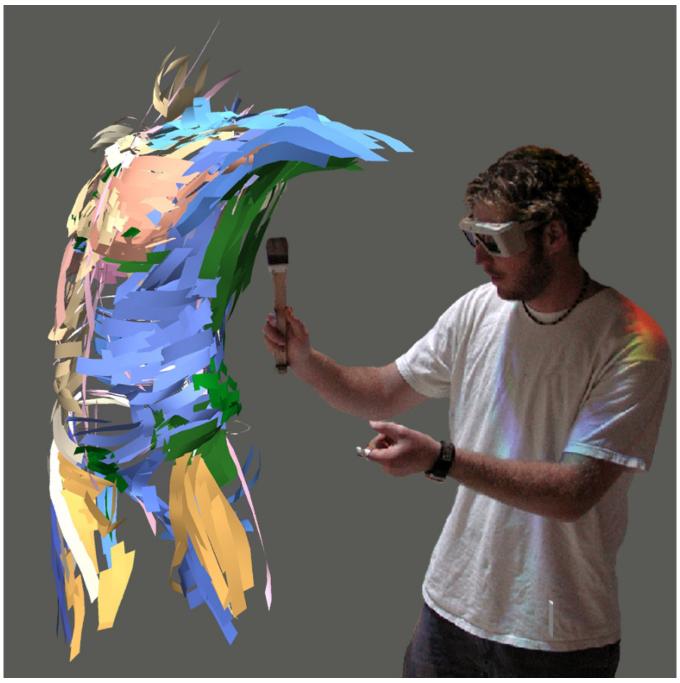
3D Spatial Interaction: Applications for Art, Design, and Science

A SIGGRAPH 2011 Course
Sunday, August, 7, 2011
2:00PM - 5:15 PM

Joseph J. LaViola Jr.
Dept. of EECS
University of Central Florida
jjl@eecs.ucf.edu

Daniel F. Keefe
Dept. of Computer Science and Engineering
University of Minnesota - Twin Cities
keefe@cs.umn.edu



Contents

1 Course Introduction **1**

1.1 Course Topics . . . 1

1.2 Course Schedule . . . 2

1.3 Speaker Biographies . . . 4

2 Common Tasks in 3D User Interfaces **6**

2.1 Navigation . . . 7

2.1.1 Gaze-Directed Steering . . . 7

2.1.2 Pointing . . . 8

2.1.3 Map-based Travel . . . 8

2.1.4 Grabbing the Air . . . 9

2.2 Selection . . . 9

2.2.1 The Virtual Hand . . . 10

2.2.2 Ray-Casting . . . 10

2.2.3 Occlusion Techniques . . . 11

2.2.4 Arm-Extension . . . 11

2.3 Manipulation . . . 12

2.3.1 HOMER . . . 12

2.3.2 Scaled-World Grab . . . 13

2.3.3 World-in-Miniature . . . 14

2.4 System Control . . . 15

2.4.1 Graphical Menus . . . 15

2.4.2 Voice Commands . . . 16

2.4.3 Gestures and Postures . . . 17

2.4.4 Tools . . . 18

3 Applications 1: Spatial Interfaces in Art and Design **20**

3.1 Background in 3DUI's for Art and Design . . . 20

3.1.1 Early Research in 3D User Interfaces for Art and Design . . . 20

3.1.2 Recent Research in 3D User Interfaces for Art and Design . . . 21

3.2 Example 3D User Interfaces for Art Applications . . . 22

3.3 Installation and Interactive Art . . . 23

3.4 Applications in Creative 3D Design . . . 24

4 Working with Video Game Motion Controllers **25**

4.1 Motion Controller Device Characteristics . . . 25

4.1.1 Nintendo Wii Remote . . . 25

4.1.2 Microsoft Kinect . . . 29

4.1.3 Playstation Move . . . 31

4.2 Getting Started and Software Tools . . . 34

4.2.1 Nintendo Wii Remote . . . 34

4.2.2 Microsoft Kinect . . . 35

4.2.3 Playstation Move . . . 36

5 Applications 2: Spatial Interfaces in Scientific Visualization **37**

5.1 3D Interactive Techniques for Scientific Visualization 37
5.2 Selection Revisited 37
5.3 Navigation Revisited 39
5.4 Does spatial computing for scientific applications require a $1,000,000 lab? 40

6 3D Gesture Recognition Techniques 41
6.1 Interpreting Wiimote Data 41
6.1.1 Integration 41
6.1.2 Recognition through Heuristics 42
6.2 Linear Classifier 45
6.3 AdaBoost Classifier 46
6.3.1 Weak Learner Formulation 46
6.3.2 AdaBoost Algorithm 47
6.4 Feature Set 48
6.5 Gesture Set 48
6.6 3D Gesture Data Collection 49
6.7 3D Gesture Recognition Experiments 50
6.7.1 User Dependent Recognition Results 51
6.7.2 User Independent Recognition Results 54
6.8 Accuracy Discussion 55
6.9 Using 3D Gestures in a Practical Setting 55
6.9.1 Usability Study 56
6.9.2 Results 57
6.9.3 Discussion 58

7 Future 2: Mix and Match Spatial Input 60
7.1 Spatial input meets multi-touch 60
7.1.1 Building the Minnesota Interactive Visualization Lab 3D Multi-touch Workbench . 61
7.1.2 Future Outlook 63
7.2 Tangible spatial input 63
7.3 Cognitive and perceptual issues 64

1 Course Introduction

3D interfaces use motion sensing, physical input, and spatial interaction techniques to effectively control highly dynamic virtual content [Bowman et al. 2004]. With the advent of the Nintendo Wii, Sony Move, and Microsoft Kinect, game developers and researchers are now faced with the challenge of creating compelling interface techniques and gameplay mechanics that make use of these technologies [Wingrave et al. 2010]. At the same time, it is becoming increasingly clear that emerging game technologies are not just going to change the way we play games, they are also going to change the way we make and view art, design new products, analyze scientific datasets, and more. Researchers in the fields of virtual and augmented reality as well as 3D user interfaces have been working on 3D interaction for nearly two decades, but today, thanks to emerging technologies, there is great potential for 3D spatial interfaces to radically change the way that we approach art, design, and science.

1.1 Course Topics

This course provides an introduction to the exciting and rapidly changing world of 3D spatial interfaces. Presenters will demystify the workings of modern day video game motion controllers and provide an overview of the techniques, strategies, and algorithms used in creating 3D interfaces for tasks such as 2D and 3D navigation, object selection and manipulation, and gesture-based application control. We will discuss the strengths and limitations of various motion controller sensing technologies found in today's peripherals and describe useful techniques for working with these devices, including gesture recognition and non-isomorphic control-to-display mappings. Current and future applications of these technologies to areas outside of games will be discussed through illustrative recent examples of 3D interfaces for art [Keefe et al. 2007], design [Keefe 2009], and science [Coffey et al. 2011] . Presenters will identify common themes and also critical differences across these applications and discuss how these insights might help shape the future of 3D spatial interaction. Attendees will receive valuable information on how to utilize existing 3D user interface techniques with emerging technologies, how to develop their own interface techniques, and how to learn from the successes and failures of spatial interfaces created for a variety of application domains.

1.2 Course Schedule

1. Welcome, Introduction, & Roadmap – 15 minutes - LaViola
   (a) Motivate the importance of the topic
   (b) Introduce presenters
   (c) Applications of 3D spatial interaction
2. Common Tasks in 3D User Interfaces – 30 minutes - LaViola
   (a) Selection and manipulation techniques
   (b) Travel techniques
   (c) System control techniques
3. Applications 1: Spatial Interfaces in Art and Design – 30 minutes - Keefe
   (a) 3D sketching and sculpting
   (b) Installation and experiential art
   (c) Design, from early concepts to precision engineering
4. Working with Video Game Motion Controllers – 30 minutes - LaViola
   (a) Device characteristics
      i. Nintendo Wii
      ii. Microsoft Kinect
      iii. Sony Move
   (b) Getting Started and Software tools
5. Applications 2: Spatial Interfaces in Scientific Visualization – 30 minutes - Keefe
   (a) Selection revisited
   (b) Navigation revisited
   (c) Does spatial computing in science require a $1,000,000 lab?
6. Future 1: 3D Gesture Recognition Techniques – 30 minutes - LaViola
   (a) 3D Gestures
   (b) Heuristic Recognition
   (c) Linear and AdaBoost classifiers
   (d) A 25 gesture open source dataset
   (e) Using 3D gestures in a practical setting
7. Future 2: Mix and Match Spatial Input – 30 minutes - Keefe

    (a) Spatial input meets multi-touch
    (b) Tangible spatial input
    (c) Cognitive and perceptual issues

8. Q&A – 10 minutes - LaViola and Keefe

1.3 Speaker Biographies

Joseph J. LaViola Jr.
SAIC Faculty Fellow and Assistant Professor, University of Central Florida

University of Central Florida
Dept. of EECS
4000 Central Florida Blvd.
Orlando, FL 32816

(407)882-2285 (voice)
(401)823-5419 (fax)
jjl@eecs.ucf.edu (email)

Joseph J. LaViola Jr. is an assistant professor in the School of Electrical Engineering and Computer Science and directs the Interactive Systems and User Experience Lab at the University of Central Florida. He is also an adjunct assistant research professor in the Computer Science Department at Brown University. His primary research interests include pen-based interactive computing, 3D spatial interfaces for video games, predictive motion tracking, multimodal interaction in virtual environments, and user interface evaluation. His work has appeared in journals such as ACM TOCHI, IEEE PAMI, Presence, and IEEE Computer Graphics & Applications, and he has presented research at conferences including ACM SIGGRAPH, ACM CHI, the ACM Symposium on Interactive 3D Graphics, IEEE Virtual Reality, and Eurographics Virtual Environments. He has also co-authored ”3D User Interfaces: Theory and Practice,” the first comprehensive book on 3D user interfaces. In 2009, he won an NSF Career Award to conduct research on mathematical sketching. Joseph received a Sc.M. in Computer Science in 2000, a Sc.M. in Applied Mathematics in 2001, and a Ph.D. in Computer Science in 2005 from Brown University.

Daniel F. Keefe
Assistant Professor, University of Minnesota - Twin Cities

University of Minnesota
Department of Computer Science and Engineering
4-192 Keller Hall
200 Union Street SE
Minneapolis, MN 55455

(612) 626-7508 (voice)
(612) 626-7508 (fax)
keefe@cs.umn.edu (email)

Dan Keefe is an Assistant Professor in the Department of Computer Science and Engineering at the University of Minnesota. His research centers on scientific data visualization (visualization of time-varying data, visualization at scale, perceptually optimized visualization) and interactive computer graphics (3D interfaces, haptics, pen and multi-touch input). Keefe received the NSF CAREER Award in 2011. He has received best paper and best panel awards at international conferences for his research and teaching. In addition to his research in computer science, Keefe is also an accomplished artist and has published and exhibited his artwork in national and international venues. Before joining the University of Minnesota, Keefe did post-doctoral work at Brown University jointly with the departments of Computer Science and Ecology and Evolutionary Biology and with the Rhode Island School of Design. He received the Ph.D. in 2007 from Brown Universitys Department of Computer Science, which nominated his dissertation for the ACM Dissertation Prize, and the B.S. in Computer Engineering summa cum laude from Tufts University in 1999.

2 Common Tasks in 3D User Interfaces

We first will take a brief tour of the techniques used for common tasks in 3D spatial interaction. What is 3D spatial interaction anyway? As starting point, we can say that a 3D user interface (3D spatial interaction) is a UI that involves human computer interaction where the user's tasks are carried out in a 3D spatial context with 3D input devices or 2D input devices with direct mappings to 3D. In other words, 3D UIs involve input devices and interaction techniques for effectively controlling highly dynamic 3D computer-generated content. Figure 1 shows example 3D UIs.

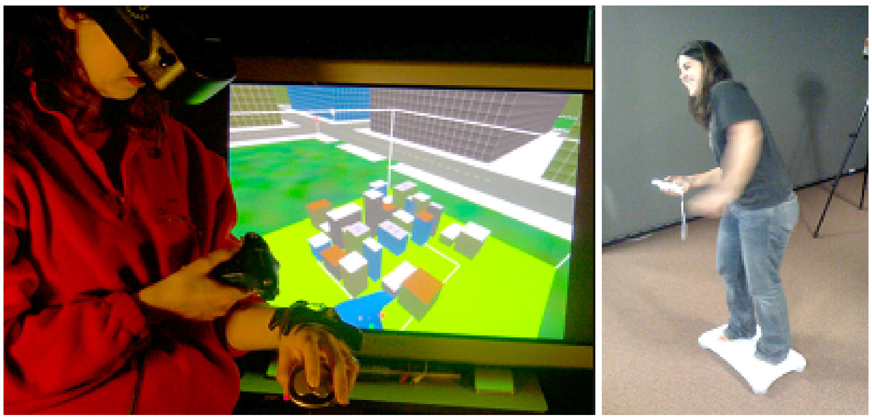


Figure 1: *Two examples of a 3D UI. The image on the left shows a user interacting with a world-in-miniature and the image on the right shows a user navigating using body-based controls.*

There are essentially four basic 3D interaction tasks that are found in most complex 3D applications [Bowman et al. 2004]. Actually, there is a fifth task called symbolic input, the ability to enter alphanumeric characters in a 3D environment, but we will not discuss it here. Obviously, there are other tasks which are specific to an application domain, but these basic building blocks can often be combined to let users perform more complex tasks. These tasks include navigation, selection, manipulation, and system control. **Navigation** is the most common VE task, and is consists of two components. *Travel* is the motor component of navigation, and just refers to physical movement from place to place. *Wayfinding* is the cognitive or decision-making component of navigation, and it asks the questions, ”where am I?”, ”where do I want to go?”, ”how do I get there?”, and so on. **Selection** is simply the specification of an object or a set of objects for some purpose. **Manipulation** refers to the specification of object properties (most often position and orientation, but also other attributes). Selection and manipulation are often used together, but selection may be a stand-alone task. For example, the user may select an object in order to apply a command such as ”delete” to that object. System control is the task of changing the system state or the mode of interaction. This is usually done with some type of command to the system (either explicit or implicit). Examples in 2D systems include menus and command-line interfaces. It is often the case that a system control technique is composed of the other three tasks (e.g. a menu command involves selection), but it's also useful to consider it separately since special techniques have been developed for it and it is quite common.

There are two contrasting themes that are common when thinking about 3D spatial interfaces; the real and the magical. The real theme or style tries to bring real world interaction into the 3D environment. Thus, the goal is to try to mimic physical world interactions in the virtual world. Examples include direct manipulation interfaces, such as swinging a golf club or baseball bat or using the hand to pick up virtual objects. The magical theme or style goes beyond the real world into the realm of fantasy and science fiction. Magical techniques are only limited by the imagination and examples include spell casting, flying, and moving virtual objects with levitation.

Two technical approaches used in the creation of both real and magical 3D spatial interaction techniques are referred to as isomorphism and non-isomorphism. Isomorphism refers to a one to one mapping between the motion controller and the corresponding object in the virtual word. For example, if the motion controller moves 1.5 feet along the x axis, a virtual object moves the same distance in the virtual world. On the other hand, non-isomorphism refers to ability to scale the input so that the control-to-display ratio is not equal to one. For example, if the motion controller is rotated 30 degrees about the y axis, the virtual object may rotate 60 degrees about the y axis. Non-isomorphism is a very powerful approach to 3D spatial interaction because it lends itself to magical interfaces and can potentially give the user more control in the virtual world.

In this section, we will review several common techniques used to perform these basic tasks. Note that it is beyond the scope of this lecture to go into great detail on these techniques or on the concepts of 3D UIs in general. The reader should examine ”3D User Interfaces: Theory and Practice” for a rigorous treatment of the subject [Bowman et al. 2004]. Also note for this section, we assume (for the techniques where it makes a difference) that ’y’ is the vertical axis in the world coordinate system.

2.1 Navigation

The motor component of navigation is known as travel (e.g., viewpoint movement). There are several issues to consider when dealing with travel in 3D UIs. One such issue is the control of velocity and/or acceleration. There are many methods for doing this, including gesture, speech controls, sliders, etc. Another issue is that of world rotation. In systems that are only partially spatially surrounding (e.g. a 4-walled CAVE, or a single screen), the user must be able to rotate the world or his view of the world in order to navigate. In fully surrounding systems (e.g. with an HMD or 6-sided CAVE) this is not necessary since the visuals completely surround the user. Next, one must consider whether motion should be constrained in any way, for example by maintaining a constant height or by following the terrain. Finally, at the lowest-level, the conditions of input must be considered - that is, when and how does motion begin and end (click to start/stop, press to start, release to stop, stop automatically at target location, etc.)? Four of the more common 3D travel techniques are gaze-directed steering, pointing, map-based travel, and ”grabbing the air”.

2.1.1 Gaze-Directed Steering

Gaze-directed steering is probably the most common 3D travel technique, although the term ”gaze” is really misleading. Usually no eye tracking is being performed, so the direction of gaze is inferred from the head tracker orientation. This is a simple technique, both to implement and to use, but it is somewhat limited in that you cannot look around while moving [Mine 1995]. Potential examples of gaze-directed steering in video games would be controlling vehicles or traveling around the world in real-time strategy games.

To implement gaze-directed steering, typically a callback function is set up that executes before each frame is rendered. Within this callback, first obtain the head tracker information (usually in the form of a 4x4 matrix). This matrix gives you a transformation between the base tracker coordinate system and the head tracker coordinate system. By also considering the transformation between the world coordinate system and the base tracker coordinates (if any), you can get the total composite transformation. Now, consider the vector $(0, 0, -1)$ in head tracker space (the negative z-axis, which usually points out the front of the tracker). This vector, expressed in world coordinates, is the direction you want to move. Normalize this vector, multiply it by the speed, and then translate the viewpoint by this amount in world coordinates. Note: current ”velocity” is in units/frame. If you want true velocity (units/second), you must keep track of the time between frames and then translate the viewpoint by an amount proportional to that time.

2.1.2 Pointing

Pointing is also a steering technique (where the user continuously specifies the direction of motion). In this case, the hand's orientation is used to determine direction. This technique is somewhat harder to learn for some users, but is more flexible than gaze-directed steering [Bowman et al. 1997]. Pointing is implemented in exactly the same way as gaze-directed steering, except a hand tracker is used instead of the head tracker. Pointing could be used to decouple line of sight and direction of motion in first and third person shooter games.

2.1.3 Map-based Travel

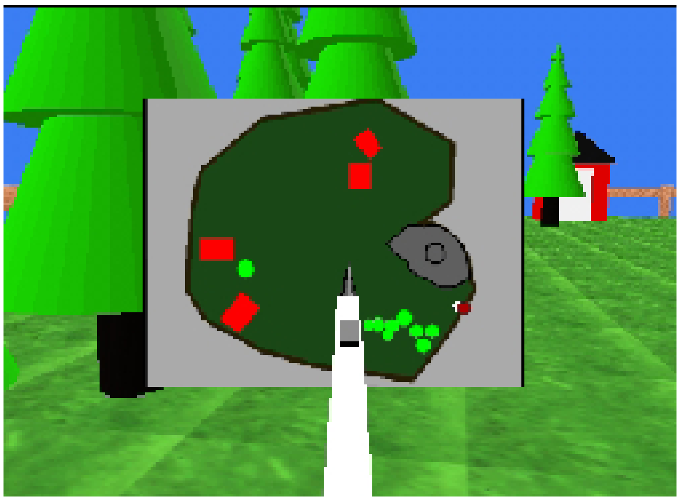


Figure 2: *Dragging a user icon to move to a new location in the world.*

The map-based travel technique is a target-based technique. The user is represented as an icon on a 2D map of the environment. To travel, the user drags this icon to a new position on the map (see Figure 2). When the icon is dropped, the system smoothly animates the user from the current location to the new location indicated by the icon [Bowman et al. 1998]. Map-based travel could be used to augment many of the 2D game maps currently found in many game genres.

To implement this technique, two things must be known about the way the map relates to the world. First, we need to know the scale factor, the ratio between the map and the virtual world. Second, we need to know which point on the map represents the origin of the world coordinate system. We assume here that the map model is originally aligned with the world (i.e. the x direction on the map, in its local coordinate system, represents the x direction in the world coordinate system). When the user presses the button and is intersecting the user icon on the map, then the icon needs to be moved with the stylus each frame. One cannot simply attach the icon to the stylus, because we want the icon to remain on the map even if the stylus does not. To do this, we first find the position of the stylus in the map coordinate system. This may require a transformation between coordinate systems, since the stylus is not a child of the map. The x and z coordinates of the stylus position are the point to which the icon should be moved. We do not cover here what happens if the stylus is dragged off the map, but the user icon should ”stick” to the side of the map until the stylus is moved back inside the map boundaries, since we don’t want the user to move outside the world.

