to 10, please rank how much easier you found the multi-touch display compared to the GamePad for 3D navigation. The higher you rank (10), the easier you found the multi-touch display versus the GamePad.

1      2      3      4      5      6      7      8      9      10

2. On scale of 1 to 10, please rank how easy you found the GamePad device compared to the multi-touch display. The higher you rank (10), the easier you found the GamePad device versus the multi-touch display.

1      2      3      4      5      6      7      8      9      10

3. On scale of 1 to 10, please rank how intuitive you found the multi-touch display. The higher you rank (10), the more intuitive you found it.

1      2      3      4      5      6      7      8      9      10

4. For the task given during the experiment: how do you rank the interaction to perform the search with the multi-touch display? The higher you rank (10), the better you found the experience with the device to perform for the assigned task during the experiment.

1      2      3      4      5      6      7      8      9      10

5. For the task given during the experiment: how do you rank the interaction to perform the search with the GamePad? The higher you rank (10), the better you found the experience with the device to perform for the assigned task during the experiment.

1      2      3      4      5      6      7      8      9      10

6. Given the time you took with multi-touch display, rank how likely you are to use this device for daily day use if you had access to it. The higher you rank, the more you expect to use if it was available to you.

1      2      3      4      5      6      7      8      9      10

7. Given the time you took with the GamePad device, rank how likely you are to use this device for daily day use if you had access to it. The higher you rank, the more you expect to use if it was available to you.

1      2      3      4      5      6      7      8      9      10

8. Please rank how did the multi-touch display compared to the GamePad device for rotating the camera. The highest you rank (10), the better you found the multi-touch display for rotations.

1      2      3      4      5      6      7      8      9      10

9. Please rank how the GamePad device compare to the multi-touch display for rotating the camera. The higher you rank (10), the better you found the GamePad device for rotations.

1      2      3      4      5      6      7      8      9      10

10. Please rank how the multi-touch display compared to the GamePad device for translation (up, down, left, right, forward, back). The higher you rank (10), the better you found the multi-touch display for translation movements.

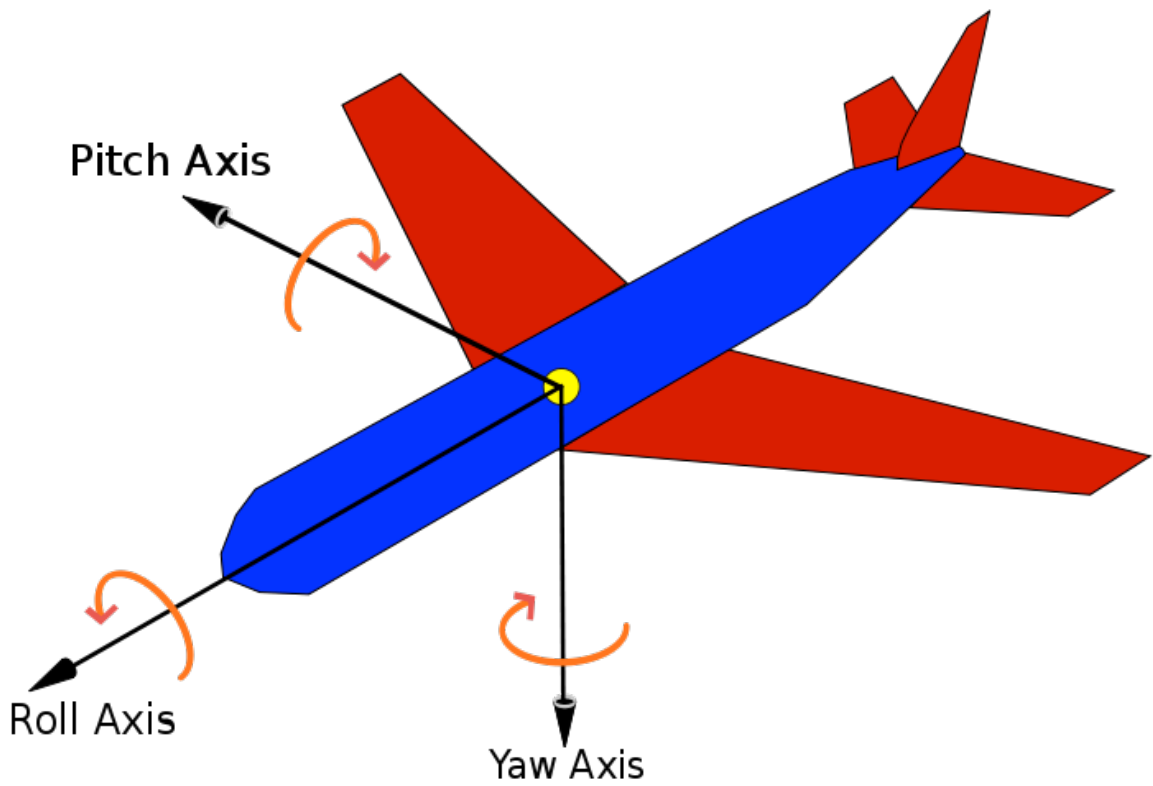
1      2      3      4      5      6      7      8      9      10