When the button is released, we need to calculate the desired position of the viewpoint in the world. This position is calculated using a transformation from the map coordinate system to the world coordinate system, which is detailed here. First, find the offset in the map coordinate system from the point corresponding to the world origin. Then, divide by the map scale (if the map is 1/100 the size of the world, this corresponds to multiplying by 100). This gives us the x and z coordinates of the desired viewpoint position. Since the map is 2D, we can’t get a y coordinate from it. Therefore, the technique should have some way of calculating the desired height at the new viewpoint. In the simplest case, this might be constant. In other cases, it might be based on the terrain height at that location or some other factors.

Once we know the desired viewpoint, we have to set up the animation of the viewpoint. The move vector $\vec{m}$ represents the amount of translation to do each frame (we are assuming a linear path). To find $\vec{m}$, we subtract the desired position from the current position (the total movement required), divide this by the distance between the two points (calculated using the distance formula), and multiplied by the desired velocity, so that $\vec{m}$ gives us the amount to move in each dimension each frame. The only remaining calculation is the number of frames this movement will take: distance/velocity frames. Note that again velocity is measured here in units/frame, not units/second, for simplicity.

2.1.4 Grabbing the Air

The grabbing the air technique uses the metaphor of literally grabbing the world around you (usually empty space), and pulling yourself through it using hand gestures [Mapes and Moshell 1995]. This is similar to pulling yourself along a rope, except that the rope exists everywhere, and can take you in any direction. The grabbing the air technique has many potential uses in video games including climbing buildings or mountains, swimming, and flying.

To implement the one-handed version of this technique (the two-handed version can get complex if rotation and world scaling is also supported), when the initial button press is detected, we simply obtain the position of the hand in the world coordinate system. Then, every frame until the button is released, get a new hand position, subtract it from the old one, and move the objects in the world by this amount. Alternately, you can leave the world fixed, and translate the viewpoint by the opposite vector. Before exiting the callback, be sure to update the old hand position for use on the next frame. Note it is tempting to implement this technique simply by attaching the world to the hand, but this will have the undesirable effect of also rotating the world when the hand rotates, which can be quite disorienting. You can also do simple constrained motion simply by ignoring one or more of the components of the hand position (e.g. only consider x and z to move at a constant height).

2.2 Selection

3D selection is the process of accessing one or more objects in a 3D virtual world. Note that selection and manipulation are intimately related, and that several of the techniques described here can also be used for manipulation. There are several common issues for the implementation of selection techniques. One of the most basic is how to indicate that the selection event should take place (e.g. you are touching the desired object, now you want to pick it up). This is usually done via a button press, gesture, or voice command, but it might also be done automatically if the system can infer the users intent. One also has to have efficient algorithms for object intersections for many of these techniques. Well discuss a couple of possibilities. The feedback given to the user regarding which object is about to be selected is also very important. Many of the techniques require an avatar (virtual representation) for the users hand. Finally, consider keeping a list of objects that are selectable, so that a selection technique does not have to test every object in the world, increasing efficiency. Four common selection techniques include the virtual hand, ray-casting, occlusion, and arm extension.

2.2.1 The Virtual Hand

The most common selection technique is the simple virtual hand, which does real-world selection via direct touching of virtual objects. In the absence of haptic feedback, this is done by intersecting the virtual hand (which is at the same location as the physical hand) with a virtual object. The virtual hand has great potential in many different video game genres. Examples include selection of sports equipment, direct selection of guns, ammo, and health packs in first person shooter games, as a hand of "God" in real-time strategy games, and interfacing with puzzles in action/adventure games.

Implementing this technique is simple, provided you have a good intersection/collision algorithm. Often, intersections are only performed with axis-aligned bounding boxes or bounding spheres rather than with the actual geometry of the objects.

2.2.2 Ray-Casting

Another common selection technique is ray-casting. This technique uses the metaphor of a laser pointer an infinite ray extending from the virtual hand [Mine 1995]. The first object intersected along the ray is eligible for selection. This technique is efficient, based on experimental results, and only requires the user to vary 2 degrees of freedom (pitch and yaw of the wrist) rather than the 3 DOFs required by the simple virtual hand and other location-based techniques. Ideally, ray-casting could be used whenever special powers are required in the game or when the player has the ability to select objects at a distance.

There are many ways to implement ray-casting. A brute-force approach would calculate the parametric equation of the ray, based on the hands position and orientation. First, as in the pointing technique for travel, find the world coordinate system equivalent of the vector $(0, 0, -1)$. This is the direction of the ray. If the hands position is represented by (x_h, y_h, z_h), and the direction vector is (x_d, y_d, z_d), then the parametric equations are given by

$$\begin{aligned} x(t) &= x_h + x_d t \quad (1) \\ y(t) &= y_h + y_d t \quad (2) \\ z(t) &= z_h + z_d t. \quad (3) \end{aligned}$$

Only intersections with $t > 0$ should be considered, since we do not want to count intersections behind the hand. It is important to determine whether the actual geometry has been intersected, so first testing the intersection with the bounding box will result in many cases being trivially rejected.

Another method might be more efficient. In this method, instead of looking at the hand orientation in the world coordinate system, we consider the selectable objects to be in the hands coordinate system, by transforming their vertices or their bounding boxes. This might seem quite inefficient, because there is only one hand, while there are many polygons in the world. However, we assume we have limited the objects by using a selectable objects list. Thus, the intersection test we will describe is much more efficient. Once we have transformed the vertices or bounding boxes, we drop the z coordinate of each vertex. This maps the 3D polygon onto a 2D plane (the xy plane in the hand coordinate system). Since the ray is $(0, 0, -1)$ in this coordinate system, we can see that in this 2D plane, the ray will intersect the polygon if and only if the point $(0, 0)$ is in the polygon. We can easily determine this with an algorithm that counts the number of times the edges of the 2D polygon cross the positive x-axis. If there are an odd number of crossings, the origin is inside, if even, the origin is outside.

2.2.3 Occlusion Techniques

Occlusion techniques (also called image plane techniques) work in the plane of the image; object are selected by covering it with the virtual hand so that it is occluded from your point of view [Pierce et al. 1997]. Geometrically, this means that a ray is emanating from your eye, going through your finger, and then intersecting an object. Occlusion techniques could be used for object selection at a distance but instead of using a laser pointer metaphor that ray casting affords, players could simply ”touch” distant objects to select them.

These techniques can be implemented in the same ways as the ray-casting technique, since it is also using a ray. If you are doing the brute-force ray intersection algorithm, you can simply define the rays direction by subtracting the finger position from the eye position.

However, if you are using the 2nd algorithm, you require an object to define the rays coordinate system. This can be done in two steps. First, create an empty object, and place it at the hand position, aligned with the world coordinate system. Next, determine how to rotate this object/coordinate system so that it is aligned with the ray direction. The angle can be determined using the positions of the eye and hand, and some simple trigonometry. In 3D, two rotations must be done in general to align the new objects coordinate system with the ray.

2.2.4 Arm-Extension

The arm-extension (e.g., Go-Go) technique is based on the simple virtual hand, but it introduces a non-linear mapping between the physical hand and the virtual hand, so that the user’s reach is greatly extended [Poupyrev et al. 1996]. Not only useful for object selection at a distance, Go-Go could also be useful traveling through an environment with Batman’s grappling hook or Spiderman’s web. The graph in Figure 3 shows the mapping between the physical hand distance from the body on the x-axis and the virtual hand distance from the body on the y-axis. There are two regions. When the physical hand is at a depth less than a threshold D, the one-to-one mapping applies. Outside D, a non-linear mapping is applied, so that the farther the user stretches, the faster the virtual hand moves away.

To implement Go-Go, we first need the concept of the position of the users body. This is needed because we stretch our hands out from the center of our body, not from our head (which is usually the position that is tracked). We can implement this using an inferred torso position, which is defined as a constant offset in the negative y direction from the head. A tracker could also be placed on the users torso.

Before rendering each frame, we get the physical hand position in the world coordinate system, and then calculate its distance from the torso object using the distance formula. The virtual hand distance can

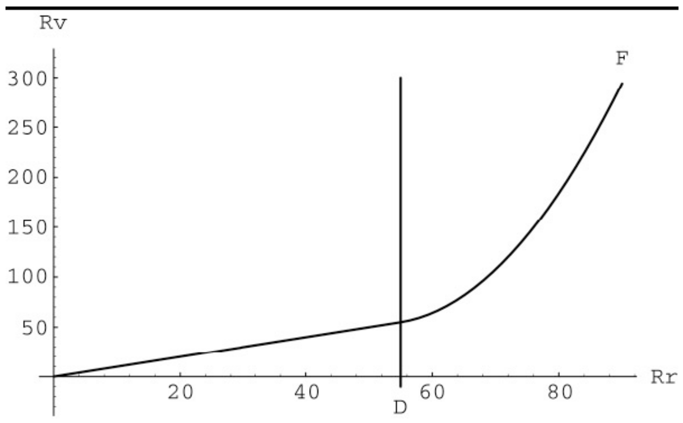


Figure 3: *The nonlinear mapping function used in the Go-Go selection technique.*

then be obtained by applying the function shown in the graph in Figure 3. $d^{2.3}$ (starting at D) is a useful function in many environments, but the exponent used depends on the size of the environment and the desired accuracy of selection at a distance. Once the distance at which to place the virtual hand is known, we need to determine its position. The most common implementation is to keep the virtual hand on the ray extending from the torso and going through the physical hand. Therefore, if we get a vector between these two points, normalize it, multiply it by the distance, then add this vector to the torso point, we obtain the position of the virtual hand. Finally, we can use the virtual hand technique for object selection.

2.3 Manipulation

As we noted earlier, manipulation is connected with selection, because an object must be selected before it can be manipulated. Thus, one important issue for any manipulation technique is how well it integrates with the chosen selection technique. Many techniques, as we have said, do both: e.g. simple virtual hand, ray-casting, and go-go. Another issue is that when an object is being manipulated, you should take care to disable the selection technique and the feedback you give the user for selection. If this is not done, then serious problems can occur if, for example, the user tries to release the currently selected object but the system also interprets this as trying to select a new object. Finally, thinking about what happens when the object is released is important. Does it remain at its last position, possibly floating in space? Does it snap to a grid? Does it fall via gravity until it contacts something solid? The application requirements will determine this choice. Three common manipulation techniques include HOMER, Scaled-World Grab, and World-in-Miniature.

2.3.1 HOMER

The Hand-Centered Object Manipulation Extending Ray-Casting (HOMER) technique uses ray-casting for selection and then moves the virtual hand to the object for hand-centered manipulation [Bowman and Hodges 1997]. The depth of the object is based on a linear mapping. The initial torso-physical hand distance is mapped onto the initial torso-object distance, so that moving the physical hand twice as far away also moves the object twice as far away. Also, moving the physical hand all the way back to the torso moves the object all the way to the users torso as well.

Like Go-Go, HOMER requires a torso position, because you want to keep the virtual hand on the ray between the users body (torso) and the physical hand. The problem here is that HOMER moves the virtual hand from the physical hand position to the object upon selection, and it is not guaranteed that the torso, physical hand, and object will all line up at this time. Therefore, we calculate where the virtual hand would be if it were on this ray initially, then calculate the offset to the position of the virtual object, and maintain this offset throughout manipulation.

When an object is selected via ray-casting, first detach the virtual hand from the hand tracker. This is due to the fact that if it remained attached but the virtual hand model is moved away from the physical hand location, a rotation of the physical hand will cause a rotation and translation of the virtual hand. Next, move the virtual hand in the world coordinate system to the position of the selected object, and attach the object to the virtual hand in the scene graph (again, without moving the object in the world coordinate system).

To implement the linear depth mapping, we need to know the initial distance between the torso and the physical hand d_h, and between the torso and the selected object d_o. The ratio d_o/d_h will be the scaling factor.

For each frame, we need to set the position and orientation of the virtual hand. The selected object is attached to the virtual hand, so it will follow along. Setting the orientation is relatively easy. Simply copy the transformation matrix for the hand tracker to the virtual hand, so that their orientation matches. To set the position, we need to know the correct depth and the correct direction. The depth is found by applying the linear mapping to the current physical hand depth. The physical hand distance is simply the distance between it and the torso, and we multiply this by the scale factor d_o/d_h to get the virtual hand distance. We then obtain a normalized vector between the physical hand and the torso, multiply this vector by the virtual hand distance, and add the result to the torso position to obtain the virtual hand position.

2.3.2 Scaled-World Grab

The scaled-world grab technique (see Figure 4) is often used with occlusion selection. The idea is that since you are selecting the object in the image plane, you can use the ambiguity of that single image to do some magic. When the selection is made, the user is scaled up (or the world is scaled down) so that the virtual hand is actually touching the object that it is occluding. If the user does not move (and the graphics are not stereo), there is no perceptual difference between the images before and after the scaling [Mine et al. 1997]. However, when the user starts to move the object and/or his head, he realizes that he is now a giant (or that the world is tiny) and he can manipulate the object directly, just like the simple virtual hand.

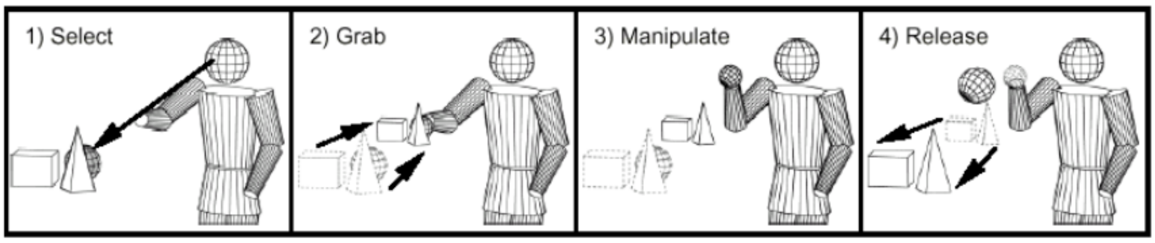


Figure 4: *An illustration of the scaled-world grab technique.*

To implement scaled-world grab, correct actions must be performed at the time of selection and release. Nothing special needs to be done in between, because the object is simply attached to the virtual hand, as in the simple virtual hand technique. At the time of selection, scale the user by the ratio (distance from eye to object / distance from eye to hand). This scaling needs to take place with the eye as the fixed point, so that the eye does not move, and should be uniform in all three dimensions. Finally, attach the virtual object to the virtual hand. At the time of release, the opposite actions are done in reverse. Re-attach the object to the world, and scale the user uniformly by the reciprocal of the scaling factor, again using the eye as a fixed point.

2.3.3 World-in-Miniature

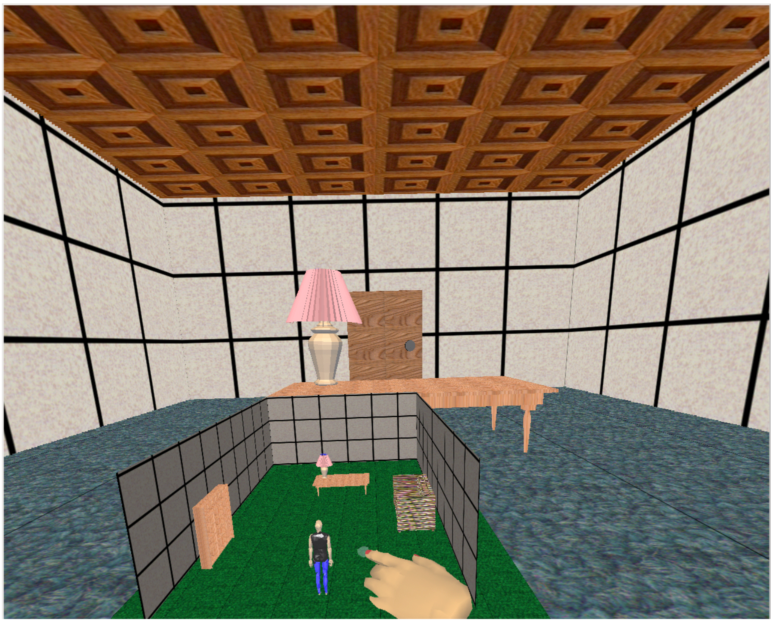


Figure 5: *An example of a WIM.*

The world-in-miniature (WIM) technique uses a small dollhouse version of the world to allow the user to do indirect manipulation of the objects in the environment (see Figure 5). Each of the objects in the WIM are selectable using the simple virtual hand technique, and moving these objects causes the full-scale objects in the world to move in a corresponding way [Stoakley et al. 1995a]. The WIM can also be used for navigation by including a representation of the user, in a way similar to the map-based travel technique, but including the 3rd dimension [Pausch et al. 1995a].

To implement the WIM technique, first create the WIM. Consider this a room with a table object in it. The WIM is represented as a scaled down version of the room, and is attached to the virtual hand. The table object does not need to be scaled, because it will inherit the scaling from its parent (the WIM room). Thus, the table object can simply be copied within the scene graph.

When an object in the WIM is selected using the simple virtual hand technique, first match this object to the corresponding full-scale object. Keeping a list of pointers to these objects is an efficient way to do this step. The miniature object is attached to the virtual hand, just as in the simple virtual hand technique. While the miniature object is being manipulated, simply copy its position matrix (in its local coordinate system, relative to its parent, the WIM) to the position matrix of the full-scale object. Since we want the full-scale object to have the same position in the full-scale world coordinate system as the miniature object does in the scaled-down WIM coordinate system, this is all that is necessary to move the full-scale object correctly.

2.4 System Control

System control provides a mechanism for users to issue a command to either change the mode of interaction or the system state. In order to issue the command, the user has to select an item from a set. System control is a wide-ranging topic, and there are many different techniques to choose from such as the use of graphical menus, voice commands, gestures, and tool selectors. For the most part, these techniques are not difficult to implement, since they mostly involve selection. For example, virtual menu items might be selected using ray-casting. For all of the techniques, good visual feedback is required, since the user needs to know not only what he is selecting, but what will happen when he selects it. In this section, we briefly highlight some of the more common system control techniques.

2.4.1 Graphical Menus

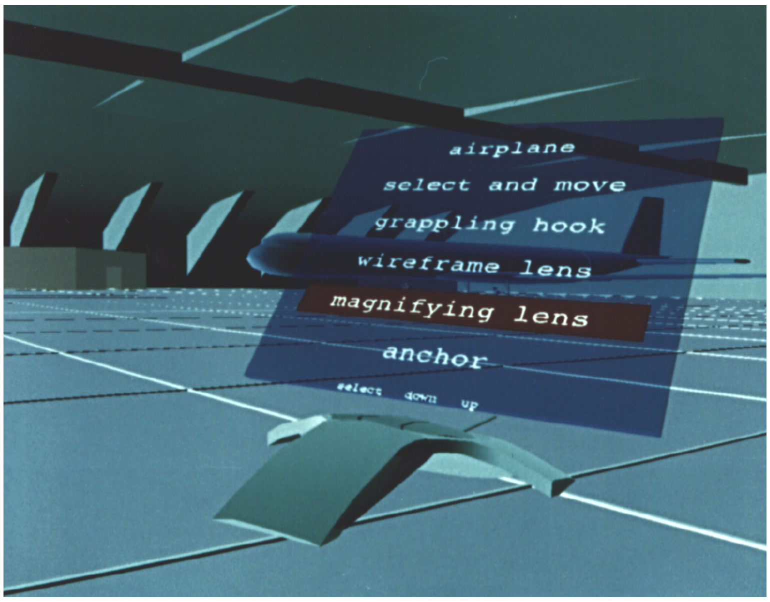


Figure 6: *The Virtual Tricorder: an example of a graphical menu with device-centered placement.*

Graphical menus can be seen as the 3D equivalent of 2D menus. Placement influences the access of the menu (correct placement can give a strong spatial reference for retrieval), and the effects of possible occlusion of the field of attention. The paper by Feiner et al. is an important source for placement issues [Feiner et al. 1993]. The authors divided placement into surround-fixed, world-fixed and display-fixed windows. The subdivision of placement can, however, be made more subtle. World-fixed and surround-fixed windows, the term Feiner et al. use to describe menus, can be subdivided into menus which are either freely placed into the world, or connected to an object. Display-fixed windows can be renamed, and made more precise, by referring to their actual reference frame: the body. Body-centered menus, either head referenced or body-referenced, can supply a strong spatial reference frame. One particularly interesting possible effect of body-centered menus is eyes-off usage, in which users can perform system control without having to look at the menu itself. The last reference frame is the group of device-centered menus. Device-centered placement provides the user with a physical reference frame (see Figure 6) [Wloka and Greenfield 1995]. A good example is the placement of menus on a responsive workbench, where menus are often placed at the border of the display device.

We can subdivide graphical menus into hand-oriented menus, converted 2D menus, and 3D widgets. One can identify two major groups of hand-oriented menus. 1DOF menus are menus which use a circular object on which several items are placed. After initialization, the user can rotate his/her hand along one axis until the desired item on the circular object falls within a selection basket. User performance is highly dependent on hand and wrist physical movement and the primary rotation axis should be carefully chosen. 1DOF menus have been made in several forms, including the ring menu, sundials, spiral menus (a spiral formed ring menu), and a rotary tool chooser. The second group of hand-oriented menus are hand-held-widgets, in which menus are stored at a body-relative position.

The second group is the most often applied group of system control interfaces: converted 2D widgets. These widgets basically function the same as in desktop environments, although one often has to deal with more DOFs when selecting an item in a 2D widget. Popular examples are pull-down menus, pop-up menus, flying widgets, toolbars and sliders.

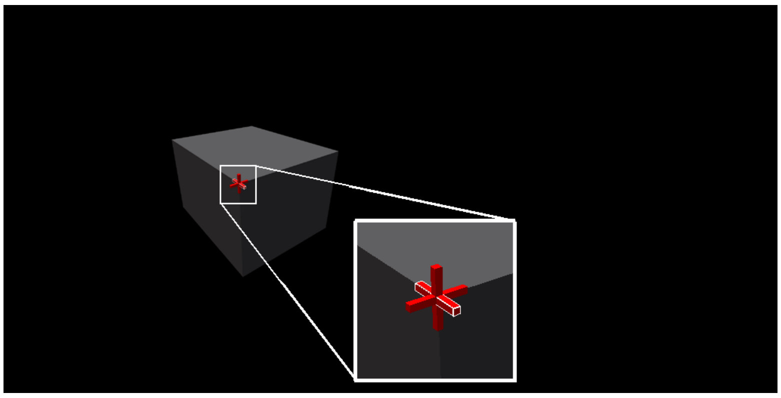


Figure 7: *An example of a 3D widget used to scale a geometric object.*

The final group of graphical menus is the group known as 3D widgets [Conner et al. 1992]. In a 3D world, widgets often mean moving system control functionality into the world or onto objects (see Figure 7). This matches closely with the definition of widgets given by Conner et al., ”widgets are the combination of geometry and behavior”. This can also be thought of as ”moving the functionality of a menu onto an object.” A very important issue when using widgets is placement. 3D widgets differ from the previously discussed menu techniques (1DOF and converted 2D menus) in the way the available functions are mapped: most often, the functions are co-located near an object, thereby forming a highly context-sensitive menu.

2.4.2 Voice Commands

Voice input allows the initialization, selection and issuing of a command. Sometimes, another input stream (like a button press) or a specific voice command is used to allow the actual activation of voice input for system control. The use of voice input as a system control technique can be very powerful: it is hands-free and natural. Still, continuous voice input is tiring, and can not be used in every environment. Furthermore, the voice recognition engine often has a limited vocabulary. In addition, the user first needs to learn the voice commands before they can be applied.

Problems often occur when applications are more complex, and the complete set of voice commands can not be remembered. The structural organization of voice commands is invisible to the user: often no visual representation is coupled to the voice command in order to see the available commands. In order to prevent mode errors, it is often very important to supply the user with some kind of feedback after she has issued a command. This can be achieved by voice output, or by the generation of certain sounds. A very interesting way of supporting the user when interacting with voice and invisible menu structures can be found in telecommunication: using a telephone to access information often poses the same problems to the user as using voice commands in a virtual environment.

2.4.3 Gestures and Postures

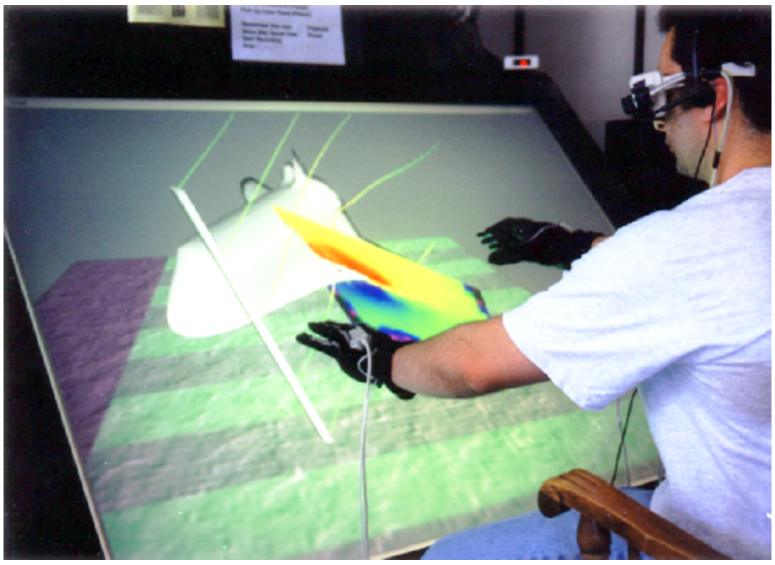


Figure 8: *A user interacting with a dataset for visualizing a flow field around a space shuttle. The user simultaneously manipulates the streamlines with his left hand and the shuttle with his right hand while viewing the data in stereo. The user asked for these tools using speech input.*

When using gestural interaction, we apply a hand-as-tool metaphor: the hand literally becomes a tool. When applying gestural interaction, the gesture is both the initialization and the issuing of a command, just as in voice input. When talking about gestural interaction, we refer, in this case, to gestures and postures, not to gestural input with pen-and-tablet or similar metaphors. There is a significant difference between gestures and postures: postures are static movements (like pinching), whereas gestures include a change of position and/or orientation of the hand. A good example of gestures is the usage of sign language.

Gestural interaction can be a very powerful system control technique. In fact, gestures are relatively limitless when it comes to their potential uses in video games. Gestures can be use to communicate with other players, to cast spells in a role playing game, call pitches or give signs in a baseball game, and issues combination attacks in action games. However, one problem with gestural interaction is that the user needs to learn all the gestures. Since the user can normally not remember more than about 7 gestures (due to the limited capacity of the working memory), inexperienced users can have significant problems with gestural interaction, especially when the application is more complex and requires a larger amount of gestures. Users often do not have the luxury of referring to a graphical menu when using gestural interaction - the structure underneath the available gestures is completely invisible. In order to make gestural interaction easier to use for a less advanced user, strong feedback, like visual cues after initiation of a command, might be needed. An example of application that used a gestural system control technique is MSVT [LaViola 2000]. This application combined gestures and voice input to create a multimodal interface for exploratory scientific visualization (see Figure 8).

2.4.4 Tools

We can identify two different kinds of tools, namely physical tools and virtual tools. Physical tools are context-sensitive input devices, which are often referred to as props. A prop is a real-world object which is duplicated in the virtual world. A physical tool might be space multiplexed (the tool only performs one function) or time multiplexed, when the tool performs multiple functions over time (like a normal desktop mouse). One accesses a physical tool by simply reaching for it, or by changing the mode on the input device itself.

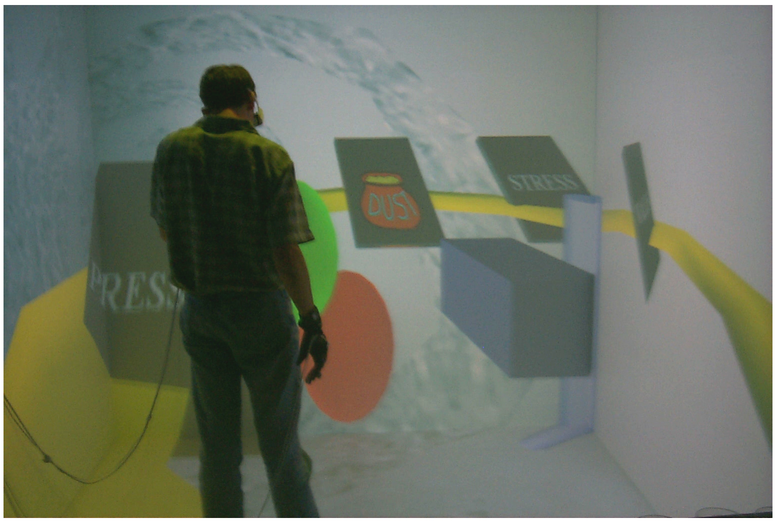


Figure 9: *An example of a virtual toolbelt. The user looks down to invoke the belt and can grab tools and use them in the virtual environment.*

Virtual tools are tools which can be best exemplified with a toolbelt (see Figure 9). Users wear a virtual toolbelt around the waist, from which the user can access specific functions by grabbing at particular places on belt, as in the real world. Sometimes, functions on a toolbelt are accessed via the same principles as used with graphical menus, where one should look at the menu itself. The structure of tools is often not complex: as stated before, physical tools are either dedicated devices for one function, or one can access several (but not many) functions with one tool. Sometimes, a physical tool is the display medium for a graphical menu. In this case, it has to be developed in the same way as graphical menus. Virtual tools often use proprioceptive cues for structuring.

It is still unclear how to best design tools for system control. Still, some general design issues can be stated. In the case of physical tools, the form of the tool often strongly communicates the function one can perform with the device, so take care with the form when developing new props. The form of a tool can highly influence the directness and familiarity with the device as well. With respect to virtual tools which can not be used without looking at them, their representation can be very similar to graphical menus.

One example of a commonly used physically-based tool is the pen and tablet. Users hold a large plastic tablet on which a (traditional) 2D interface is displayed in the virtual world (see Figure 10). Users are able to use graphical menu techniques and can move objects with a stylus within a window on the tablet. The tablet supplies the user with strong physical cues with respect to the placement of the menu, and allows increased performance due to faster selection of menu items [Bowman et al. 1998].

For the implementation of this technique, the most crucial thing is the registration (correspondence) be-

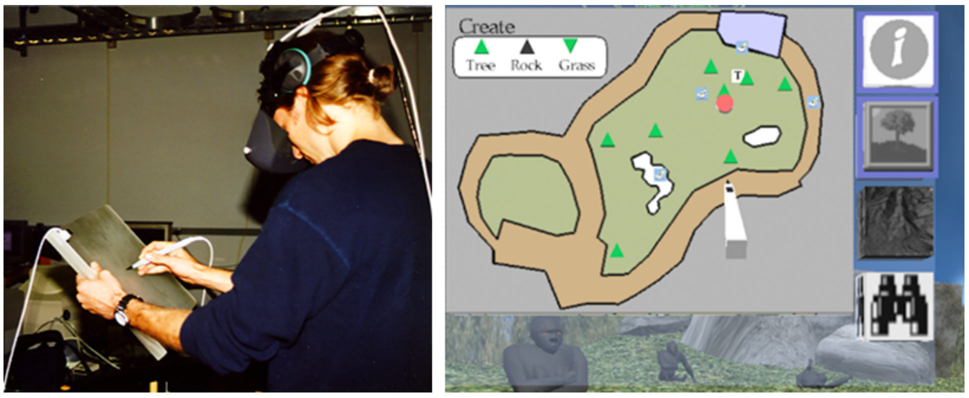


Figure 10: *The Pen and Tablet Metaphor. The image on the left shows the user holding the pen and tablet and the image on the right shows what the user's sees in the virtual world.*

tween the physical and virtual pens and tablets. The tablets, especially, must be the same size and shape so that the edge of the physical tablet, which the user can feel, corresponds to the edge of the virtual tablet as well. In order to make tracking easy, the origin of the tablet model should be located at the point where the tracker is attached to the physical tablet, so that rotations work properly. Even with care, its difficult to do these things exactly right, so a final tip is to include controls within the program to tweak the positions and/or orientations of the virtual pen and tablet, so that they can be into registration if there's a problem.

Another useful function to implement is the ability for the system to report (for example when a callback function is called) the position of the stylus tip in the tablet coordinate system, rather than in the world or user coordinate systems. This can be used for things like the map-based travel technique described earlier.

3 Applications 1: Spatial Interfaces in Art and Design

In this course, we aim to give a broad overview of current and potential applications of spatial interfaces. Applications to art and design can teach us a great deal about how spatial interfaces can be employed creatively. This chapter begins with an overview of early work in the computer graphics and user interface research communities. This research paved the way for a variety of freeform 3D modeling techniques, many of which utilize spatial interfaces to create art and design results that are dramatically different from what we have come to expect from more traditional desktop-based design tools (e.g., CAD tools). After this research context, a variety of specific contemporary applications are discussed. Much of the discussion in this chapter is adapted from Keefe's Ph.D. dissertation [Keefe 2007], a good source for additional information on the topic of spatial interfaces in art and design.

3.1 Background in 3DUI's for Art and Design

This section provides a computer graphics and user interface research context for applications of 3D user interfaces to art and design. The discussion places a special focus on the development of freeform 3D modeling tools, as these interfaces (typically based on sweeping movement of tracked hands or props in 3D space) have played an important role in innovative art and design applications.

3.1.1 Early Research in 3D User Interfaces for Art and Design

Sutherland introduced the first head-mounted, tracked, stereo display in 1968 [Sutherland 1968]. Eight years later, Clark combined this hardware with a tracked-wand device to make the first virtual reality system for direct manipulation of 3D surfaces [Clark 1976]. Clark's work introduced the idea that the intuitive control afforded by direct 3D interaction coupled with a 3D display could be a better paradigm for 3D modeling tasks than more indirect methods involving 2D projections of 3D objects or mapping 2D input to a 3D space. He also discussed the implications of this approach for the design process. Freeing designers from needing to know particular numerical coordinate values describing a surface or orientations of coordinate system axes allows them to concentrate fully on the design task. These concepts have inspired many researchers, artists, and designers who have followed.

Schmandt [Schmandt 1983] was the first we know to implement a modeling system that could generate form from sweeping movements of a 3D input device. Schmandt's system used an early Polhemus 6D device to create a magic wand that could emit 3D paint in space. He used half-silvered mirrors, conventional video monitors, and shutter glasses to produce a stereo view. Schmandt's work was one of the first experiments on the interactive capabilities of stereo displays. His results indicated a good natural correspondence between the wand and the 3D paint, but lag and distortion in the tracking field detracted significantly from the interactive feel of the tool.

Galyean and Hughes [Galyean and Hughes 1991] used direct 3D input with a monoscopic display to produce a voxel-based modeling technique. They created a "poor man's" 3D force-feedback device to assist in controlling the input tool by suspending a Polhemus tracker in a cube with eight elastic cords attached to the cube corners. This work was the first of the 3D sculpting systems to use a clay metaphor and one of the first to examine creating artistic, blobby models that were a drastic departure from the more rigid geometric shapes produced with CAD programs. In this system, material could be cut away from or added to a block of clay. A large body of work based on this metaphor has followed, and virtual clay remains one of the most successful approaches to obtaining output in the form of voxel-based descriptions or enclosed triangle meshes, which are useful for applications in rapid prototyping and animation. Advances in computer and haptic hardware continue to make possible a wide variety of extensions to this work that provide more control of the interface and more sophisticated forms.

At roughly the same time as the work by Galyean and Hughes, Sachs et al. presented the 3-Draw system [Sachs et al. 1991], which used direct 3D input with both hands to create an intuitive CAD modeling system. This work was ground breaking in establishing computers as a viable tool for the initial phases of industrial design. 3-Draw models consisted of lines only, but the lines can easily be thought of as defining surfaces. The tool allowed both unconstrained and various types of semi-constrained sweeping input to specify complex 3D curves and provided interaction techniques for specifying start and end points for curves and several curve-editing operations. This work was important in establishing free-form 3D input as a viable tool for serious design problems and in presenting interaction techniques for enhancing control over free-form input.

The 3DM system [Butterworth et al. 1992] presented by Butterworth et al. built on Clark's conception of a head-mounted display modeling environment with the additional goal of providing the user with an interface that was as easy to learn and use as that commonly available in 2D drawing programs of the time, such as MacPaint. In addition to performing geometrically constrained CAD-style modeling operations, 3DM incorporated a modeling mode based on sweeping 3D input for creating surfaces by extruding curves. One of the important contributions of 3DM with respect to future art and design applications is its discussion of user experiences with this extrusion tool. Users were reported to have difficulty aligning 3D objects, keeping two triangles parallel, and doing other geometrically constrained operations in this VR environment, but for the first time they could easily perform highly complex free-form extrusions. Extrusions had already been proven to be extremely useful modeling tools in desktop-based programs. The difference in 3DM was the user's ability to perform the extrusion directly in 3D by dragging and twisting an extrusion curve along a free-form path. In this way, users could naturally control more than one parameter (position and twist) of the extrusion while creating it.

Deering's Holosketch [Deering 1995] was the first system to combine a head-tracked stereo VR environment with a modeling system geared towards artistic creation. Holosketch was a strikingly complete modeling and animation package with a fully developed menu of modeling modes and operations. Several of Holosketch's drawing modes were based on continuous sweeping input, including a toothpaste mode reminiscent of Gaylean and Hughes' additive sculpting [Galyean and Hughes 1991], a wire-frame-lines mode reminiscent of the 3D line drawings of 3-Draw [Sachs et al. 1991], and a mode in which clouds of random small triangle particles were left behind the wand as it was swept through the air. Holosketch worked in a fishtank VR [Deering 1992] setup using a 20-inch CRT and was also explored in alternative VR form factors.

3.1.2 Recent Research in 3D User Interfaces for Art and Design

In more recent work, two classes of free-form modeling techniques that utilize direct, sweeping 3D input have emerged: those based on large-scale movements of a tracked device in the air and those based on haptic feedback, usually utilizing the clay sculpting metaphor first introduced by Gaylean and Hughes [Galyean and Hughes 1991].

Schkolne et al.'s Surface Drawing [Schkolne et al. 2001] is an example of free-form modeling utilizing a large-screen table display device. This display lends itself to the use of physical props that can be placed on the table when not in use. Natural use and selection of appropriate props and metaphors is the main thrust of Schkolne's work [Schkolne 2003; Schkolne et al. 2004]. To create form in the system, the artist uses his hand augmented with a bend sensing glove as a device for sweeping out bits of surface in space. These surface fragments are then stitched together to create a bigger triangle-mesh model.

Several other large-scale, open-air input systems have been created for free-form modeling. Keefe et al.'s CavePainting system, described in more detail in the following sections, uses similar input metaphors (sweeping movements of the hands in space) to create 3D form [Keefe et al. 2001]. The FreeDrawer [Wesche and Seidel 2001] system runs on a responsive workbench and is a good example of a modern approach to a spline-based modeling system that utilizes sweeping 3D input.

In the area of haptics-based free-form modeling, SensAble Technologies, makers of the Phantom force feedback device, introduced what was probably the first haptics modeling system that allowed artists to feel the 3D shapes they modeled while pushing, pulling, and deforming them. The tool, which has undergone considerable refinements and is still available today is called FreeForm™ [SensAble Technologies, Inc. 2005]. The current incarnation targets product designers working anywhere from the early conceptual stages of product design to final steps where output from the modeling tool can be sent to rapid prototyping machines for production of physical models. The models that skilled artists can create with this system are impressive in their precision and refined aesthetic. Similar approaches have also been presented within the research community (e.g., [Gregory et al. 2000], [Hua and Qin 2004]).

One haptics-based modeling system that takes a different approach from the rest is the springs-and-constraints 3D drawing system presented by Snibbe et al. [Snibbe et al. 1998]. Snibbe et al.'s approach is different in that it uses dynamic haptic models to help artists explore various modes of creation, but these models are not based on interacting with a static geometry or properly simulating contact forces. Rather, they help to control and guide free-form input, but tend to allow artists to remain gestural in their interactions. The artistic potential of the new medium described by Snibbe et al. is left relatively unexplored.

Of final note is Schroering et al.'s work [Schroering et al. 2003] in which a light pen is used to draw on a tablet that is tracked as it is moved through the air. The drawing/modeling application presented in this work is quite limited in its scope, but this input style could have important implications for free-form modeling in 3D.

3.2 Example 3D User Interfaces for Art Applications

Figure 11 shows a number of 3D models created with the CavePainting system mentioned earlier [Keefe et al. 2001], and Figure 12 shows the haptic Drawing on Air interface that evolved from the original CavePainting tool. Both of these systems were designed to explore the potential of 3D user interfaces in art applications. Keefe describes his art practice with the two systems over a period of nearly a decade in an article to appear shortly in the journal, Leonardo [Keefe 2011, In press]. One of the most interesting points of feedback from the many artists who have used the systems and critiqued work generated with the tools is that the style of form generated using these 3D user interfaces is quite different from what is available via other tools. For example, in the works shown in Figure 11, each 3D model includes some visual evidence that a human hand was involved in creating the model – this is rarely seen with models created using traditional desktop modeling tools. Similarly, there is a painting-like aesthetic in the models produced with these tools that is quite distinct from the watertight meshes produced with more traditional modeling systems. While these unique aesthetic qualities are not desirable for all applications, they are particularly exciting for artists interested in exploring the connection between human movement and computer generated 3D form.