11. Please rank how the GamePad compared to the multi-touch display for translation (up, down, left, right, forward, back). The higher you rank (10), the better you found GamePad for translation movements.

1      2      3      4      5      6      7      8      9      10

12. Which device do you prefer: GamePad or multi-touch or No Difference (Both)? -- (please circle one)

13. Which device did you find better when asked to typed? GamePad or multi-touch or No Difference (Both) -- (please circle one)





14. Please select which Rotation or Translation you found better for the experiment you tested. Please mark with X for each of the categories. Use previous figure for reference.

	GamePad Device	Multi-Touch Display
Rotation: Yaw		
Rotation: Roll		
Rotation: Pitch		
Up – Down		
Left – Right		
Forward – Back		

15. Please tell us what did you thought about the experiment.

16. Tells us why you prefer one device over the other for the experiment.

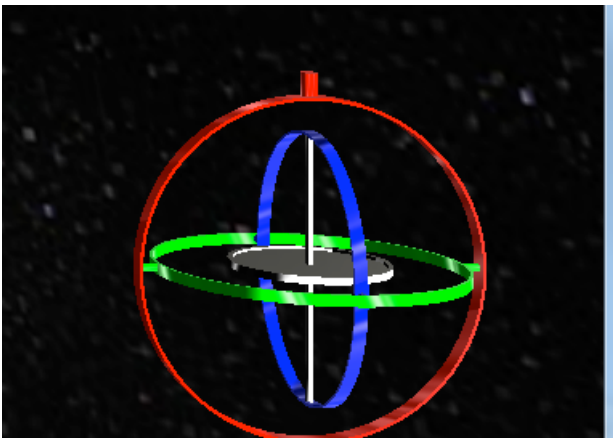
17. Tell us why do you prefer one device over the other when you have to type, as the experimented demanded.

18. Please tell us your overall opinion about the design of the multi-touch display

19. Please tell us your overall opinion about the design of the GamePad device.

20. Please tell us how we did in the experiment. Is there anything in the experiment that can be done better next time?

21. What do you think about the sphere that gave you a sense of rotation in space?



Thank you for participating.

22. You can use the rest of this page to write any comments before starting the experiment about the questions asked. Please feel free to write anything about video games below.

Additional questions for Hold-and-Roll (selected subjects only)

23) Please rate the Hold-and-Roll gesture you used during the experiment (10 = very useful, 1= not useful at all)

1      2      3      4      5      6      7      8      9      10

24) Would you like to see this gesture in new games? Please explain

25) Would you like to see this gesture in new applications? Please explain

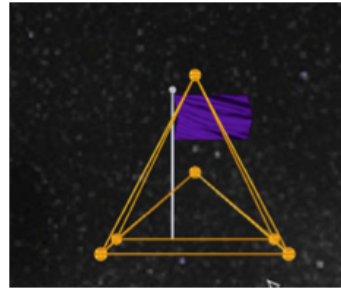
26) What is your opinion about Hold-and-Roll?

27) Please describe what benefits the multi-touch interaction gave you during this experiment. How about for your daily use?

28) Please describe what benefits the GamePad gave you during the interaction. How about for your daily use?