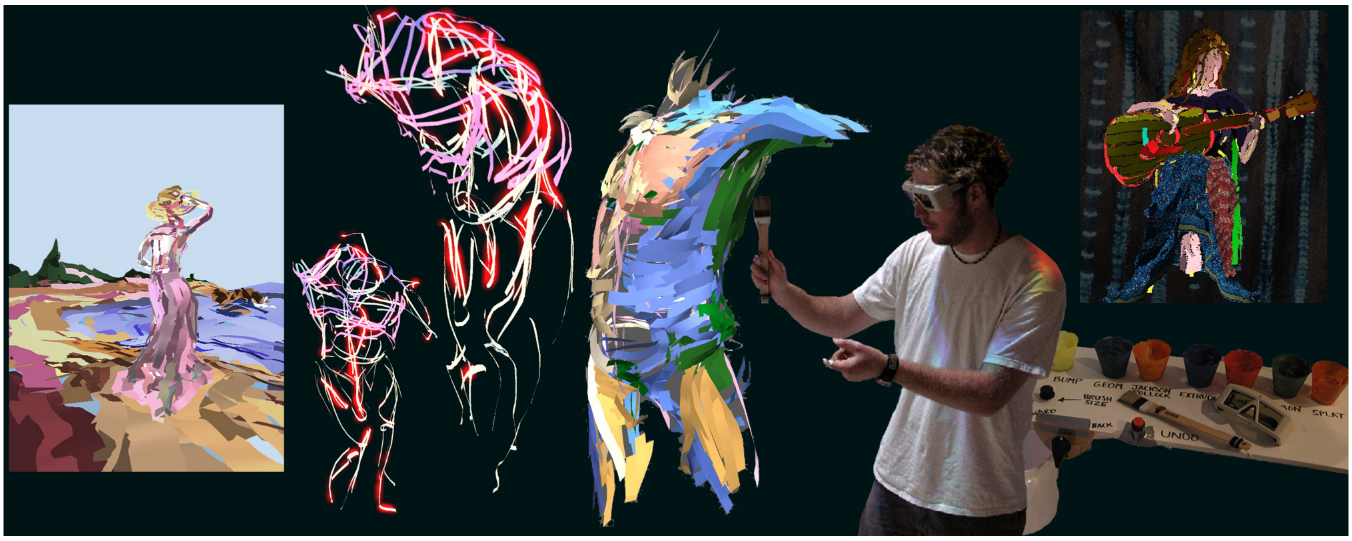


Figure 11: *A composite of several 3D virtual sculptures drawn in space using the CavePainting 3D user interface.*

Several other artists have worked with 3D user interfaces, especially in the area of 3D modeling and sketching. Of note is the work of the artist Mäkelä, who has teamed with researchers at Helsinki University of Technology to create a system in which form can be generated by tracking fingertips [Mäkelä 2005]. The fingertip control, achieved through a custom ultrasonic input device, adds the ability to control the thickness of swaths of form as they are swept out in space, a feature similar to that provided by the haptic Drawing on Air interface [Keefe et al. 2007]. Mäkelä's work also illustrates compelling visual effects that are possible when combining point- and surface-based representations. Artist Jen Zen has worked extensively with 3D user interfaces as a mode of tracing form in space [Grey 2002]. The BLUI system [Brody and Hartman 2000] has been the topic of several sketches at SIGGRAPH in which physical printouts, both 2D and 3D, of free-form objects created with the system have been presented. The IMAX SANDDE™digital system, used in the films June [Ferguson 2003], Falling in Love Again [Ferguson 2006], and Moon Man [Morstad 2004], is also based on freehand 3D drawing with a tracked wand in a stereoscopic environment.

3.3 Installation and Interactive Art

3D user interfaces have also been explored in installation and interactive art environments. New media writing projects have been developed and shown extensively within CAVE environments [Brown University Literary Arts Program 2011]. 3D input has many times been coupled with live dance performances, and new research continues to advance the possibilities of linking physical and virtual performers [Sheppard and Nahrstedt 2009]. Other artists have explored mappings between spatial input provided by audience members walking through an artwork and the form displayed in the work. Rubin and Keefe explored this concept in a Cave of Elusive Immateriality, presented at SIGGRAPH 2002 [Rubin and Keefe 2002]. Snibbe extends similar 3D input concepts to installations that involve multiple viewers [Snibbe 1998].

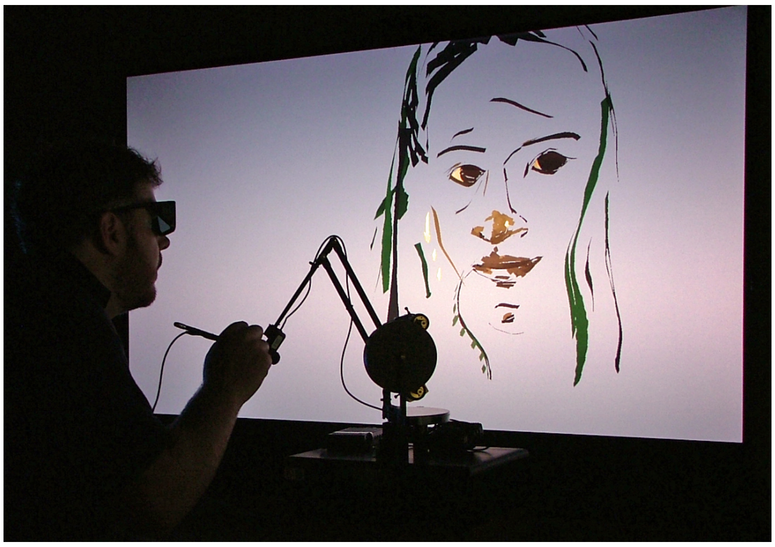


Figure 12: *The haptic-based Drawing on Air 3D interface.*

3.4 Applications in Creative 3D Design

Many applications of 3D user interfaces to creative design problems are closely related to applications in art. For example, building directly on the freeform modeling applications described earlier, Keefe et al. have applied 3D user interfaces to the problem of developing early stage 3D prototyping tools for designing complex scientific visualizations [Keefe et al. 2008a]. While current commercial computer-aided design tools allow designers to create amazingly intricate 3D models, what designers often miss in these tools is an ability to work as naturally and expressively as with a pencil and paper. 3D user interfaces can provide new ways to work with 3D design problems using tools that are immediate and gestural, similar to traditional sketching.

The ModelCraft interface developed by Song et al. [Song et al. 2006] is another particularly interesting design-oriented 3D user interface. The system uses an Anoto pen and paper. The paper can be folded into various 3D shapes, and then pen markings made directly on the shapes are used to indicate 3D design actions, such as cutting away or extruding material.

In still other design-oriented applications of 3D user interfaces, several researchers and practitioners have explored the use of 3D input for architectural design. For example, Anderson et al. describe a virtual reality design system and the benefits it has provided, including a new ability to simultaneously work at multiple scales [Anderson et al. 2003].

A final example is motivated by automotive design and creative uses of spray paint in art. Konieczny et al., present a virtual spray paint simulator [Konieczny et al. 2008]. Many other applications of 3D user interfaces to creative design tasks exist; 3D user interfaces show great potential for creative 3D art and design work, especially in the early stages of conceptual design.

4 Working with Video Game Motion Controllers

With the advent of the motion controller on every major console and PC gaming platform, the video game industry has made it possible for literally anyone to build and explore 3D spatial interfaces for a variety of different applications. These devices, specifically, the Nintendo Wii Remote, Microsoft Kinect, and Playstation Move, provide user tracking devices that are both inexpensive and accurate. In this section, we will explore how these devices work and what is needed to start using them to build 3D user interfaces.

4.1 Motion Controller Device Characteristics

4.1.1 Nintendo Wii Remote

The Nintendo Wii Remote (Wiimote) presents three axes of acceleration data in no particular frame of reference, with intermittent optical sensing. (The Wii MotionPlus gyroscope attachment can add three axes of orientation change, or angular velocity.) Although this means the Wiimote's spatial data doesn't directly map to a real-world positions, the device can be employed effectively under constrained use [Shirai et al. 2007; Shiratori and Hodgins 2008]. The Wiimote incorporates several buttons, some in a gamepad and trigger configuration, and has a speaker, programmable LEDs, and a rumble device. It is easy to set up, turn on, and maintain. Overall, the Wiimote incorporates many useful input and output features in an inexpensive, consumer-oriented, easy-to-replace, and easy to repurchase package. This lets game developers, researchers, and homebrew engineers use and modify it to best serve their needs.

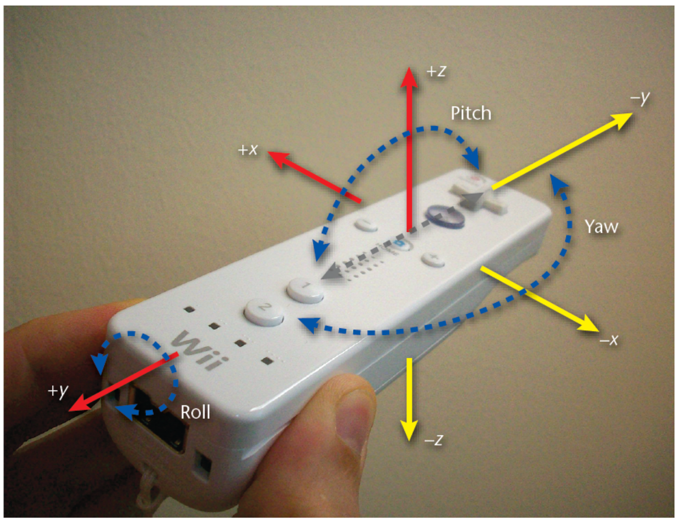


Figure 13: *The Wiimote, with labels indicating the Wiimote's coordinate system. Multiple coordinate systems and partial spatial data make the Wiimote difficult to design for.*

Frames of Reference. Figure 13 shows the Wiimote's FOR. The x, y and z axes are labeled, along with the rotation about each axis: pitch, roll, and yaw, respectively. A second FOR is the Earth's, important because the Wiimote's accelerometers detect the Earth's gravity. A third FOR is the Wiimote's relationship to the sensor bar. Three examples clarify how these FOR interact. To begin, a user is considered holding a Wiimote naturally when +z is up in both the Wiimote's and the Earth's FOR and the Wiimote's front points away from the user and toward a sensor bar, which is usually on top of the display. In the first example, the user moves the Wiimote toward the sensor bar. This results in acceleration reported in the y-axis of both the Earth and Wiimote FORs and the sensor bar reporting decreased distance between the Wiimote and it. In the second example, the user rotates the Wiimote down to point toward the earth (a 90 pitch). When the user again moves the Wiimote directly toward the sensor bar in front of him or her, the Wiimote reports acceleration in its z-axis. However, the Earth's FOR has acceleration in its y-axis because the 90 downward pitch didn't change the Earth's FOR. The sensor bar has no FOR because as the Wiimote rotates away from the sensor bar, the Wiimote loses contact with it. The third example has the same configuration as the second example, but with the sensor bar on the ground directly below the Wiimote, at the user's feet. When the user repeats that forward motion, the sensor bar reports the Wiimote moving up in the sensor bar's z-axis, whereas the Earth and Wiimote data are the same as in the previous example. These examples show that each FOR captures important and different information. The following sections explain this in more detail.

The Sensor Bar Connection. The SBC is one of the Wiimote's two primary spatial sensors. It occurs when the Wiimote's infrared optical camera points at a sensor bar and sees the infrared (IR) light emitted by its LEDs. A sensor bar has LEDs on each side (see Figure 14), with a known width between them. This produces IR blobs that the Wiimote tracks and reports in x and y coordinates, along with the blob's width in pixels. To improve tracking distance (the Wiimote can sense blobs up to 16 feet away), the sensor bar LEDs are spread in a slight arc, with the outer LEDs angled out and the inner LEDs angled in. Actually, any IR source will work, such as custom IR emitters, candles, or multiple sensor bars (provided you have the means to differentiate between the sensor bars).

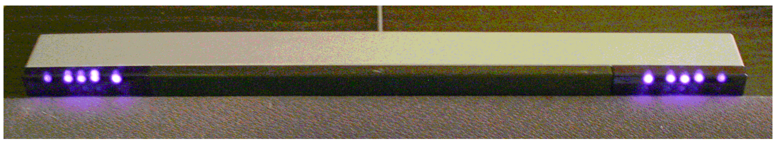


Figure 14: *The Wiimote sensor bar has two groups of IR LEDs at fixed widths.*

When the Wiimote is pointed at the sensor bar, it picks up two points $P_L = (x_L, y_L)$ and $P_R = (x_R, y_R)$ from the LED arrays. The midpoint between these P_L and P_R can easily be calculated and used as a 2D cursor or pointer on the display. In addition, if the Wiimote is rotated about the Y-axis, we can calculate the roll of the device with respect to the X-axis using

$$roll = \arccos\left(\vec{x} \cdot \frac{\vec{v}}{\|\vec{v}\|}\right) \tag{4}$$

where $\vec{x} = (1, 0)$ and $\vec{v} = P_L - P_R$.

Combining information from the sensor bar and the Wiimote's optical sensor also make it possible to determine how far the Wiimote is from the sensor bar using triangulation. The distance d between the sensor bar and Wiimote is calculated using

$$
\begin{aligned}
d &= \frac{w/2}{\tan(\theta/2)} && (5)\\
w &= \frac{m \cdot w_{img}}{m_{img}} && (6)\\
m_{img} &= \sqrt{(x_L - x_R)^2 + (y_L - y_R)^2} && (7)
\end{aligned}
$$

where θ is the optical sensor's viewing angle, m is the distance between the sensor bar's left and right LEDs, w_{img} is the width of the image taken from the optical sensor, and m_{img} is the distance between P_L and P_R taken from the optical sensor's image. Note that θ, w_{img}, and m are all constants. This calculation only works when the Wiimote device is pointing directly at the sensor bar (orthogonally). When the Wiimote is off-axis from the sensor bar, more information is needed to find depth. We can utilize the relative sizes of the points on the optical sensor's image to calculate depth in this case. To find d in this case, we compute the distances corresponding to the the left and right points on the optical sensor's image

$$
\begin{aligned}
d_L &= \frac{w_L/2}{\tan(\theta/2)} && (8)\\
d_R &= \frac{w_R/2}{\tan(\theta/2)} && (9)
\end{aligned}
$$

using

$$
\begin{aligned}
w_L &= \frac{w_{img} \cdot diam_{LED}}{diam_L} && (10)\\
w_R &= \frac{w_{img} \cdot diam_{LED}}{diam_R} && (11)
\end{aligned}
$$

where $diam_{LED}$ is the diameter of the actual LED marker from the sensor bar and $diam_L$ and $diam_R$ are the diameters of the points found on the optical sensor's image. With d_L and d_R, we can then calculate Wiimote's distance to the sensor bar as

$$
d = \sqrt{d_L^2 + (m/2)^2 - 2d_L(m/2)^2\cos(\phi)} \tag{12}
$$

where

$$
\cos(\phi) = \frac{d_L^2 m^2 - d_R^2}{2md_L}. \tag{13}
$$

Note that with d, d_L, and m we can also find the angular position of the Wiimote with respect to the sensor bar, calculated as

$$\alpha = \arccos\left(\frac{d^2m^2 - d_L^2}{md_L}\right). \tag{14}$$

3-Axis Accelerometer. The second Wiimote input is the device's 3-axis accelerometer. The accelerometer reports acceleration data in the device's x, y and z directions, expressed conveniently in g's. This is common of many devices employing 3-axis accelerometers such as the cell phones like the iPhone, laptops and camcorders. With this information, the Wiimote is able to sense motion, reporting values that are a blend of accelerations exerted by the user and gravity. As the gravity vector is constantly oriented towards the Earth (or $(0, 0, 1)$ in Earth's FOR), the gravity vector can be used to discover part of the Wiimote's orientation in terms of earth's frame of reference using

$$\begin{aligned} pitch &= \arctan\left(\frac{a_z}{a_y}\right) & (15) \\ roll &= \arctan\left(\frac{a_z}{a_x}\right). & (16) \end{aligned}$$

Unfortunately, determining yaw in the Earth's FOR isn't possible because the Earth's gravity vector aligns with its z-axis. Another unfortunate issue is that determining the actual acceleration of the Wiimote is problematic owing to the confounding gravity vector. To determine the actual acceleration, one of the following must take place:

- the Wiimote must be under no acceleration other than gravity so that you can accurately measure the gravity vector (in which case, you already know the actual acceleration is zero),
- you must make assumptions about the Wiimote's orientation, thus allowing room for errors, or
- you must determine the orientation by other means, such as by the SBC or a gyroscope (we discuss this in more detail later).

The implications for orientation tracking by the accelerometers are that the Wiimote's orientation is only certain when it is under no acceleration. For this reason, many Wii games require that users either hold the Wiimote steady for a short period of time before using it in a game trial or have it pointed at the screen and orient by the SBC.

Wii MotionPlus. This attachment uses two gyroscopes to report angular velocity along all three axes (one dual-axis gyro for x and y and a single-axis gyro for z). Mechanical gyroscopes would typically be too large and expensive for a Wiimote. So, it uses MEMS (microelectromechanical system) gyroscopes, which operate using a vibrating structure, are inexpensive, use little power, and are fairly accurate. A MotionPlus-augmented Wiimote provides information on changes to the Wiimote's orientation, alleviating many of the device's data limitations. With this, Nintendo is attempting to improve the orientation accuracy of the device.

The MotionPlus isn't yet fully reverse engineered. It reports orientation changes in two granularities, fast and slow, with fast being roughly four times the rate per bit. The gyroscope manufacturer reports that the two gyroscopes have a linear gain but that the different gyroscopes report values in two different scales, so there's no single scaling factor. Additionally, temperature and pressure changes can impact this scale factor and change the value associated with zero orientation change.

Merging the acceleration and gyroscopic data isn't simple; both sensors have accuracy and drift errors that, albeit small, amount to large errors over short time periods. When using the SBC, you can compensate for these accumulating errors by providing an absolute orientation and position. Researchers have improved orientation by merging accelerometer and gyroscopic data but didn't test a system under translational motion [Luinge et al. 1999]. Other research has shown that you can combine accelerometers and gyroscopes for accurate position and orientation tracking [Williamson and Andrews 2001]. In addition, researchers have successfully used Kalman filters to merge accelerometer and gyroscopic data [Azuma and Bishop 1994; Williamson et al. 2010]

4.1.2 Microsoft Kinect

The Microsoft Kinect is an accessory for the XBOX 360 console that turns the users body into the controller. It is able to detect multiple bodies simultaneously and use their movements and voices as input.

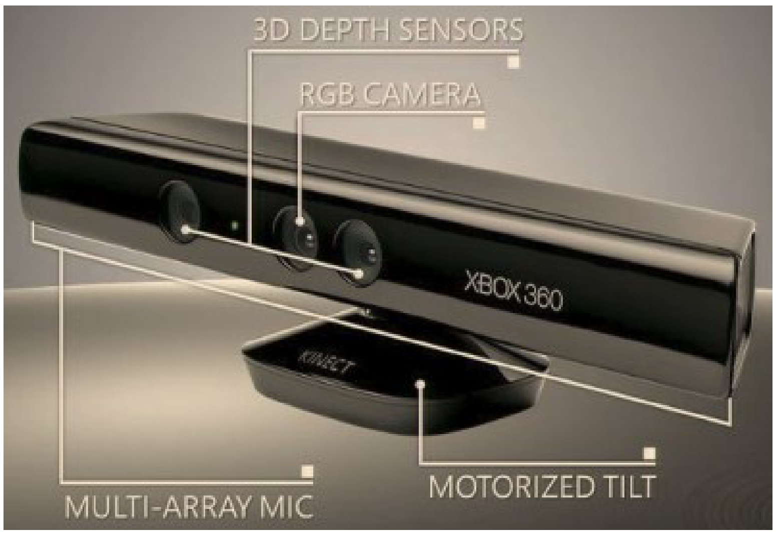


Figure 15: *The Microsoft Kinect sensor.*

The hardware for the Kinect (see Figure 15) is comprised of a color camera, a depth sensor, a multiarray microphone, and a motorized tilt system. The camera is used to determine different features of the user and space by detecting RGB colors and is mainly used for facial recognition of the user. The multi-array microphone is a set of four microphones that are able to isolate the voices of multiple users from the ambient noises in the room. It makes use of acoustic source localization and ambient noise suppression allowing users to be a few feet away from the device but still be able to use the voice controls in a headset-free manner. The third component of the hardware, the depth sensor (generally referred to as the 3D camera), combines an infrared laser projector and a CMOS (complimentary metal-oxide semiconductor) sensor. The infrared projector casts out a myriad of infrared dots (see Figure 16) that the CMOS sensor is able to see regardless of the lighting in the room. This is, therefore, the most important portion of the Kinect which allows it to function.

To acquire 3D depth information, software component of the Kinect interprets the data from the CMOS sensor. Rays are cast out via the infrared projector in a pseudo-random array across a large area. The CMOS sensor is able to then read the depth of all of the pixels at 30 frames per second. It is able to do this because it is an active pixel sensor (APS), which is comprised of a two-dimensional array of pixel sensors. Each pixel sensor has a photo detector and an active amplifier. This camera is used to detect the

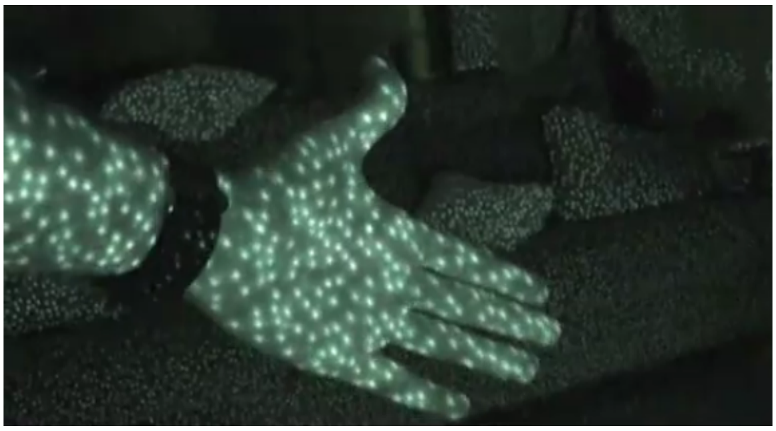


Figure 16: *The structured light pattern generated by the Kinect's infrared laser projector.*

location of the infrared dots. Following this, depth calculations are performed in the scene using stereo triangulation. Stereo triangulation requires two cameras to be able to perform this calculation. The depth measurement requires that corresponding points in one image need to be found in the second image. Once those corresponding points are found, we can then find the disparity (the number of pixels between a point in the right image and the corresponding point in the left image) between the two images. If the images are rectified (along the same parallel axis), then, once we have the disparity, we can then use triangulation to calculate the depth of that point in the scene.

Traditionally, stereoscopic triangulation requires two cameras, the Kinect is unique in that the depth sensor only has one camera to perform these calculations. The infrared projector is, in and of itself, a camera in the sense that it has an image to compare with the image taken from CMOS sensor camera. The projected speckles are semi-random in the fact that they are a generated pattern that the Kinect understands. Since the device knows where the speckles are located, it has an image which can be compared to find the focal points. The CMOS sensor captures an offset image to detect differences in the scene where the disparity between dots can be analyzed and the depth can therefore be calculated. Assuming the images are rectified, the depth calculations are straightforward. The depth data is then interpreted and used in the system. To visualize the depth information, a depth image can be generated by assigning a color coding to the data as shown in Figure 17.

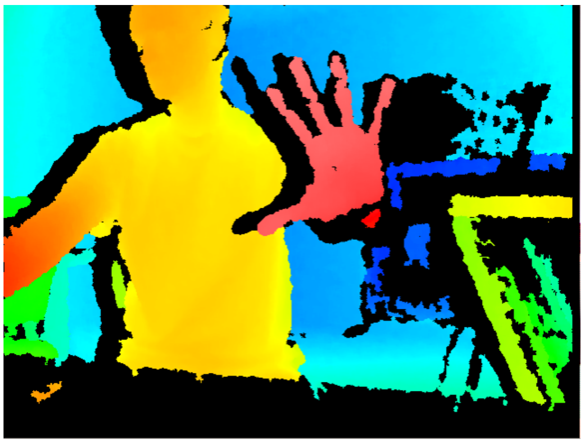


Figure 17: *A depth image from the Microsoft Kinect.*

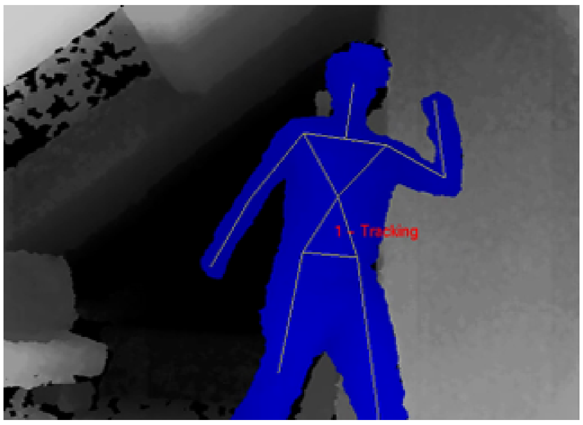


Figure 18: *A skeleton representation of a user acquired from the Microsoft Kinect.*

The Kinect software is able to track users' skeletons by combining the depth information and knowledge about human body kinematics. This knowledge was obtained by gathering and labeling data from special rigs that captured user motions in everyday life. The images and labels were used for training a machine learning algorithm to create probabilities and statistics about the human form and movement.

The software goes through a series of steps to make sense of the input from the camera and have the users body be the controller. Beginning with the user stepping in front of the Kinect, a 3D surface is generated using the depth information, creating a point cloud of the user. The Kinect then creates a guess at the users skeleton. Next, using using the kinematic data,the Kinect makes an educated guess at determining the different parts of the body. A level of confidence is also assigned to each guess based on how confident the algorithm is about guessing the correct parts. Once this is done, the Kinect finds the most probable skeleton (see Figure 18 that would fit these body parts and their confidence levels assigned to them. This is performed real tine at 30 frames per second. For more detail on the Kinect skeleton tracking algorithm, see Shotton et al. [Shotton et al. 2011].

4.1.3 Playstation Move

The PlayStation Move system consists of a PlayStation Eye and one to four PlayStation Move motion controllers (see Figure 19). It combines the advantages of camera tracking and motion sensing with the reliability and versatility of buttons. The wireless controller is used with one hand, and it has several digital buttons on the front and a long-throw analog T button on the back. Internally, it has several MEMS inertial sensors, including a three-axis gyroscope and three-axis accelerometer. But the most distinctive feature of the controller is the 44mm-diameter sphere on the top which houses a RGB LED that applications can set to any color. The sphere color can be varied to enhance interaction, but the primary purpose of the sphere is to enable reliable recovery of the controller 3D position using color tracking with the PlayStation Eye.

The illuminated sphere design solves many of the color tracking issues experienced with original EyeToy. Because the sphere generates its own light for the camera to see, scene lighting is much less of an issue. Tracking works perfectly in a completely dark room, and for scenes with highly varied lighting, the generated light mitigates the variability. Also, the light color can be adjusted to ensure it is visually unique with respect to the rest of the scene. Finally, the choice of the sphere shape makes the color tracking invariant to rotation; this simplifies position recovery and improves position precision by allowing a strong model to be fit to the projection.

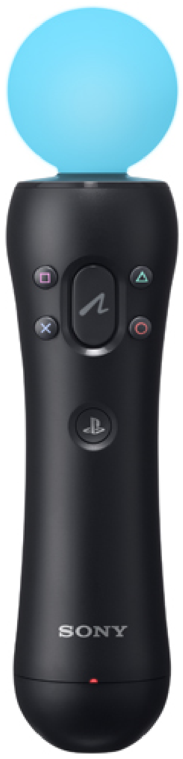


Figure 19: *The PlayStation Move motion controller.*

Deriving the Playstation Move state involves two major steps: image analysis and sensor fusion. Though the exact details of these steps are beyond the scope of these notes, the following overview provides a qualitative understanding of each step.

Image Analysis. Conceptually, the image analysis can be broken up into two stages: finding the sphere in the image and then fitting a shape model to the sphere projection. Color segmentation is used to find the sphere and includes two steps; segmentation and pose recovery. Segmentation consists of labeling every video pixel that corresponds to the object being tracked. General shape segmentation for arbitrary objects is a difficult and computationally expensive problem. However, by designing the tracking object to have distinctive, solid, saturated colors, the object segmentation process can be simplified to basic color segmentation. By choosing extremely saturated colors, the likelihood of these colors occurring in the environment is reduced. Color segmentation can be accomplished in a single pass through the image using chrominance banding/thresholding. Figure 20 shows the results of segmentation for a ”mace” object consisting of a large orange ball mounted on a blue stick. By using only chrominance and not luminance, the segmentation process can be more robust to variation caused by scene lighting.

Pose recovery consists of converting 2D image data into 3D object pose (position and/or orientation). Pose recovery can be accomplished robustly for certain shapes of known physical dimensions by measuring the statistical properties of the shape’s 2D projection. In this manner, for a sphere the 3D position can be recovered (but no orientation), and for a cylinder, the 3D position and a portion of the orientation can be recovered. Multiple objects can be also be combined for complete 3D pose recovery, though occlusion issues arise.

The exact color thresholds used for segmentation are based on results from initial calibration of the sphere LED brightness and room lighting which is performed at the start of play (by simply pointing the controller

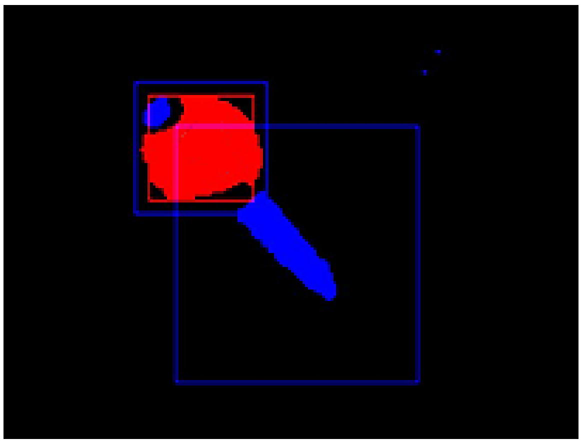


Figure 20: *Color thresholding for object segmentation. The object is a large sphere mounted on a stick.*

at the camera and pressing a button). The approximate size and location in the image are derived from the area and centroid of the segmented pixels (see Figure 21).

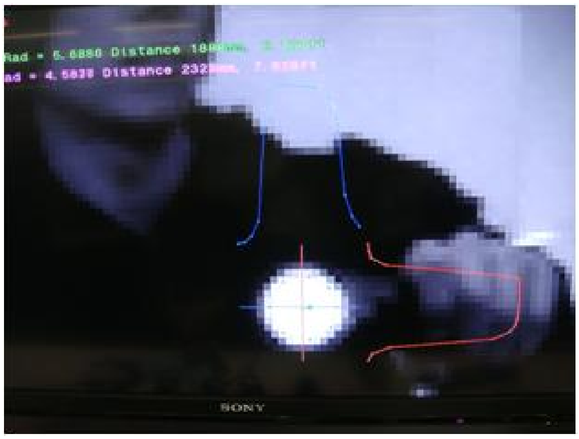


Figure 21: *Visualizing the 2D pixel intensity distribution of the PlayStation Move sphere is useful for debugging the model fitting stage.*

This size and location in the image are used as a starting point for fitting the shape model. To achieve the sub-pixel precision necessary for acceptable 3D position recovery, the shape model is fit to the original image RGB data rather than only the segmented pixels and accounts for camera lens blur. It is well-known that the 2D perspective projection of a sphere is an ellipse [Shivaram and Seetharaman 1998], though many tracking systems introduce significant error by approximating the projection as a circle. In theory, fitting such a model to the image data is straightforward, but in practice many issues arise. Motion of the sphere causes motion blur in the image proportional to the exposure time. Motion also causes the sphere to appear warped due to rolling shutter effects; like most low-cost CMOS webcams, PlayStation Eye uses a rolling electronic shutter. This means every line in the video is imaged at a slightly different time, which leads to stretching or shortening for vertical motions and shear for horizontal motions. Subtle artifacts can also be introduced depending on the method used to convert the raw sensor data to RGB data. Like most low-cost cameras, the PlayStation Eye sensor uses a Bayer pattern such that only R,G, or B is actually measured at every pixel, and RGB must be inferred from surrounding pixels (a process known as Bayer-pattern demosaicking, for which many approaches exist [Gunturk et al. 2005]). Another major concern is partial occlusion, which generally causes large errors in a global fit. To address this, a 2D ray-casting approach is used to identify areas of partial occlusion and remove them from the fit.

Because the size of the sphere and the focal length of the PlayStation Eye are known, the 3D position of the sphere relative to the camera can be computed directly from the 2D ellipse parameters. The output of the image analysis stage is a timestamp and a measurement of the 3D camera-relative position for each sphere, updated at the framerate of the camera (typically 60Hz).

Sensor Fusion. The results of the image analysis are combined with the inertial sensor data using a modified unscented Kalman filter. The details of this powerful state estimation technique are beyond the scope of these notes, but there are many excellent explanations available [Crassidis and Markley 2003][Julier and Uhlmann 1997][Wan and Van Der Merwe 2002]. Though the sensors all contribute to the final state in a complex manner, each has a fundamental contribution that is necessary for the complete state computation. For example, the camera tracking provides an absolute measure of the 3D position. When the controller is not moving, the accelerometer provides the direction of gravity, which gives an absolute measure of the pitch and roll angles. In addition, when the orientation is known, gravity can be "subtracted" from the accelerometer data to recover the controller acceleration. The acceleration is part of the state, and it can also be used to reduce noise in the 3D position and to derive the 3D velocity. The gyroscope data is also crucial because it directly provides angular velocity. When integrated, this provides a responsive measure of 3D rotation (relative orientation) and can be used to derive angular acceleration. The remaining unknown, absolute yaw, is the most tricky, but it can be inferred by comparing the motion direction computed from body-relative inertial measurements to the motion direction computed from camera-relative image measurements.

Again, though theoretically straightforward, in practice there are many issues for sensor fusion. Because each controller, the PlayStation Eye, and the PS3 are driven by independent clocks, each piece of sensor data must be carefully timestamped so that it can be combined with other sensor data properly. Inertial sensor biases must be computed dynamically as they vary, most often due to changes in temperature. When the sphere is temporarily occluded or beyond the camera view, the state must be updated despite the lack of direct position measurements. And wireless data dropouts, though infrequent, must be hidden by extrapolating the state based on previous inertial data.

4.2 Getting Started and Software Tools

4.2.1 Nintendo Wii Remote

The Wiimote connects to a Wii game console or computer wirelessly through Bluetooth. When you press both the 1 and 2 buttons simultaneously or press the red Sync button in the battery case, the Wiimote's LEDs blink, indicating it's in discovery mode. Make sure your computer has a Bluetooth adapter and proper drivers. Currently, each OS and library handles the connection process differently.

PCs. Here are the basic steps (for extra assistance, seek online help such as at www.brianpeek.com):

1. Open Bluetooth Devices in the Control Panel.
2. Under the Devices tab, click the Add button.
3. Put the Wiimote into discovery mode (continuously reenter this mode during this process as it times out).

4. Select Next, select the Wiimote to connect to, and select Next again.
5. Don't use a passkey, and select Next, then Finish.

If a failure occurs, repeat this process until the PC connects to the Wiimote. A simple start for a PC to retrieve data from a Wiimote is the WiimoteLib (www.wiimotelib.org).

Macs. The setup needs to run only once, and future Wiimote connections are simpler. To set up the Wiimote, run the Bluetooth Setup Assistant utility and follow these steps:

1. Select the Any Device option.
2. Put the Wiimote into discovery mode.
3. Select the device (Nintendo RVL-CNT-01) and set the passkey options to "Do not use a passkey."
4. Press Continue until the Mac is connected; then quit the Setup Assistant.

To connect in the future, in the Bluetooth section of System Preferences, select the Wiimote entry and set it to connect. Then, put the Wiimote into discovery mode, and the Wiimote will connect. The OS can keep some applications from connecting to the Wiimote. In those cases, disconnect the Wiimote and let the application connect to the Wiimote.

There are a variety software APIs and SDKs for using the Wiimote to develop 3D spatial interfaces. For C# programmers, Varcholiks Bespoke XNA 3DUI Framework [Varcholik et al. 2009] has a lot of functionality and makes it fairly easy to get started (www.bespokesoftware.org). It has many useful tools, and incorporates Brian Peeks Wiimote library (www.brianpeek.com). Note with Bespoke, you can use as many Wiimotes as you want. Bespoke also has a gesture recognition engine which make it easy to train and use gestures. Other Wiimote libraries include WiiCade for Flash developers, GlovePie, and a host of others (simply do a google search for Wiimote and API).

4.2.2 Microsoft Kinect

At the time these notes were written, Microsoft had announced but not yet released their Kinect SDK. By the time of SIGGRAPH 2011, the SDK will be available. Microsoft's SDK is an ideal platform for using the Kinect to created 3D interfaces because it will be fully supported and will take advantage of the software used in XBox 360 game development.

The SDK provides

- Access to the four-element microphone array with acoustic noise and echo cancellation for crystal clear audio.
- Sound source localization for beamforming, which enables the determination of a sounds spatial location, enhancing reliability when integrated with the Microsoft speech recognition API.
- The Kinect depth data, which provides the distance of an object from the Kinect camera, as well as the raw audio and image data.
- Robust skeletal tracking capabilities for determining the body positions of one or two persons moving within the Kinect field of view.
- Documentation for the APIs and a description of the SDK architecture.

- Sample code that demonstrates how to use the functionality in the SDK.

As an alternative to the Kinect SDK from Microsoft, you can use PrimeSense's NITE middleware drivers along with the OpenNI framework. This approach allows developer access to the RGB and depth cameras and provides simple gesture recognition and skeleton tracking. To use the Kinect with this approach perform the follow the steps found on http://www.codeproject.com/Articles/148251/How-to-Successfully-Install-Kinect-on-Windows-Open.aspx

4.2.3 Playstation Move

Sony has released the Move.Me application for the Playstation 3. This application lets the PS3 act as a device server for the Playstation Move system (see Figure 22) . Other applications on the PC can then connect to the server and get Move data to build 3D user interfaces and applications. The API for the Move.Me application is modular, so in can be incorporated into a variety of different graphics packages and game libraries such as XNA and Unity3D.

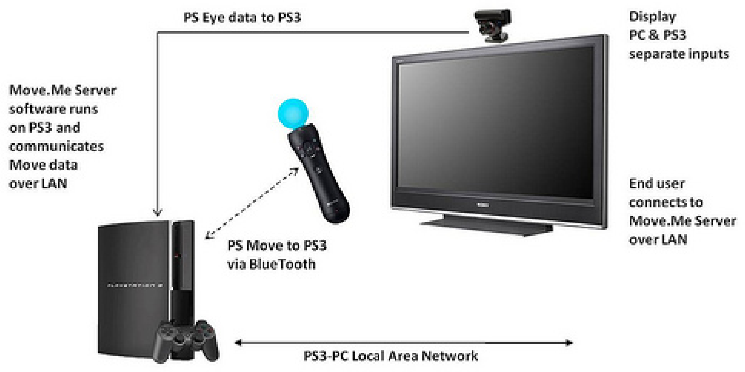

PS Eye data to PS3
Display PC & PS3 separate inputs
Move.Me Server software runs on PS3 and communicates Move data over LAN
End user connects to Move.Me Server over LAN
PS Move to PS3 via BlueTooth
PS3-PC Local Area Network

Figure 22: *The Move.Me application architecture.*

The Move.Me SDK provides a simple interface for developers. The high-level state for each controller state can be queried at any time, and consists of the following information:

- 3D position
- 3D orientation (as a quaternion)
- 3D velocity
- 3D angular velocity
- 3D acceleration
- 3D angular acceleration
- Buttons status
- Tracking status (e.g. visible)

In addition, it allows developers to set the color of the sphere and initiate rumble feedback.

5 Applications 2: Spatial Interfaces in Scientific Visualization

This chapter of the course notes returns to many of the themes and tasks introduced initially in Chapter 2, Common Tasks in 3D User Interfaces. In this chapter, we consider selection and travel (navigation) techniques from the standpoint of scientific visualization applications. In addition to providing some concrete examples of the use of these techniques, scientific visualization applications are unique in that they demand a level of precision and accuracy of 3D interfaces that is not always required of other applications. For example, researchers have extensively studied selection tasks, and the interaction literature documents a variety of selection techniques. However, selecting 3D regions of interest in medical brain-imaging data provides challenges that require us to revisit selection techniques in a new context. Keefe's recent visualization viewpoints article presents additional detail on this and other 3D user interface challenges specific to scientific visualization applications [Keefe 2010].

5.1 3D Interactive Techniques for Scientific Visualization

There is a long history of applications of 3D interactive techniques to scientific visualization. Early success stories include the virtual wind tunnel project [Bryson and Levit 1991] and pioneering work in virtual reality and haptics systems at UNC [Brooks 1988]. More recently, several applications have focused on interacting with anatomical models and fluid flows (e.g., [Sobel et al. 2004; Hentschel et al. 2008] and even in ground-breakingly realistic surgical simulation [Zhang et al. 2010]. Although these and other applications have demonstrated what we believe is a great potential for 3D interaction in science, it seems that 3D user interfaces remain underutilized in science and medicine. This may soon change. In fact, one of our motivations in preparing this course is in recognizing that the emerging hardware and techniques being developed today, primarily for game and entertainment applications, could have a major impact on scientific applications.