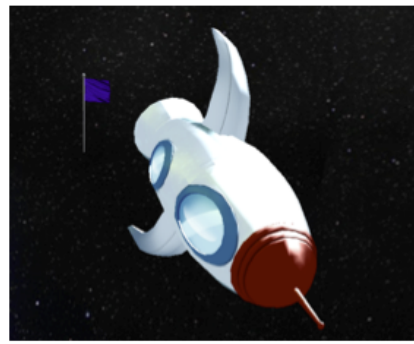
**CREATURE**



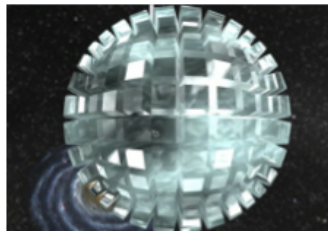
**Tetrahedra**



**Satellite**



**Ship**



You will find one of the objects in a purple nebula


**TARGET WILL HAVE ONE FLAG NEXT TO THEM, DEPENDING ON THE POINT OF VIEW**

Figure B.1: Handout with Search Objects

## **APPENDIX C**

The following appendix provides memorandums from the Office of Research Integrity, sent to the principal investigator, Dr. Armando Barreto, and this author (Francisco R. Ortega). For more information, use the IRB protocol approval number (IRB-13-0212) or Topaz reference number (101085). This appendix includes the first memorandum issued on June 7, 2013 and the second memorandum issued on April 29, 2014.

## MEMORANDUM

**To:** Dr. Armando Barreto  
**CC:** File  
**From:** Maria Melendez-Vargas, MIBA, IRB Coordinator   
**Date:** June 7, 2013  
**Protocol Title:** "3D Data Navigation Via Multi-Touch Display"

---

The Health Sciences Institutional Review Board of Florida International University has approved your study for the use of human subjects via the **Expedited Review** process. Your study was found to be in compliance with this institution's Federal Wide Assurance (00000060).

**IRB Protocol Approval #:** IRB-13-0212      **IRB Approval Date:** 05/31/13  
**TOPAZ Reference #:** 101085      **IRB Expiration Date:** 05/31/14


As a requirement of IRB Approval you are required to:

- 1) Submit an Event Form and provide immediate written notification to the IRB of:
  - Any additions or changes in the procedures involving human subjects.
  - Every serious or unusual or unanticipated adverse event as well as problems with the rights or welfare of the human subjects.
- 2) Utilize copies of the date stamped consent document(s) for the recruitment of subjects.
- 3) **Receive annual review and re-approval of your study prior to your expiration date.**  
Projects should be submitted for renewal at least 30 days in advance of the expiration date.
- 4) Submit a Project Completion Report Form when the study is finished or discontinued.

**Special Conditions:** N/A

For further information, you may visit the IRB website at <http://research.fiu.edu/irb>.

## MEMORANDUM

**To:** Dr. Armando Barreto  
**CC:** File  
**From:** Maria Melendez-Vargas, MIBA, IRB Coordinator   
**Date:** April 29, 2014  
**Protocol Title:** "3D Data Navigation Via Multi-Touch Display"

---

The Health Sciences Institutional Review Board of Florida International University has re-approved your study for the use of human subjects via the **Expedited Review** process. Your study was found to be in compliance with this institution's Federal Wide Assurance (0000060).

**IRB Protocol Approval #:** IRB-13-0212      **IRB Approval Date:** 04/22/14  
**TOPAZ Reference #:** 101085      **IRB Expiration Date:** 04/22/15

As a requirement of IRB Approval you are required to:

- 1) Submit an IRB Amendment Form for all proposed additions or changes in the procedures involving human subjects. All additions and changes must be reviewed and approved by the IRB prior to implementation.
- 2) Promptly submit an IRB Event Report Form for every serious or unusual or unanticipated adverse event, problems with the rights or welfare of the human subjects, and/or deviations from the approved protocol.
- 3) Utilize copies of the date stamped consent document(s) for obtaining consent from subjects (unless waived by the IRB). Signed consent documents must be retained for at least three years after the completion of the study.
- 4) **Receive annual review and re-approval of your study prior to your IRB expiration date.** Submit the IRB Renewal Form at least 30 days in advance of the study's expiration date.
- 5) Submit an IRB Project Completion Report Form when the study is finished or discontinued.

**Special Conditions:** N/A

For further information, you may visit the IRB website at <http://research.fiu.edu/irb>.

## **APPENDIX D**

This appendix has the legends for the Xbox 360 controller, as shown in Figures D.1 and D.2. This is the control used during the experiment. Figure D.1, shows both thumb-sticks (left and right), the digital pad (dPad), and the four buttons (A,B,X,Y). Figure D.2 shows the left and right shoulder back (LB,RB) and the analog left and right triggers.





Figure D.1: Xbox 360 Legend

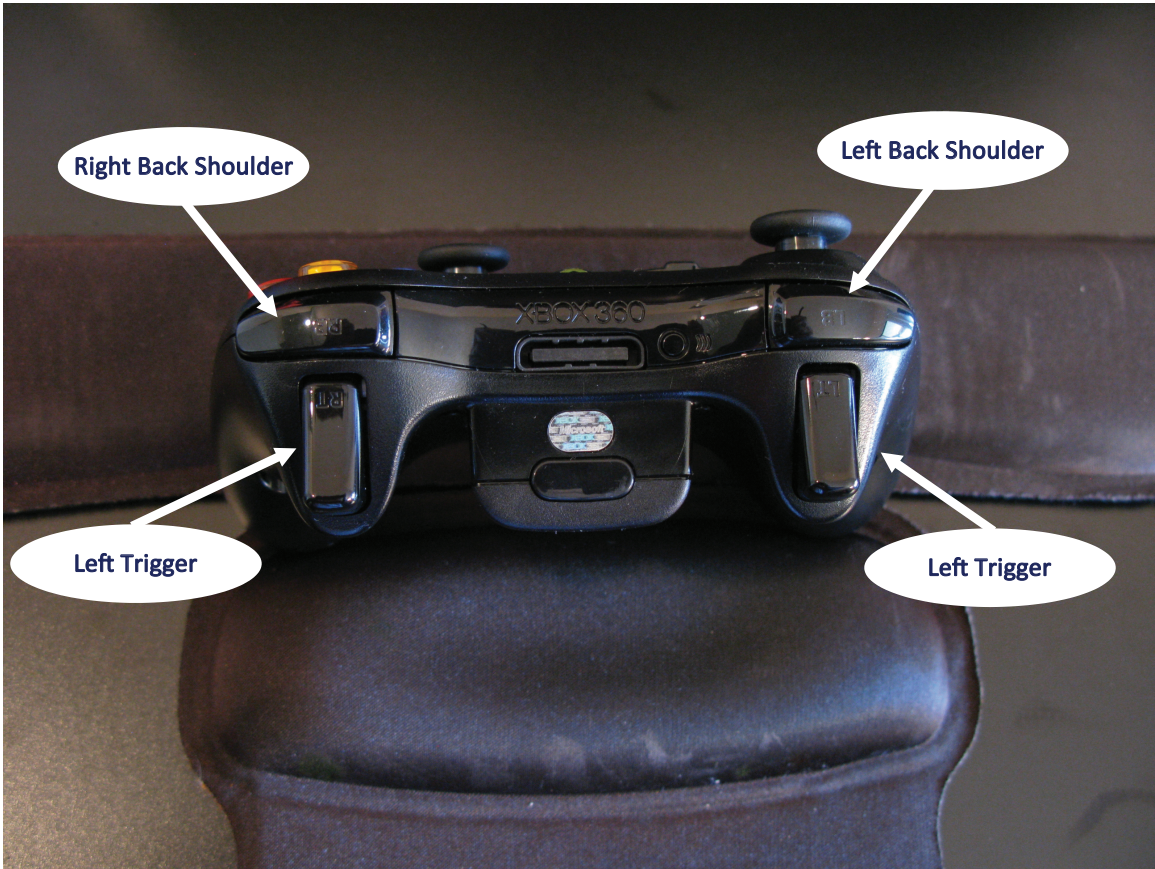


Figure D.2: Xbox 360 Legend

## APPENDIX E

### IRML for PeNTa The Syntax and Static Semantics

A HLPN is a tuple:

$$HLPN = (N, Spec, ins)$$

where:

- $N = (P, T, F)$  is a net structure:
  - $P$  a finite set of nodes called places;
  - $T$  a finite set of nodes, called transitions disjoint from  $P$ ;
  - $F$  a finite set of directed flow relations called arcs, where  $F \subseteq (P \times T) \cup (T \times P)$ .
- $Spec = (S, OP, Eq)$  is the underlying specifications:
  - $S$  a set of sorts;
  - $OP$  a set of sorted operations;
  - $Eq$   $S$ -equations that define the meanings and properties of operations in  $OP$ .
  - Note: Tokens of a HLPN are ground terms of the signature  $(S, OP)$ ;
- $ins = (\varphi, L, R, M_0)$  is the net inscription:
  - $\varphi$  the data definition associates each place  $p \in P$  with sorts;
  - $L$  labeling of net, an abbreviation is defined as  $L(x, y)$  iff  $(x, y) \in F$  and  $\tilde{A}\tilde{Y}$  otherwise;
  - $R = (Pre, Post)$  well defined constraining mapping, associates each transition  $t \in T$  with constraint algebraic formulas and predefined functions;  $Pre$  and  $Post$  are the pre and post mappings of marking;

- $M_0$  sort-respecting initial marking that assigns a multi-set of tokens to each place  $p \in P$ .

### Dynamic Semantics

- Marking: Markings of a HLPN are mappings  $M : P \rightarrow Tokens$ ;
- Enabling: Given a marking  $M$ , a transition  $t \in T$  is enabled at marking  $M$  iff  $Pre(t) \leq M$
- Concurrent Enabling: Given a marking  $M$ ,  $\alpha_t$  is an assignment for variables of  $t$  that satisfies its transition condition and  $A_t$  denotes the set of all assignments. Define the set of all transition modes to be  $TM = \{(t, \alpha_t) \mid t \in T, \alpha_t \in A_t\}$  iff  $Pre(TM) \leq M$ .
- Transition Rule: Given a marking  $M$ , if  $t \in T$  is enabled in mode  $\alpha_t$ , firing  $t$  by a step may occur in a new marking  $M' = M - Pre(t_{\alpha_t}) + Post(t_{\alpha_t})$ ; A step is denoted by  $M[t > M'$ .
- Behavior of a HLPN: an execution sequence  $M_0[t_0 > M_1[t_1 > \dots$  is either finite when the last marking is terminal (no more enabled transition) or infinite. The behavior of a HLPN model is the set of all execution sequences starting from  $M_0$ .

## **APPENDIX F**

This appendix contains the list of algorithms and the list of source code. For the list of tables and figures, please refer to the beginning of the document.

## LIST OF ALGORITHMS

3.1	GestureDetection . . . . .	70
3.2	TOUCHMOVE . . . . .	73
3.3	GestureDetection . . . . .	75
3.4	Rotation Algorithm for a Gyroscope . . . . .	81
3.5	Yield: Fetch Points . . . . .	102
3.6	Yield: Fetch Gestures . . . . .	104
3.7	Yield: Reset Active Gesture . . . . .	105
3.8	Yield: Process Candidate . . . . .	108
3.9	Yield: Pre-Process Action . . . . .	109
3.10	Yield: Pre-Process Action (alt) . . . . .	109

## SOURCE CODE LISTING

3.1	CloudFeatureMatch (onDown)	76
3.2	CloudFeatureMatch (onUp)	76
3.3	CloudFeatureMatch (onMove)	76
3.4	CloudFeatureMatch (Slit)	77
3.5	CloudFeatureMatch (BreakTop)	77
3.6	CloudFeatureMatch (Features)	78
3.7	WiiMote (Initialize)	81
3.8	Gyroscope (wiiMote)	81
3.9	Yield (gc)	98
3.10	Yield (processActionGesture)	110
3.11	Yield (Data Structures)	111
3.12	Yield (Gesture.ProcessGesture)	111
3.13	Navigation (Update)	113
3.14	Navigation (Push Translation)	114
3.15	Navigation (Push Rotations)	115
3.16	Navigation (Push Translation)	116
4.1	Observer Pattern (Listener.h)	120
4.2	Observer Pattern (Registry.h)	120
4.3	Observer Pattern (Registry.cpp)	121
4.4	InputPad (Interface)	121
4.5	3DMouse (Global and Initial Checks)	125
4.6	3DMouse (Partial Initialize)	125
4.7	3DMouse (Process Input Error Checking)	126
4.8	3DMouse (Process Input)	127
4.9	YEI 3Space Sensor Wireless (Wrapper)	129