5.2 Selection Revisited

When working with complex 3D data, developing accurate visual representations of spatial relationships is critical; intuitively, it makes sense that our input to the computer should match the visual output, also taking advantage of our abilities to process and describe information spatially in three dimensions. The task of 3D selection is one area where utilizing reliable 3D input to complement 3D visualization seems as though it could have a major positive impact on scientific workflows. Earlier, we discussed selection in the context of generic virtual environments, consider now the challenges of selection in 3D or 4D scientific datasets, such as medical imaging data or fluid flow data. One example of these data comes from diffusion tensor MRI (DT-MRI), which can be used to derive dense 3D pathway datasets that correlate with neural fiber tracts in the brain. One important task that is often required when analyzing these data is partitioning 3D pathways into bundles that represent coherent structures in the brain. A scientist might be interested in the bundle of pathways that pass through a particular 3D location in the brain, the set of pathways that connect one region to another, or the set of pathways that belong to a known brain structure. From an interface standpoint, the task is selecting a subset of the visual elements displayed in a 3D scene. The challenge is that these data typically result in some very complex 3D scenes. In particular, the pathways are densely packed within space and many of the pathways follow complex trajectories that vary in curvature and direction as they move through space. These properties of the data make this a particularly challenging case for traditional 3D selection techniques.

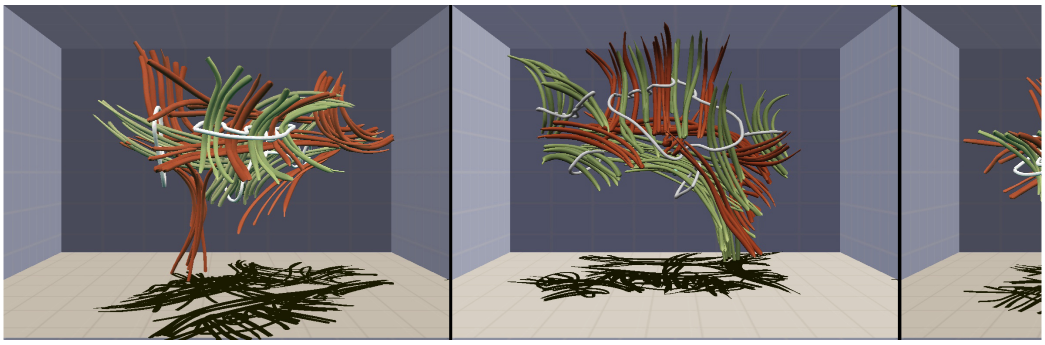


Figure 23: *Three views of the results of a simulated 3D lasso selection task. The red and green curves are representative of the type of 3D structures that are derived from brain imaging data. The user has hand-drawn the white curves, with the goal of drawing lassos that separate the green from the red curves, using a 3D user interface.*

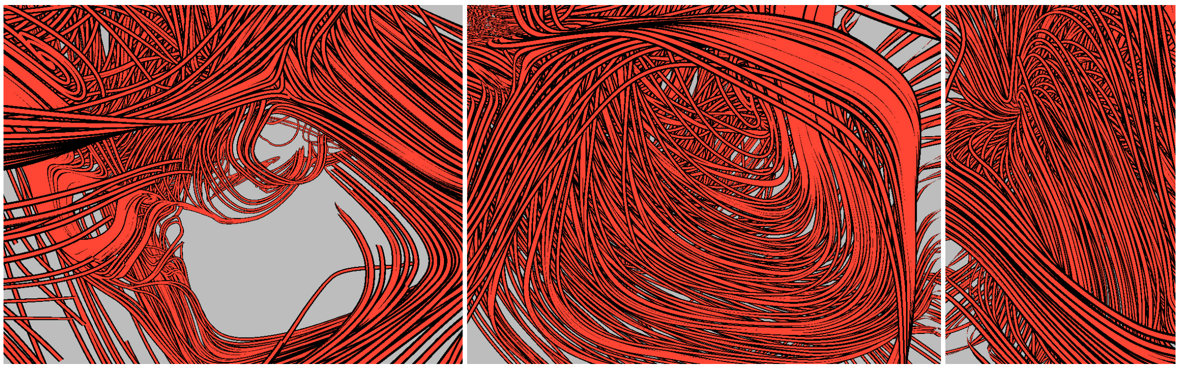


Figure 24: *Three different views of the same simulated 3D fluid flow volume demonstrate the spatial complexity that is typical of scientific datasets, motivating the need for improved spatial interfaces for selection, navigation, and other data analysis tasks (Flow data: Paraview office demonstration dataset, http://www.paraview.org).*

Several researchers have addressed the challenge of working with complex scientific data through approaches that utilize 3D user interfaces. Building on the brain tract selection problem described above and the 3D drawing tools described in the first chapter on applications, Keefe et al. [Keefe et al. 2008b] and Zhou et al. [Zhou et al. 2008] developed techniques wherein 3D lassos are sketched around data features to select them. Figure 23 shows several examples of the types of 3D curves drawn with these interfaces, notice how they twist and turn in space to encompass the features of interest – specifying these 3D curves with a 2D interface would be very difficult to do. An exciting future direction for these 3D user interfaces would be to extend them to include additional data-driven interpretations of user input, such as the query-by-example strategies utilized in the CINCH system [Akers 2006] and other 2D visualization interfaces. The anticipated advantage of 3D user interfaces as demonstrated by the preliminary work described here is that the input from the user is particularly rich – complex 3D curve inputs can be specified directly whereas 2D interfaces would require, for example, multiple 2D inputs from the user to convey similar 3D information.

Analyzing fluid flow data is another extremely important application of visualization that poses similar challenges for 3D selection interfaces, and when a time dimension is added, 4D fluid flow data introduce even more challenges. Figure 24 illustrates the complexity of data that is often encountered. Several viewpoints of a line-based visualization of the same flow vector field are shown in the figure. These data come from the office air flow simulation demonstration dataset distributed with the Paraview tool [Kitware, Inc.]. A variety of different flow structures are seen within the data, even within this small set of views. Analyses of data like these often require researchers to identify features or volumes of interest within the flow and then query underlying data values, for example, pressure, shear stress, and the like. Lasso selection techniques enabled by drawing curves in space have also been applied in this domain [Sobel et al. 2004], but from visualizing fluids to medical imaging data, volumetric selection remains a major challenge in interactive scientific visualization applications. Beyond the lasso-oriented strategies discussed in detail here, we believe that many additional 3D selection strategies designed specifically for scientific visualization applications will be made possible and practical in the near future, given the proliferation of new commodity technologies for 3D input.

5.3 Navigation Revisited

Chapter 2 of the course notes introduced the frequently used virtual reality interaction metaphor of a World in Miniature (WIM). As an example of how 3D user interfaces can be utilized to support navigation in scientific applications, this section describes an extension of the core WIM framework for use in volume visualizations, for example, navigating through 3D medical imaging data. Through this discussion, we aim to illustrate both the potential of applying 3D user interfaces to problems in scientific visualization and also describe how scientific applications often demand a rethinking and refinement of interfaces developed in more general contexts.

Figure 25 shows a recently developed WIM interface called Slice WIM [Coffey et al. 2011]. As the name implies, one of the distinguishing characteristics of this interface relative to previous WIM methods is that it centers on slicing through volume data. Introduced by Stoakley et al.[1995b] and Pausch et al. [1995b], the core WIM concept is to provide a small (e.g. handheld) model of the virtual environment that can act as both a map and an interaction space as the user explores the large-scale environment—essentially a world within a world. Operations that are difficult to perform in the large-scale world (navigating a great distance, selecting/manipulating an object that is far away) can be done easily in the WIM. To this core concept and related interaction techniques, researchers have more recently added features to support scaling and scrolling the WIM for use in very large environments [Wingrave et al. 2006] and automatic selection of optimal WIM views, including handling issues of occlusion, for use in complex architectural models where the 3D structure of the model can be analyzed algorithmically [Trueba et al. 2009].

For volumetric scientific data, issues of occlusion and scale are particularly important, and similarly important is the need to relate 3D representations of the data back to the raw imaging data from which they are derived. For these reasons, the SliceWIM interface pictured in Figure 25 makes heavy use of interactive slicing planes that clip the 3D geometry being visualized so as to provide useful internal views of the dataset. The WIM itself can be rotated to be viewed from any orientation, and the user positions multiple slicing planes relative to the WIM in order to navigate through the dataset. Thus, to support scientific visualization of medical imaging data, this WIM-based interface introduces a reorient-able WIM and a metaphor for working with slicing planes for navigating inside the volume data. A series of new multi-touch interaction techniques also accompany these visuals, and these are described in detail in Chapter 6, which discusses mix and match spatial input, in this case, mixing 2D input from a multi-touch table with 3D interactions in the WIM space.

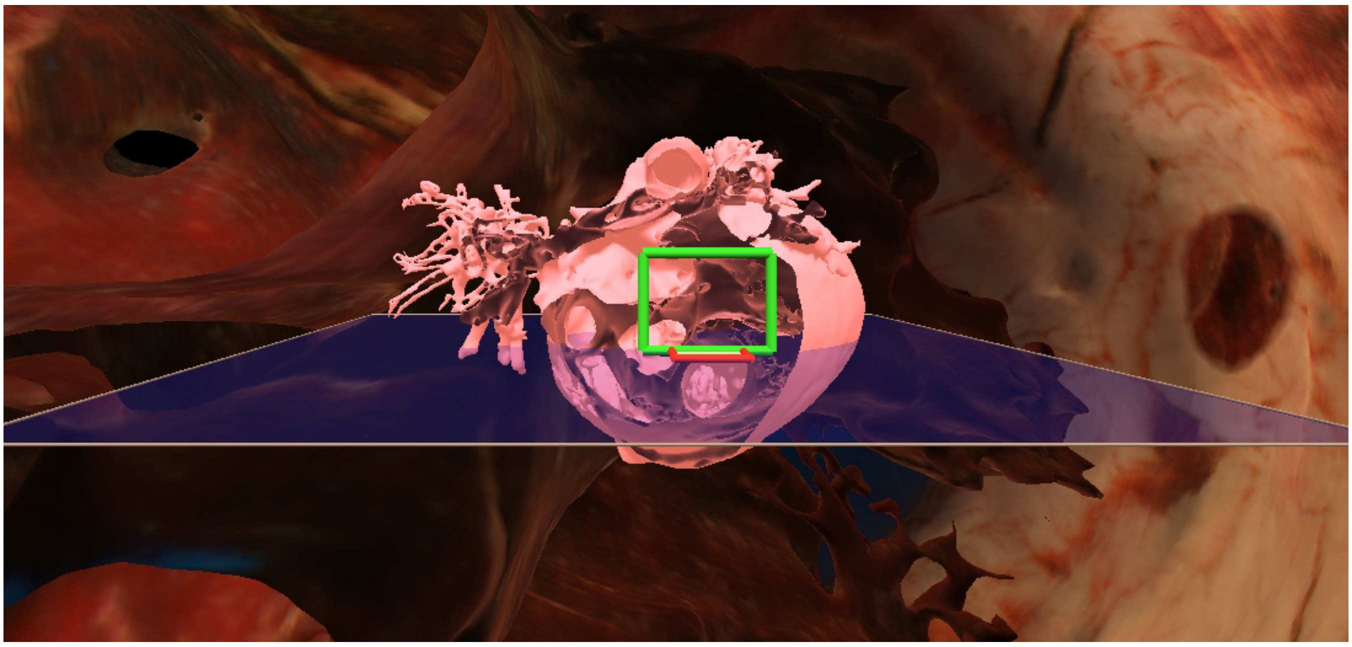


Figure 25: *A screen capture from the Slice WIM tool being used to explore a heart model extracted from imaging data. The pink miniature model in the foreground is an isosurface of the volume data, displayed at a small scale so that the user can understand his current placement within the dataset. Behind the miniature is a detailed (zoomed-in) display of the data that includes texture-mapped tissue information, etc. The WIM serves to facilitate navigation tasks and maintaining a global context while immersed within a small portion of the volume. (Imaging data: National Library of Medicine Visible Human Project http://www.nlm.nih.gov/research/visible/visible_human.html)*

5.4 Does spatial computing for scientific applications require a $1,000,000 lab?

Until recently, each of the applications described earlier would have certainly required a major institutional investment in a virtual reality research lab, complete with 3D tracking and display technology. Now, with the advent of new technologies primarily targeted at game and movie applications, this is changing very rapidly. The scientific community is recognizing this and has held several workshops and special sessions at conferences on the topic of supporting new scientific workflows via emerging, low-cost technologies [Keefe 2008; Coffey et al. 2010]. It is quickly becoming possible for scientists to have in their labs and offices the same type of 3D interface technology that would have required major investments only a few years ago. This makes research in 3D user interfaces for science even more important, and it highlights the importance of research in areas such as accurate, controllable 3D selection and navigation. Although many game technologies also support selection and navigation, the fast-as-you-can style of interaction typically involved in games is very different than the type of interaction needed for science. Ultimately, we believe that one of the exciting aspects of work in this emerging area of applying 3D user interface technology to scientific problems is that the various communities involved can all learn from each other. Scientists clearly want to be able to leverage game technologies in their work, but smart redesign and algorithm development will be needed. Game developers and researchers will undoubtably benefit from learning about scientific applications and their requirements as these can open up new creative uses of emerging technologies and lead to the development of novel, robust interface techniques and algorithms that could be widely applicable.

6 3D Gesture Recognition Techniques

Recognizing 3D gestures has had a long history in the virtual reality and 3D user interface communities. The motion controllers we have discussed thus far can all be used to detect 3D gestures for different applications. We divide gesture recognition into heuristic-based and machine-learning approaches. With heuristic-based approaches, the designer manually identifies the heuristics that classify a particular gesture and differentiate it from other gestures that the system recognizes. Although this manual approach can be time consuming and tedious, especially as the set of recognized gestures grows, it's also highly understandable. Understandability is useful when new gestures are added to the gesture set and must be differentiated from past gestures. With machine learning approaches, gestures can be recognized by finding useful features in the input data stream. These features define a feature vector that is used as part of a machine learning algorithm. Typically, a set of gestures are collected to train the machine learning algorithm, which is used to match the feature vector to the set of possible gestures, differentiating between them.

In this section, we first examine some heuristic-based approaches to gesture recognition. Then, we examine two commonly used machine learning algorithms,a linear classifier and an algorithm using the AdaBoost framework, and conduct experiments to examine their accuracy [Hoffman et al. 2010]. There are, of course, a variety of different machine learning algorithms that could be used to recognize 3D gesture and the reader is encouraged to examine [Duda et al. 2001]. Finally, we discuss using these gestures in a practical setting by exploring user reactions and experiences while playing a video game. We make use of Wiimote data to explore these approaches, but data from any of the devices we have discussed in this course could be used. The only difference would be in how the features are calculated.

6.1 Interpreting Wiimote Data

Here we review two basic types of Wiimote data interpretation, integration and recognition through heuristics.

6.1.1 Integration

Because acceleration is the rate of velocity change, which is the rate of positional change, you can theoretically integrate the Wiimote acceleration data to find velocity and reintegrate it to find the Wiimote's actual position change. You should also be able to integrate the gyroscope's angular velocity to achieve absolute orientation. However, this approach has three significant limitations [Giansanti et al. 2003]. First, acceleration jitter can lead to significant positional deviations during a calculation. So, you can ignore accelerations below a threshold because they're most likely the result of jitter. Alternatively, a smoothing function can help reduce jitter. Second, you must remove the gravity vector from the reported acceleration before computing velocity. If the sensor bar or a gyroscope is available, you can possibly infer the Wiimote's orientation and realize the gravity vector as a 1-g downward force; otherwise, you must determine the gravity vector on the fly. One way to compute the gravity vector is by watching the derivative of the acceleration (or jerk data). When this is close to zero (that is, no acceleration change is computed) and the reported acceleration magnitude is close to 1 g, thereby eliminating cases of user-induced constant acceleration, you can assume the Wiimote is reporting only the gravity vector. Subtracting this vector from future accelerations results in the acceleration being directly attributable to user action, assuming the Wiimote maintains its orientation. Third, orientation must be accurate; even slight errors produce large positional errors after movement. For example, one meter of travel after a five-degree orientation change (well within the variance of the hand) results in an error of nearly 9 cm. A larger 30-degree orientation change results in an error of 50 cm. Additionally, unaccounted-for orientation changes can give very wrong results. For example, raising the Wiimote a foot, inverting it, and lowering it to its original location will result in no actual positional change, but as computed will result in a two-foot upward movement in the Wiimote's FOR. You can address this only by using the SBC or a gyroscope, because the Wiimote's accelerometers can't easily provide a gravity vector while the Wiimote is moving, as we discussed earlier.

In work with a locomotion interface for American football [Williamson et al. 2010], double integration let users control the application using a Wiimote and have their body movements maneuver the quarterback. To achieve this, a Wiimote was attached to the center of the user's chest, which was close to the body's center of mass. Although this improved the reported acceleration data, it broke the SBC. The Wiimote acceleration data was then passed through an exponential smoothing filter:

$$\vec{a}_{current} = \alpha\vec{a}_i + (1-\alpha)\vec{a}_{i-1} \tag{17}$$

where $\alpha = 0.9$. An alternate approach is to use a Kalman filter, but this requires more computation [LaViola 2003]. Finally the double-integration step was performed on the smoothed acceleration data. Owing to these three steps, the Wiimote was responsive, and the user seemed to move relatively accurately for that application.

To further evaluate the control's accuracy, tests were performed in which the user moved from a starting location and then back. These tests showed little error in position over short time periods (5 to 10 seconds), but only when users maneuvered in an unnaturally upright and stiff fashion.

6.1.2 Recognition through Heuristics

The Wiimote provides a raw data stream, and you can use heuristics to interpret and classify the data. Whether you use heuristics on their own or as features for gesture recognition, their higher-level meaning is more useful for 3DUI design and implementation. In this section, we discuss several heuristic-based approaches for interpreting Wiimote data used in two prototype game applications. One Man Band used a Wiimote to simulate the movements necessary to control the rhythm and pitch of several musical instruments [Bott et al. 2009]. Careful attention to the Wiimote data enabled seamless transitions between all musical instruments. RealDance explored spatial 3D interaction for dance-based gaming and instruction [Charbonneau et al. 2009]. By wearing Wiimotes on the wrists and ankles, players followed an on-screen avatar's choreography and had their movements evaluated on the basis of correctness and timing.

Poses and underway intervals. A pose is a length of time during which the Wiimote isn't changing position. Poses can be useful for identifying held positions in dance, during games, or possibly even in yoga. An underway interval is a length of time during which the Wiimote is moving but not accelerating. Underway intervals can help identify smooth movements and differentiate between, say, strumming on a guitar and beating on a drum.

Because neither poses nor underway intervals have an acceleration component, you can't differentiate them by accelerometer data alone. To differentiate the two, an SBC can provide an FOR to identify whether the Wiimote has velocity. Alternatively, you can use context, tracking Wiimote accelerations over time to gauge whether the device is moving or stopped. This approach can be error prone, but you can successfully use it until you reestablish the SBC.

Poses and underway intervals have three components. First, the time span is the duration in which the user maintains a pose or an underway interval. Second, the gravity vector's orientation helps verify that the user is holding the Wiimote at the intended orientation. Of course, unless you use an SBC or a gyroscope, the Wiimote's yaw won't be reliably comparable. Third, the allowed variance is the threshold value for the amount of Wiimote acceleration allowed in the heuristic before rejecting the pose or underway interval.

In RealDance, poses were important for recognizing certain dance movements. For a pose, the user was supposed to stand still in a specific posture beginning at time t_0 and lasting until $t_0 + N$, where N is a specified number of beats. So, a player's score could be represented as the percentage of the time interval during which the user successfully maintained the correct posture.

Impulse motions. An impulse motion is characterized by a rapid change in acceleration, easily measured by the Wiimote's accelerometers. A good example is a tennis or golf club swing in which the Wiimote motion accelerates through an arc or a punching motion, which contains a unidirectional acceleration.

An impulse motion has two components, which designers can tune for their use. First, the time span of the impulse motion specifies the window over which the impulse is occurring. Shorter time spans increase the interaction speed, but larger time spans are more easily separable from background jitter. The second component is the maximum magnitude reached. This is the acceleration bound that must be reached during the time span in order for the Wiimote to recognize the impulse motion.

You can also characterize impulse motions by their direction. The acceleration into a punch is basically a straight impulse motion, a tennis swing has an angular acceleration component, and a golf swing has both angular acceleration and even increasing acceleration during the follow-through when the elbow bends. All three of these impulse motions, however, are indistinguishable to the Wiimote, which doesn't easily sense these orientation changes. For example, the punch has an acceleration vector along a single axis, as does the tennis swing as it roughly changes its orientation as the swing progresses. You can differentiate the motions only by using an SBC or a gyroscope or by assuming that the Wiimote orientation doesn't change.

RealDance used impulse motions to identify punches. A punch was characterized by a rapid deceleration occurring when the arm was fully extended. In a rhythm game, this instant should line up with a strong beat in the music. An impulse motion was scored by considering a one-beat interval centered on the expected beat. For the Wiimote corresponding to the relevant limb, the time sample in the time span corresponding to the maximal acceleration in the Wiimote's longitudinal axis was selected. If this maximal acceleration was below a threshold, no punch occurred, and the score was zero. Otherwise, the score was computed from the distance to the expected beat. If the gesture involved multiple limbs, the maximal acceleration value had to be greater than the threshold for all Wiimotes involved. The average of all individual limb scores served as the gesture's overall score.