4.10	YEI 3Space Sensor (Wrapper)	129
4.11	Kinect (WINAPI Messages)	133
4.12	Skelton Tracking (Left/Right)	134
4.13	Keyboard (Navigation)	136
4.14	XBox360 Controller (h)	138
4.15	XBox360 Controller (cpp)	139
4.16	Game Navigation Controller(h)	140
4.17	Game Navigation Controller(cpp)	141
4.18	XBox360 Controller (Smooth Input)	141
4.19	WINAPI (Multi-Touch init)	146
4.20	WINAPI (Multi-Touch WinMain)	146
4.21	WINAPI (Events)	147
4.22	WINAPI (Multi-Touch Touch Events)	148
4.23	WINAPI (Multi-Touch down,move,up)	149
4.24	GWC (GWTouch.h)	150
4.25	GWC (GWTouch.cpp)	151
4.26	Scene Nodes (Partial XML file)	155
4.27	OGRE (Advanced Framework (h))	156
4.28	OGRE (Menu State (h))	157
4.29	OGRE (Game State (h))	157
4.30	OGRE (App State Interface)	158
4.31	OGRE (InitOgre)	159
4.32	OGRE (Check Input)	160
4.33	OGRE (Collision)	160
4.34	Experiment Controller	163
4.35	Experiment Device	163



4.36	Experiment Task	164
4.37	Experiment Search Object	165
5.1	3DNAV game.cfg (Experiment Setup)	174

## APPENDIX G

This appendix contains miscellaneous notes about this dissertation and permission letters to reprint figures.

- Web links were provided as footnotes. In most cases, the typical format was “See <http://fiu.edu>.”. Note the last period of the web address is not part of the URL. In a few instances, because of the web address’s length, the “See” keyword and/or the last period were omitted.
- For footnotes with “See Wikipedia: topic”, please search using <http://www.wikipedia.com> the “topic”.
- All pictures are owned by this author, except when external images were used. When external images were used, permission was obtained in writing.



Francisco Ortega <franciscoraulortega@gmail.com>

---

**Re: Fw: Permission**

ikimball@mmm.com <ikimball@mmm.com>  
To: franciscoraulortega@gmail.com

Wed, Sep 24, 2014 at 9:55 AM

Hi Francisco,

You have our permission to use these images for your dissertation These images are publicly variable on our website and intended for public consumption.

Thanks,

Ian

---



**Ian Kimball** | Market Development Manager  
Display Materials & Systems Division  
501 Griffin Brook Park Drive | Methuen, MA 01844  
Office: 978 659 9377 | Mobile: 978 404 0254  
[ikimball@mmm.com](mailto:ikimball@mmm.com) | [www.3M.com/multitouch](http://www.3M.com/multitouch) | [touchtopics.com](http://touchtopics.com)

---

From: Francisco Ortega <franciscoraulortega@gmail.com>  
To: us-ts-techsupport <us-ts-techsupport@mmm.com>  
Date: 09/22/2014 11:54 AM  
Subject: Permission

---

Hello,

I would like to have permission to print 7 images from Tech Brief-1013 (2013)  
[http://solutions.3m.com/3MContentRetrievalAPI/BlobServlet?lmd=1332776733000&locale=en\\_US&assetType=MMM\\_Image&assetId=1319224169961&blobAttribute=ImageFile](http://solutions.3m.com/3MContentRetrievalAPI/BlobServlet?lmd=1332776733000&locale=en_US&assetType=MMM_Image&assetId=1319224169961&blobAttribute=ImageFile)

Images 1,2,3,4,5,6,7, to be added to my dissertation, since I used a 3M Multi-Touch monitor.

Can you provide with the email for permissions or provide such permission via email.

Thanks,  
Francisco R. Ortega  
Ph.D. Candidate in Computer Science  
Mcknight DYF Fellow, GAANN Fellow



Francisco Ortega <franciscoraulortega@gmail.com>

---

## Press Images request 3DConnexion : 3D Mouse Space Navigator

---

Mike Kaput <mike@pr2020.com>

Thu, Jul 31, 2014 at 10:29 AM

To: Francisco Ortega <franciscoraulortega@gmail.com>

Hi Francisco,

Thank you so much for reaching out. Glad to hear you're using a 3D mouse (and writing about them!).

I've attached a ZIP file with the images you requested. Also, [here's a link to the image of the SpaceNavigator](#). We're going through an update / improvement process right now, so this is the only photo we've got at the moment—but I may be able to send some more over next week.

Please feel free to use these images in your two publications. And be sure to send the final publications to us when you're done, as we'd love to read them :)

**Note:** Could you please use the following copyright with the images:

**© 2014 3Dconnexion. All rights reserved. 3Dconnexion, the 3Dconnexion logo, and other 3Dconnexion marks are owned by 3Dconnexion and may be registered.**

I'm working right now to confirm the 1,000,000 sold number. I'll update you when I have more information. **If you don't hear back from me in time on that number**, please just use the 1,000,000 sold number in the press release.

Thanks, Francisco! Best of luck with your writing! Just let me know if you need anything else.

Best,

Mike Kaput

[Quoted text hidden]

--

Mike Kaput

Consultant | [PR 20/20](#)


[@mikekaput](#)

[216.812.3960](#)

---

 **3Dconnexion\_Images-for-Francisco.zip**  
1346K

DIRECTPATH
GET PERMISSION
PRODUCTS & SOLUTIONS
EDUCATION
ABOUT US



1  
PAYMENT
2  
REVIEW
3  
**CONFIRMATION**

### Step 3: Order Confirmation

**Thank you for your order!** A confirmation for your order will be sent to your account email address. If you have questions about your order, you can call us at +1.855.239.3415 Toll Free, M-F between 3:00 AM and 6:00 PM (Eastern), or write to us at [info@copyright.com](mailto:info@copyright.com). This is not an invoice.

**Confirmation Number: 11270276**  
**Order Date: 10/10/2014**

If you paid by credit card, your order will be finalized and your card will be charged within 24 hours. If you choose to be invoiced, you can change or cancel your order until the invoice is generated.

#### Payment Information

Francisco Ortega  
franciscoraulortega@gmail.com  
+1 (305)3056391  
Payment Method: n/a

---

#### Order Details

#### 3D graphics for game programming

<p><b>Order detail ID:</b> 65874912  <b>Order License Id:</b> 3485530797036  <b>ISBN:</b> 978-1-4398-2737-6  <b>Publication Type:</b> Book  <b>Publisher:</b> Chapman and Hall/CRC  <b>Author/Editor:</b> Han, JungHyun</p>	<p><b>Permission Status:</b> <span style="color: green;">✔</span> <b>Granted</b>  <b>Permission type:</b> Republish or display content  <b>Type of use:</b> Republish in a thesis/dissertation  <a href="#">View details</a></p>
---	--


**Note:** This item will be invoiced or charged separately through CCC's **RightsLink** service. [More info](#) **\$ 0.00**

**Total order items: 1**


**This is not an invoice.**

**Order Total: 0.00 USD**

DIRECTPATH    GET PERMISSION    PRODUCTS & SOLUTIONS    EDUCATION    ABOUT US



[Back to view orders](#)

 [Print this page](#)  
[Print terms & conditions](#)  
[Print citation information](#)  
[\(What's this?\)](#)

**Confirmation Number: 11275519**  
**Order Date: 11/04/2014**

**Customer Information**

**Customer:** Francisco Ortega  
**Account Number:** 3000730667  
**Organization:** Francisco Ortega  
**Email:** franciscoraulortega@gmail.com  
**Phone:** +1 (305)3056391


Search order details by:

**This is not an invoice**

**Order Details**

**Human-computer interaction : an empirical research perspective**

Billing Status: <b>N/A</b>
-------------------------------

<b>Order detail ID:</b> 65922990	<b>Permission Status:</b>  <b>Granted</b>
<b>ISBN:</b> 978-0-12-405865-1	<b>Permission type:</b> Republish or display content
<b>Publication Type:</b> Book	<b>Type of use:</b> Thesis/Dissertation
<b>Author/Editor:</b> MacKenzie, I. Scott	<b>Order License Id:</b> 3502130106832

[View details](#)

**Note:** This item was invoiced separately through our **RightsLink service**. [More info](#) **\$ 0.00**

<b>Total order items: 1</b>	<b>Order Total: \$0.00</b>
-----------------------------	----------------------------

## VITA

### FRANCISCO RAUL ORTEGA

- 2005 - 2007                      B.S., Computer Science  
Florida International University  
Miami, FL
- 2007 - 2008                      M.S., Computer Science  
Florida International University  
Miami, FL
- 2009 - 2014                      Ph.D., Computer Science  
Florida International University  
Miami, FL

### PUBLICATIONS AND PRESENTATIONS

Francisco R. Ortega, Fatemeh Abyarjoo, Armando Barreto, Naphtali Rische, Malek Adjouadi. *3D User Input Interfaces*. CRC Press. 2015.