Impact events. An impact event is an immediate halt to the Wiimote due to a collision, characterized by an easily identifiable acceleration bursting across all three dimensions. Examples of this event include the user tapping the Wiimote on a table or a dropped Wiimote hitting the floor. To identify an impact event, compute the change in acceleration (jerk) vectors for each pair of adjacent time samples. Here, t_k corresponds to the largest magnitude of jerk:

$$t_k = \underset{T}{\operatorname{argmax}} \|\vec{a}_t - \vec{a}_{t-1}\|. \tag{18}$$

If the magnitude is larger than a threshold value, an impact has occurred. RealDance used impact motions to identify stomps. If the interval surrounding a dance move had a maximal jerk value less than a threshold, no impact occurred, and the score was zero. Otherwise, the score was calculated the same way as for an impulse motion. One Man Band also used impact events to identify when a Nintendo Nunchuk controller and Wiimote collided, which is how users played hand cymbals.

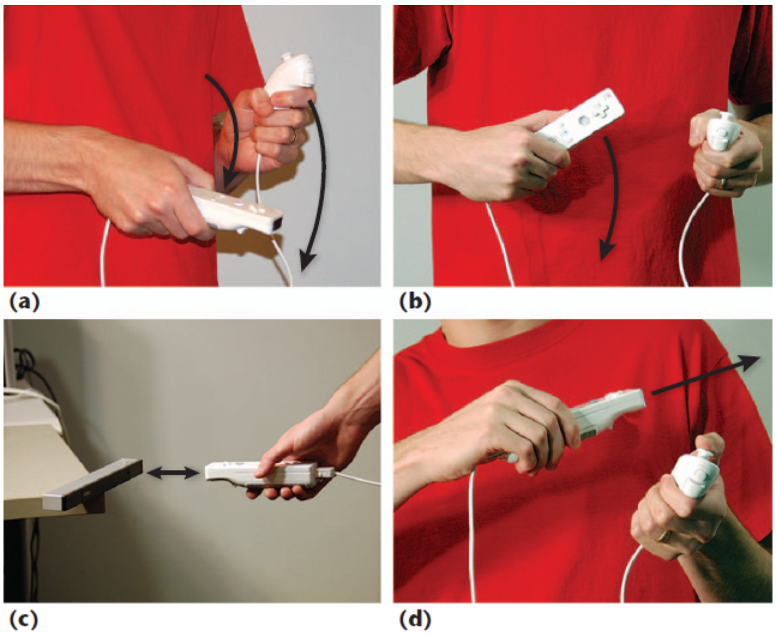


Figure 26: *One Man Band differentiated between multiple Wiimote gestures using mostly simple modal differentiations for (a) drums, (b) guitar, (c) violin, and (d) theremin. To the player, changing instruments only required orienting the Wiimote to match how an instrument would be played.*

Modal differentiation. You can use easily recognized modes in an interface to differentiate between functionality. For example, a button's semantics can change as the Wiimote changes orientation, or the Wiimote's pitch might differentiate between a system's states. Although modes can lead to errors when users are unaware of them, constant user action such as holding a button or pose can lead to quasimodal states that are easily understood and useful. This is important because the Wiimote has several attachments, such as the Nunchuk, and multiple buttons that you can use to create quasimodes. In One Man Band, the multi-instrument musical interface (MIMI) differentiated between five different instruments by implementing modal differences based on the Wiimote's orientation. Figure 26 shows four of these. If the user held the Wiimote on its side and to the left, as if playing a guitar, the application interpreted impulse motions as strumming motions. If the user held the Wiimote to the left, as if playing a violin, the application interpreted the impulse motions as violin sounds. To achieve this, the MIMI's modal-differentiation approach used a normalization step on the accelerometer data to identify the most prominent orientation:

$$\vec{a}_{norm} = \frac{\vec{a}}{||\vec{a}||} \tag{19}$$

followed by two exponential smoothing functions (see Equation 17). The first function, with an $\alpha = 0.1$, removed jitter and identified drumming and strumming motions. The second function, with an $\alpha =$

0.5, removed jitter and identified short, sharp gestures such as violin strokes. The MIMI also used the Nunchuk's thumbstick to differentiate between the bass and guitar.

6.2 Linear Classifier

The linear classifier we discuss in this section is based on Rubine's gesture recognition algorithm [Rubine 1991]. Given a feature vector $\vec{f}$, associated with a given 3D gesture g in a gesture alphabet C, a linear evaluation function is derived over the features. The linear evaluation function is given as

$$g_c = w_{c0} + \sum_{i=1}^{F} w_{ci} f_i \tag{20}$$

where $0 \leq c < C$, F is the number of features, and w_{ci} are the weights for each feature associated with each gesture in C. The classification of the correct gesture g is the c that maximizes g_c.

Training of this classifier is done by finding the weights w_{ci} from the gesture samples. First, a feature vector mean $\vec{\bar{f}_c}$ is calculated using

$$\bar{f}_{ci} = \frac{1}{E_c} \sum_{e=0}^{E_c-1} f_{cei} \tag{21}$$

where f_{cei} is the i^{th} feature of the e^{th} example of gesture c and $0 \leq e < E_c$ where E_c is the number of training samples for gesture c. The sample covariance matrix for gesture c is

$$\Sigma_{cij} = \sum_{e=0}^{E_c-1} (f_{cei} - \bar{f}_{ci})(f_{cej} - \bar{f}_{cj}). \tag{22}$$

The Σ_{cij} are averaged to create a common covariance matrix

$$\Sigma_{ij} = \frac{\sum_{c=0}^{C-1} \Sigma_{cij}}{-C + \sum_{c=0}^{C-1} E_c}. \tag{23}$$

The inversion of Σij then lets us calculate the appropriate weights for the linear evaluation functions,

$$w_{cj} = \sum_{i=1}^{F} (\Sigma^{-1})_{ij} \bar{f}_{ci} \tag{24}$$

and

$$w_{c0} = -\frac{1}{2} \sum_{i=1}^{F} w_{ci} \bar{f}_{ci} \tag{25}$$

where $1 \leq j \leq F$.

6.3 AdaBoost Classifier

The AdaBoost algorithm is based on LaViola's pairwise AdaBoost classifier [LaViola and Zeleznik 2007]. AdaBoost [Schapire 1999] takes a series of weak or base classifiers and calls them repeatedly in a series of rounds on training data to generate a sequence of weak hypotheses. Each weak hypothesis has a weight associated with it that is updated after each round, based on its performance on the training set. A separate set of weights are used to bias the training set so that the importance of incorrectly classified examples are increased. Thus, the weak learners can focus on them in successive rounds. A linear combination of the weak hypotheses and their weights are used to make a strong hypothesis for classification.

More formally, for each unique 3D gesture pair, our algorithm takes as input training set $(\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)$, where each $\vec{x}_i$, represents a feature vector containing J features. Each y_i labels $\vec{x}_i$ using label set $Y = \{-1, 1\}$, and m is the total number of training samples. Since we are using a pairwise approach, our algorithm needs to train all unique pairs of gestures. For each unique pair, the AdaBoost algorithm is called on a set of weak learners, one for each feature discussed in Section 6.4. We chose this approach because we found, based on empirical observation, that our features can discriminate between different 3D gesture pairs effectively. We wanted the features to be the weak learners rather than having the weak learners act on the features themselves. Thus, each weak learner C_j uses the j^{th} element in the $\vec{x}_i$ training samples, which is noted by $\vec{x}_i(j)$ for $1 \leq j \leq J$.

6.3.1 Weak Learner Formulation

We use weak learners that employ a simple weighted distance metric, breaking $(\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)$ into two parts corresponding to the training samples for each gesture in the gesture pair. Assuming the training samples are consecutive for each gesture, we separate $(\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)$ into $(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)$ and $(\vec{x}_{n+1}, y_{n+1}), ..., (\vec{x}_m, y_m)$ and define $D^1(i)$ for $i = 1, ..., n$ and $D^2(i)$ for $i = n + 1, ..., m$ to be training weights for each gesture. Note that in our formulation, D^1 and D^2 are the training weights calculated in the AdaBoost algorithm (see Section 6.3.2).

For each weak learner C_j in each feature vector $\vec{x}_i(j)$ in the training set, the weighted averages are then calculated as

$$\mu_{j1} = \frac{\sum_{k=1}^{n} x_k(j) D^1(k)}{\sum_{l=1}^{n} D^1(l)} \tag{26}$$

and

$$\mu_{j2} = \frac{\sum_{k=n+1}^{m} x_k(j) D^2(k)}{\sum_{l=n+1}^{m} D^2(l)}. \tag{27}$$

These averages are used to generate the weak hypotheses used in the AdaBoost training algorithm. If a given feature value for a candidate gesture is closer to μ_{j1}, the candidate is labeled as a 1, otherwise the candidate is labeled as a -1. If the feature value is an equal distance away from μ_{j1} and μ_{j2}, we simply choose to label the gesture as a 1. Note that it is possible for the results of a particular weak classifier to obtain less than 50% accuracy. If this occurs the weak learner is reversed so that the first gesture receives a -1 and second gesture receives a 1. This reversal lets us use the weak learner's output to the fullest extent.

6.3.2 AdaBoost Algorithm

For each round $t = 1, ..., T * J$ where T is the number of iterations over the J weak learners, the algorithm generates a weak hypothesis $h_t : X \rightarrow \{-1, 1\}$ from weak learner C_j and the training weights $D_t(i)$ where $j = mod(t-1, J) + 1$ and $i = 1, ..., m$. This formulation lets us iterate over the J weak learners and still conform to the AdaBoost framework [Schapire 1999]. Indeed, the AdaBoost formulation allows us to select weak classifiers from different families at different iterations.

Initially, $D_t(i)$ are set equally to $\frac{1}{m}$, where m is the number of training examples for the gesture pair. However, with each iteration the training weights of incorrectly classified examples are increased so the weak learners can focus on them. The strength of a weak hypothesis is measured by its error

$$\epsilon_t = Pr_{i \sim D_t}[h_t(\vec{x}_i(j)) \neq y_i] = \sum_{i:h_t(\vec{x}_i(j)) \neq y_i} D_t(i). \tag{28}$$

Given a weak hypothesis, the algorithm measures its importance using the parameter

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right). \tag{29}$$

With α_t, the distribution D_t is updated using the rule

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(\vec{x}_i(j)))}{Z_t} \tag{30}$$

where Z_t is a normalization factor ensuring that D_{t+1} is a probability distribution. This rule increases the weight of samples misclassified by h_t so that subsequent weak learners will focus on more difficult samples. Once the algorithm has gone through $T * J$ rounds, a final hypothesis

$$H(x) = \text{sgn} \left(\sum_{t=1}^{T*J} \alpha_t h_t(x) \right) \tag{31}$$

is used to classify gestures where α_t is the weight of the weak learner from round t, and h_t is the weak hypothesis from round t. If $H(x)$ is positive, the new gesture is labeled with the first gesture in the pair and if $H(x)$ is negative it is labeled with the second gesture in the pair.

These strong hypotheses are computed for each pairwise recognizer with the labels and strong hypothesis scores tabulated. To combine the results from each strong hypothesis we use the approach suggested by Friedman [Friedman 1996]; the correct classification for the new gesture is simply the one that wins the most pairwise comparisons. If there is a tie, then the raw scores from the strong hypotheses are used and the one of greatest absolute value breaks the tie.

6.4 Feature Set

The features used in the linear and AdaBoost classifier on Rubine's feature set [Rubine 1991]. Since Rubine's features were designed for 2D gestures using the mouse or stylus, they were extended to work for 3D gestures. The Wiimote and Wii MotionPlus sense linear acceleration and angular velocity respectively. Although we could have derived acceleration and angular velocity-specific features for this study, we chose to make an underlying assumption to treat the acceleration and angular velocity data as position information in 3D space. This assumption made it easy to adapt Rubine's feature set to the 3D domain and the derivative information from the Wiimote and Wii MotionPlus.

The first feature in the set, quantifies the total duration of the gesture in milliseconds, followed by features for the maximum, minimum, mean and median values of x,y and z. Next we analyzed the coordinates in 2D space by using the sine and cosine of the starting angle in the XY and the sine of the starting angle in the XZ plane as features. Then the feature set included features with the sine and cosine of the angle from the first to last points in the XY and the sine of the angle from the first to last points in the XZ plane. After that, features for the total angle traversed in the XY and XZ planes, plus the absolute value and squared value of that angle, completed the features set which analyzes a planar surface. Finally, the length of the diagonal of the bounding volume, the Euclidian distance between the first and last points, the total distance traveled by the gesture and the maximum acceleration squared fulfill Rubine's list.

Initially, this feature set was used for both the Wiimote and the Wii MotionPlus[1], totaling 58 features used in the two classifiers. However, after some initial pilot runs, a singular common covariance matrices was forming with the linear classifier. A singular common covariance matrix cause the matrix inverse needed to find the weights for the linear evaluation functions impossible. These singular matrices were formed when trying to use the feature set with the Wii MotionPlus attachment. Because of this problem, the MotionPlus feature set was culled to use only the minimum and maximum x, y, and z values, the mean x, y, and z, values, and the median x, y, and z values. Thus, 29 features were used when running the experiments with the Wiimote and 41 features were used when running experiments with the Wiimote coupled with the Wii MotionPlus.

6.5 Gesture Set

The majority of the work on 3D gesture recognition with accelerometer and gyroscope-based input devices contain experiments using only four to 10 unique gestures [Kratz et al. 2007; Rehm et al. 2008; Schlömer et al. 2008]. In order to better understand how many gestures these types of devices can accurately recognize and how many training samples would be needed to do so, we chose to more than double that set and use 25 gestures. The gestures, depicted graphically in Figure 27, are performed by holding the Wiimote in various orientations. For a majority of the gestures, the orientation and rotation are the same for both left and right handed users. The only exceptions are the gestures Tennis Swing, Golf Swing, Parry, and Lasso. The gestures Tennis Swing and Golf Swing are performed on either the left or right side of the user, corresponding to the hand that holds the Wiimote. For the Parry gesture, a left-handed person will move the Wiimote towards the right, while a right-handed person will move the Wiimote towards the left. Finally, the Lasso gesture requires a left- and right-handed user to rotate in opposite directions, clockwise versus counterclockwise, respectively.

This gesture set was developed by examining existing video games that make use of 3D spatial interaction,

[1]The Wii MotionPlus used maximum angular velocity squared instead of maximum acceleration squared.

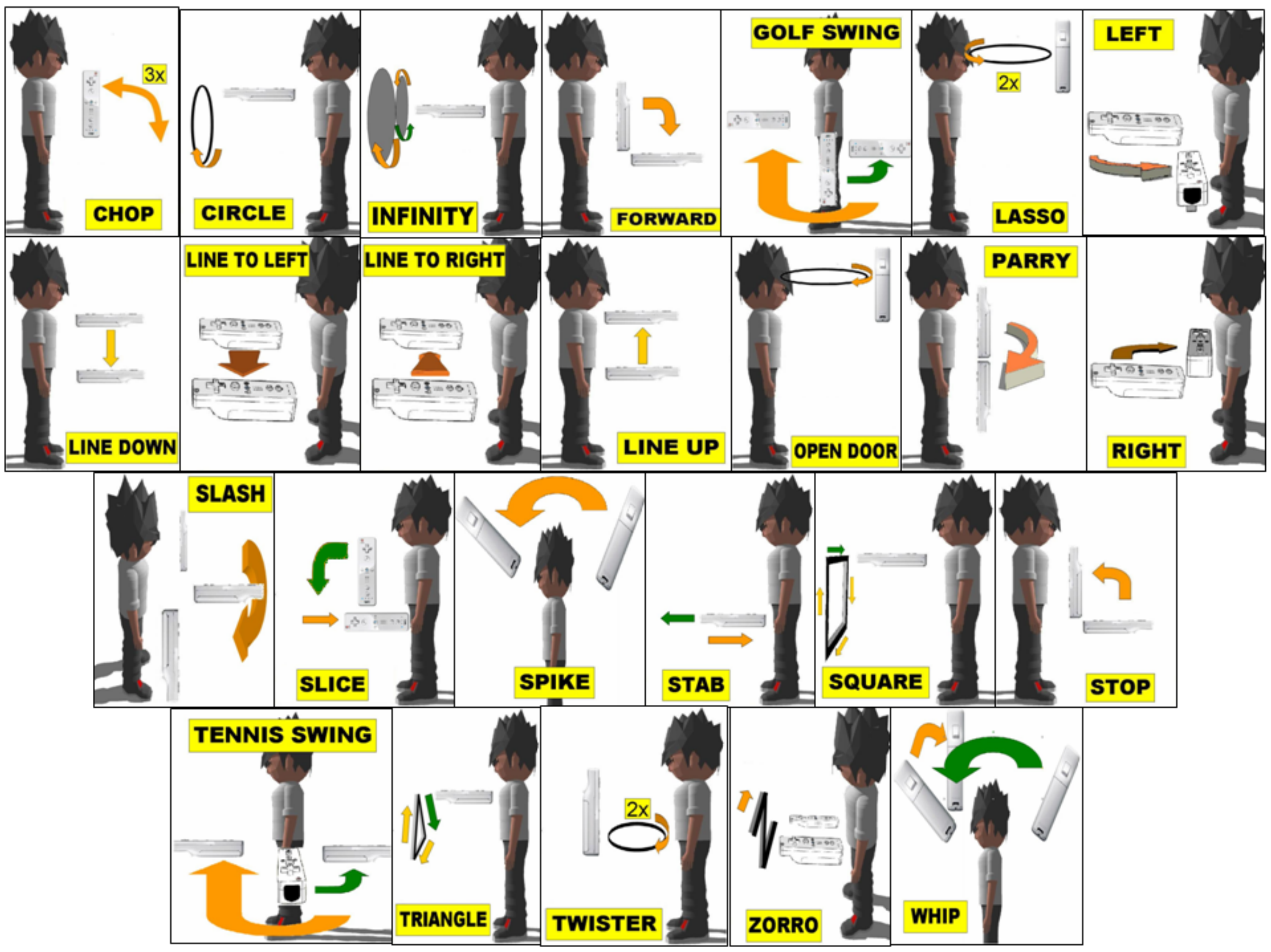


Figure 27: *Illustration showing the 25 gestures used in our study performed with a Wiimote. For compound movements the green arrows show the initial movement and the orange arrows show the remaining movements.*

specifically from the Nintendo Wii gaming console. We examined a variety of different games from different genres including sports games, first person shooters, fighting games, and cooking games. Although the gesture set is game specific, it is general enough to work in a wide variety of virtual and augmented reality applications. Initially, we focused on simple movements such as line to the right and line to the left which could be applied to various actions. From there, slightly more complex gestures were added which involved closed figures such as the square, triangle and circle. With this group, we add user variations in velocity, duration and distance traversed. Finally the last set of maneuvers allowed for more freedom in body movement in an effort to help disambiguate gestures during feature analysis. Examples of these gestures include golf swing, whip and lasso.

6.6 3D Gesture Data Collection

Machine learning algorithms such as the linear and AdaBoost classifiers need data to train on. In order for the algorithms to learn and then recognize gestures, a gesture database was created to use in both training and testing. We recruited 17 participants (4 female and 13 male) from the University of Central Florida, of which four men were left handed, to provide the gesture samples. Each participant had experience using the Nintendo Wii gaming console, with two participants doing so on a weekly basis.

Several steps were taken to ensure that participants provided good gestures samples. First, they were given a brief overview of the Wiimote gesture data collection interface and a demonstration of how to interact with the application. After the demonstration, participants were presented with an introduction screen asking them to choose what hand they would hold the Wiimote with to perform the gestures. This decision also told the application to present either left or right-handed gesture demonstration videos. Next, an orientation window (see Figure 28) was displayed to help participants learn how to perform the gestures before data collection began. Each of the 25 gestures are randomly listed on the left hand side of the window to reduce the order effect due to fatigue. Participants moved through the list of gestures using the up and down buttons on the Wiimote. At any time, participants could view a video demonstrating how to perform a gesture by pressing the Wiimote's right button. These videos demonstrated how to hold the Wiimote in the proper orientation in addition to showing how to actually perform the motion of the gesture. Before the gesture collection experiment began, an orientation phase required participants to perform and commit one sample for each gesture. This reduced the order effect due to the learning curve. To perform a gesture, participants pressed and held down the Wiimote's "B" button to make a gesture. After participants released the "B" button, they could either commit the gesture by pressing the Wiimote's "A" button or redo the gesture by pressing and holding the "B" button and performing the gesture again. This feature enabled participants to redo gesture samples they were not happy with. Once a gesture sample is committed to the database, the system pauses for two seconds, preventing the start of an additional sample. This delay between samples allows participants to reset the Wiimote position for the next gesture sample. In addition, the delay prevents the user from trying to game the system by quickly performing the same gesture with little to no regard of matching the previous samples of that gesture.