F. Ortega, A. Barreto, N. Rische, M. Adjouadi, and P. Ren, “Emperical Analisys of 3D Navigation using Multi-Touch with 6DOF”. Submitted, ACM Transactions on Computer- Human Interaction (TOCHI).

Francisco R Ortega, Su Liu, Frank Hernandez, Armando Barreto, Naphtali Rische, and Malek Adjouadi, “PeNTa: Formal Modeling for Multi-Touch Systems Using Petri Net”, In Human-Computer Interaction. Theories, Methods, and Tools, pp. 361–372. Springer International Publishing, January 2014.

Hernandez H., Ortega F., “Eberos GML2D: A Graphical Domain-Specific Lan- guage for Modeling 2D Video Games”, The 10th Workshop on Domain-Specific Model- ing proceedings 2010.

P. Ren, A. Barreto, J. Huang, Y. Gao, F. R. Ortega, and M. Adjouadi, “Offline and online stress detection through processing pupil diameter signal” Annals of Biomedical Engineering, Vol. 42, No. 1, January 2014 ( 2013) pp. 162-176.

Ortega F., Barreto A., Rische N., Adjouadi M., and Abyarjoo F., “Multi-Touch Gesture Recognition using Feature Extraction”, Proceedings of CISSE 2012: The International Joint Conferences on Computer, Information and Systems Sciences and Engineering, December 7–9, 2012, Bridgeport, CT. Springer 2014. LNEE 152, pp. (Note: Printed edition contains typo in first author’s last name. It appears as Ortego).

Ortega F., Barreto A., Rische N., and Adjouadi M., “Interaction with 3D Environments using Multi-Touch Screens”, Proceedings of CISSE 2011: The International Joint Conferences on Computer, Information and Systems Sciences and Engineering, December 3, 2011, Bridgeport, CT. Springer 2013, LNCS 7152, pp. 381–392.

Ortega, F., Barreto A., Rische N. and Adjouadi M., Abyarjoo F, “GyroTouch: Complementing the Multi-Touch Display”, ACM Richard Tapia Celebration of Diversity in Computing, 2014. Seattle, WA.

Ortega, F., Hernandez, F., Barreto A., Rische N., Adjouadi M., Liu S., “Exploring Modeling Language for Multi-Touch Systems using PetriNet”, Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces (ITS '13), ACM, New York, NY, USA, 361–364.

Ortega F., Barreto A., Rische N. “Augmenting Multi-Touch with Commodity Device”, In Proceedings of the 1st symposium on Spatial user interaction (SUI '13), ACM, New York, NY, USA, p. 95.

Ortega F., Barreto A., Rische N. and Adjouadi M., Abyarjoo F, “Poster: Real-Time Gesture Detection for Multi-Touch Devices”, IEEE 8th Symposium on 3D User Interfaces, 2013, pp 167-168.

Ortega F., Barreto A., Rische N. and Adjouadi M., “Towards 3D Data Environments using Multi-Touch Screens”, ACHI 2012 : The Fifth International Conference on Advances in Computer-Human Interactions. pp 118-121.

Ortega, F., Rische N, and Barreto A., “Multi-Touch Machine Framework – mtMachine”, Provisional Patent Application Filed. USPTO. Expected to submit patent July, 2015.

Ortega, F., PeNTa: Formal Modeling for Multi-Touch Systems Using Petri Net, HCI International 2014. Crete, Greece. June 2014.

Ortega, F., Exploring Modeling Language for Multi-Touch Systems using PetriNet. 2013 ACM international conference on Interactive tabletops and surfaces (ITS 2013). St. Andrew, Scotland. October, 2013.

Verhoef T., Lisetti C., Barreto A., Ortega F., Van der Zant T. and Cnossen F., “Bio-sensing for Emotional Characterization without Word Labels”, Human-Computer Interaction. Ambient, Ubiquitous and Intelligent Interaction, 13th International Conference, HCI International, LNCS 5612, pp. 693–702, 2009.

Wu Y., Hernandez F., Ortega F., Clarke PJ. and France R. , “Measuring the Effort for Creating and Using Domain-Specific Models”, The 10th Workshop on Domain-Specific Modeling proceedings 2010.