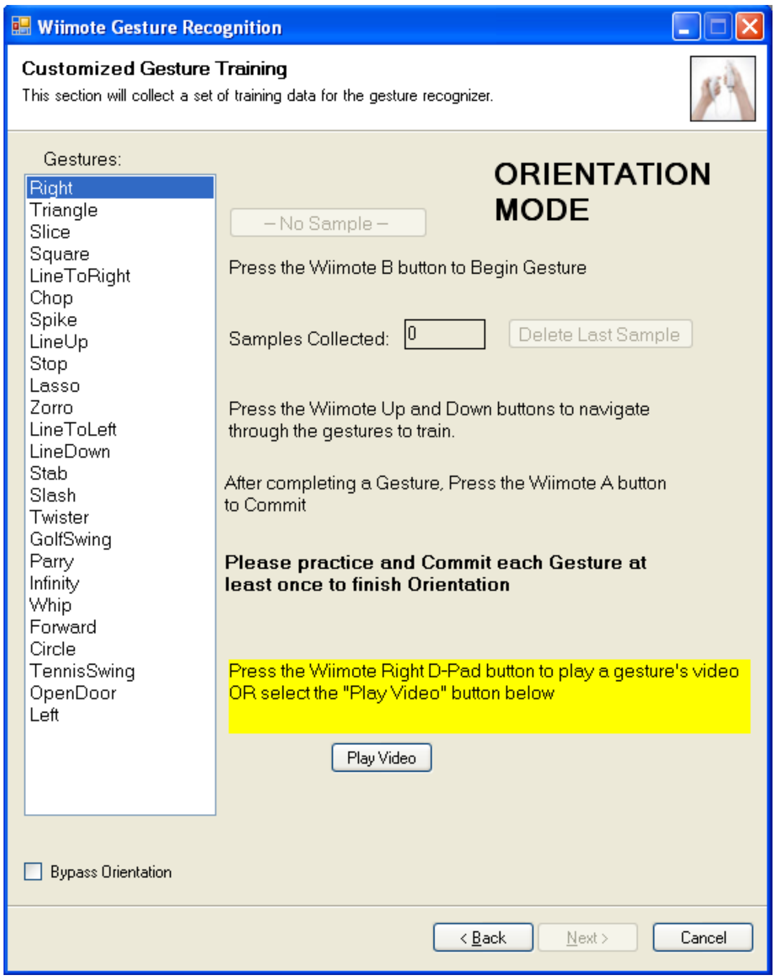
After participants created one sample for each of the 25 gestures, they entered data collection mode. The only difference between orientation mode and data collection mode is the amount of samples which need to be committed. When participants commit a training sample for a particular gesture, a counter is decremented and displayed. A message saying the collection process for a given gesture is complete is shown after 20 samples have been entered for a gesture. This message indicates to the participant that they can move on to providing data for a remaining gesture. A total of 20 samples for each gesture, or 500 samples in all, is required for a full user training set. In total, 8,500 gesture samples were collected[2]. Each data collection session lasted from 30-45 minutes.

6.7 3D Gesture Recognition Experiments

With the 3D gesture database, we were able to run a series of experiments to examine both learning algorithms in terms of user dependent and user independent recognition. The primary metric analyzed in our experiments is gesture classification accuracy, while having the over-arching goal of maximizing the number of gestures correctly recognized at varying degrees of training the machine learning algorithms. For both the dependent and independent experiments, the linear and AdaBoost classifiers were tested using the Wiimote data only and using the Wiimote in conjunction with the Wii MotionPlus attachment. Thus, the experiments attempted to answer the following questions for both the user dependent and independent cases:

- How many of the 25 gestures can each classifier recognize with accuracy over 90%?

[2]The gesture database can be downloaded at http://www.eecs.ucf.edu/isuelab/downloads.php.


Wiimote Gesture Recognition

Customized Gesture Training

This section will collect a set of training data for the gesture recognizer.

Gestures:

Right
Triangle
Slice
Square
LineToRight
Chop
Spike
LineUp
Stop
Lasso
Zorro
LineToLeft
LineDown
Stab
Slash
Twister
GolfSwing
Parry
Infinity
Whip
Forward
Circle
TennisSwing
OpenDoor
Left

ORIENTATION MODE

– No Sample –

Press the Wiimote B button to Begin Gesture

Samples Collected: 0

Press the Wiimote Up and Down buttons to navigate through the gestures to train.

After completing a Gesture, Press the Wiimote A button to Commit

Please practice and Commit each Gesture at least once to finish Orientation

Press the Wiimote Right D-Pad button to play a gesture's video OR select the "Play Video" button below

Figure 28: *The gesture collection window that each participant interacted with in order to provide 20 samples for each of the the 25 gestures.*

- How many training samples are needed per gesture to achieve over 90% recognition accuracy?
- How much accuracy improvement is gained from using the Wii MotionPlus device?
- Which classifier performs better?
- Which gestures in our gesture set cause recognition accuracy degradation?

6.7.1 User Dependent Recognition Results

For the user dependent recognition experiments, we used a subset of samples for each gesture from a single user to train the machine learning algorithms and the remaining samples to test the recognizers. This approach is equivalent to having a user provide samples to the recognizer up front, so the algorithm can be tailored to that particular user. For each user dependent test, two categories of experiments were created: classification over all 25 gestures and finding the maximum recognition accuracy rate over as many gestures as possible. Both categories were then broken into three experiments providing 5, 10, or 15 training samples per gesture with the remaining samples used for testing accuracy. Each experiment was executed on all four of the classifiers mentioned earlier. The experiment set was conducted on each of the

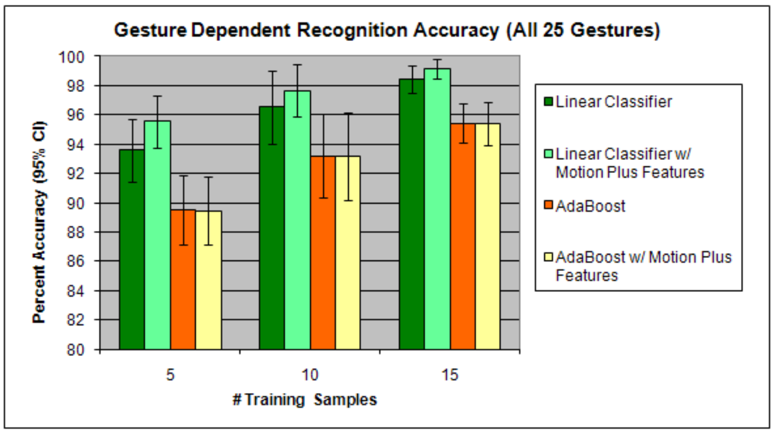


Figure 29: *The average recognition results for the user dependent experiments over all 25 gestures. The results are grouped by the number of training samples (5, 10, or 15) provided for each gesture within an experiment. A single user dependent experiment utilized two classifiers (linear and AdaBoost), each executing with either the Wiimote or Wiimote + MotionPlus input device producing four accuracy values.*

17 participant's data.

Comparison	Test Statistic	P Value
Linear5 - Ada5	$t_{16} = 7.17$	$p < 0.01$
LinearMP5 - AdaMP5	$t_{16} = 8.36$	$p < 0.01$
Linear10 - Ada10	$t_{16} = 6.48$	$p < 0.01$
LinearMP10 - AdaMP10	$t_{16} = 6.54$	$p < 0.01$
Linear15 - Ada15	$t_{16} = 7.69$	$p < 0.01$
LinearMP15 - AdaMP15	$t_{16} = 7.16$	$p < 0.01$

Table 1: *Results from a set of t-tests showing significance differences between the linear classifier and AdaBoost indicating the linear classifier outperforms AdaBoost in our test cases. Note that under the comparison column, MP stands for whether the Wii MotionPlus was used and the number represents how many samples were used for training the algorithms.*

The results of trying to recognize all 25 gestures in each experiment are shown in Figure 29. We analyzed this data using a 3 way repeated measures ANOVA and found significance for the number of training samples ($F_{2,15} = 17.47, p < 0.01$), the classification algorithm ($F_{1,16} = 119.42, p < 0.01$), and the use of the Wii MotionPlus data ($F_{1,16} = 8.23, p < 0.05$). To further analyze the data, pairwise t tests were run. The most notable observation is the high level of accuracy across all experiments. In particular, the linear classifier gave a mean accuracy value of 93.6% using only 5 training samples for each gesture and 98.5% using 15 training samples per gesture ($t_{16} = -5.78, p < 0.01$). Furthermore, the linear classifier outperformed AdaBoost in every situation by at least 3% (see Table 1 for the statistical results). Another important result shown in Figure 29 is the role of the Wii MotionPlus in both recognition algorithms. The Wii MotionPlus significantly increased the recognition accuracy for the linear classifier to 95.5% using 5 training samples ($t_{16} = -2.81, p < 0.05$) per gesture and 99.2% using 15 training samples per gesture ($t_{16} = -2.54, p < 0.05$). For AdaBoost, the change in accuracy was negligible.

While these accuracy levels are good for some applications, it is important to know which gestures would need to be removed from the classification set in order to improve recognition results. To begin this

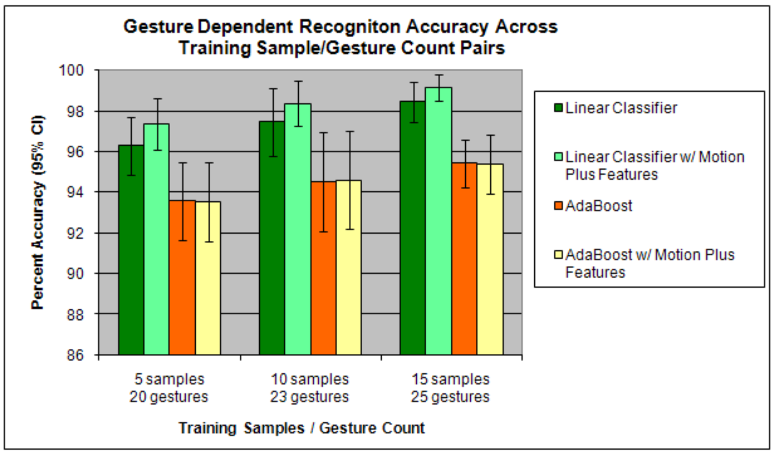


Figure 30: *The average recognition results for the user dependent experiments over a varying number of gestures. The goal of this experiment was to recognize, with high accuracy, as many gestures as possible with different levels of training. The results are grouped by the number of training samples (5, 10, or 15) provided for each gesture within an experiment. A single user dependent experiment utilized two classifiers (linear and AdaBoost), each executing with either the Wiimote or Wiimote + MotionPlus input device producing four accuracy values.*

examination, the set of gestures frequently recognized incorrectly from the previous three experiment categories were removed. Within this set we noticed a few gestures with similar attributes such as Tennis Swing and Golf Swing or Square and Triangle. These gesture pairs involve similar movement which caused the classifiers to misidentify each type as the other. Once the poorly classified gestures were extracted the accuracy results easily increased to above 98%. We systematically added each of the removed gestures back into the gesture set on a case by case basis, leaving one gesture from each similar pairing out in order to improve the recognition on the other included gesture from each pair. The results are shown in Figure 30. As with the previous experiment, we ran a 3 way repeated measures ANOVA as well as t tests and found significant results for the training sample/number of gestures pairs ($F_{2,15} = 5.01, p < 0.05$), the classification algorithm ($F_{1,16} = 48.55, p < 0.01$), and the use of the Wii MotionPlus data ($F_{1,16} = 9.69, p < 0.01$).

In the experiment using 5 training samples, the gestures Forward, Golf Swing, Spike, Triangle and Line to Left were removed, thereby producing over 93.5% accuracy for AdaBoost and over 96.3% for the linear classifier. Once the number of training samples was increased to 10, the set of removed gestures included only Spike and Triangle. This experiment yielded higher accuracy with the linear classifier ($t_{16} = -1.90, p = 0.075$) and AdaBoost ($t_{16} = -1.07, p = 0.3$), but these results were not significant. Finally, since the recognition rates for 15 training samples were already greater than 95%, no gestures were removed during this last test. These results show that recognition accuracy rates as high as 97% can be achieved for 20 gestures using only 5 samples per gesture and 98% for 23 gestures using 10 samples per gesture with the Wiimote coupled with the Wii MotionPlus attachment. In fact, for the linear classifier, the recognizer obtained significantly higher accuracy using the Wii MotionPlus attachment in the 5 training sample/20 gesture case ($t_{16} = -2.32, p < 0.05$) and the 10 training sample/23 gesture case ($t_{16} = -2.18, p < 0.05$).

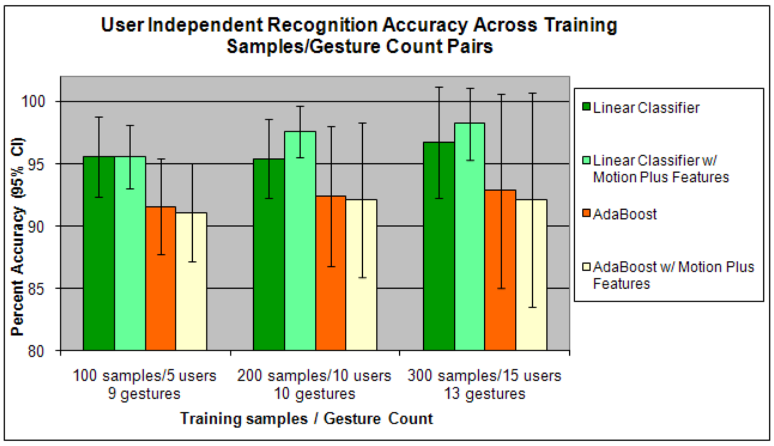


Figure 31: *The average recognition results for the user independent experiments over a varying number of gestures. The goal of this experiment was to recognize, with high accuracy, as many gestures as possible when given different levels of training. The results are grouped by the number of user training samples (100,200, or 300) per gesture used for training within an experiment. These numbers are analogous to using 5, 10, and 15 users' data for training. A single user independent experiment utilized two classifiers (linear and AdaBoost), each executing with either the Wiimote or Wiimote + MotionPlus input device producing four accuracy values.*

6.7.2 User Independent Recognition Results

For the user independent recognition experiments, a subset of the 17 user study gesture data files was used for training. From the remaining files, recognition is performed and reported on a per user basis. This approach is equivalent to having the recognizers trained on a gesture database and having new users simply walk up and use the system. The benefit of this approach is there is no pre-defined training needed for each user at a potential cost in recognition accuracy. We ran three tests in this experiment, using data from 5, 10, and 15 users from our gesture database for training with the remaining user data for testing. As in the user dependent study, each scenario was executed on all four of the classifiers mentioned earlier. Note that for the independent recognition results, we did not perform any statistical analysis on the data because we did not have an equal number of samples for each condition and the sample size for the 15 user case as two small (only two test samples). An approach to dealing with this issue is to perform cross validation on the data so proper statistical comparisons can be made.

The results, shown in Figure 31, shows that the linear classifier outperforms AdaBoost by at least 3% and using the Wii MotionPlus attachment improved recognition for only the linear classifier. However, unlike in the user dependent tests, the Wii MotionPlus slightly hindered AdaBoost accuracy. After removing the poorly classified gestures, we followed the reintroduction method we used for the user dependent tests. The linear classifier was able to recognize a total of 9, 10, and 13 gestures with mean accuracy values of 95.6%, 97.6%, and 98.3% respectively using the Wiimote coupled with the Wii MotionPlus attachment. The gestures enabled for the 5 user training set included Forward, Stop, Open Door, Parry, Chop, Circle, Line to Right, Line Up and Stab. When using the 10 user training set, the Twister and Square gestures were added while maintaining slightly higher accuracy levels. On the other hand, the gesture Open Door was removed because the newly introduced training data increased confusion among samples of that type. Finally, for the 15 user training set, the gestures Open Door (again), Infinity, Zorro and Line Down were added but the Twister gesture was removed.

6.8 Accuracy Discussion

From these experiments, we can see that 25 3D gestures can be recognized using the Wiimote coupled with the Wii MotionPlus attachment at over 99% accuracy in the user dependent case using 15 training samples per gesture. This result significantly improves upon the results in the existing literature in terms of the total number of gestures that can be accurately recognized using a spatially convenient input device. There is, of course, a tradeoff between accuracy and the amount of time needed to enter training samples in the user dependent case. 15 training samples per gesture might be too time consuming for a particular application. As an alternative, the results for 5 training samples per gesture only shows a small accuracy degradation. For the user independent case, we can see the accuracy improves and the number of gestures that can reliably be recognized also increases as the number of training samples increases. Although the overall recognition accuracy and the number of gestures was higher in the dependent case (as expected), the independent recognizer still provides strong accuracy results.

Comparing the two classifiers in the experiments shows the linear classifier consistently outperforms the AdaBoost classifier. This is somewhat counterintuitive given the AdaBoost classifier is a more sophisticated technique. However, as we discussed in Section 6.4, the linear classifier can suffer from the singular matrix problem which can limit its utility when new features that could improve accuracy are added.

With the Wii MotionPlus, we also tried to adjust for gyroscopic drift in data. Our attempt at calibration involved setting the Wiimote coupled with the MotionPlus device on a table with the buttons down for approximately 10 seconds, followed by storing a snap shot of the roll, pitch and yaw values. That snapshot was then used as a point of reference for future collection points. However, using those offsets caused decreased accuracy in AdaBoost and singular matrices in the linear classifier leading to the use of raw gyroscope data instead.

6.9 Using 3D Gestures in a Practical Setting

Testing heuristic-based approaches and gesture recognition algorithms on a restricted set of data will provide some insight into how effective the accuracy can be. However, these experiments are often considered to provide an upper bound on performance, given ideal conditions. Thus it is important to understand performance during real situations. In this case, performance is not just accuracy, but also player experience.

We have designed and developed a video game prototype, Wizard of Wii (See Figure 32), a linear, first person adventure game with its gameplay based entirely on 3D gestures performed with a Nintendo Wiimote. Players are placed in the role of a wizard on an adventure and are required to complete a series of four quests in order to win. The player interacts with the world using 3D gestures to cast spells, wield a sword, and teleport between quests. The player moves automatically from the start to finish in a linear fashion. Gameplay tasks are presented to players at predefined points, with visual cues to alert players that they are required to perform a gesture. Each visual cue, in the form of an image, also tells players which gesture to perform and how to perform it. Limiting players' freedom to move and perform gestures enabled us to predict when a gesture was to be performed and subsequently determine if it was recognized correctly.

For the game, we use the gestures shown in Figure 27. In-game gestures are performed by holding a Nintendo Wiimote in different starting orientations and making a motion in the air. Some gestures have a clear mapping to sword manipulation. These include 'Parry', 'Slash', 'Slice', 'Stab', 'Zorro', and 'Chop'. Additionally, gestures with simple flicking motions were adapted to vertical and horizontal sword cuts

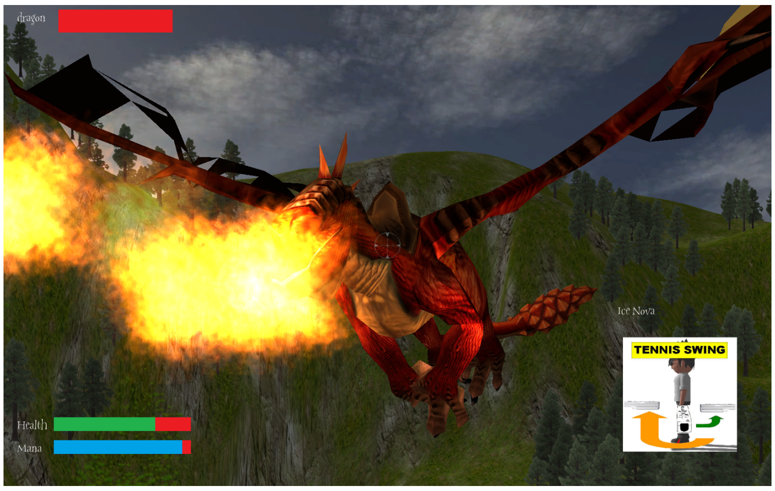


Figure 32: *One of the quests in Wizard of Wii.*

('Left', 'Right', 'Forward', and 'Stop'). Gestures depicting lines (Line to 'Left , Right, Up, Down') are very easy to perform and were adapted to allow the player to teleport between quests. The 'Square' gesture was mapped into drawing a virtual window in the world, where players could see the quest information. All remaining gestures were mapped to spells.

At each point where players were expected to perform a gesture, we limited the player to a maximum of 5 mistakes. This feature was introduced after a pilot run of the experiment, during which one person was unable to proceed in the game due to poor recognition accuracy. Also, before each quest in the game, players were required to summon a quest description by use of the 'Square' gesture. This screen displayed the storyline for the quest and described the quest tasks in terms of gestures to be performed by the player. The quest description screen also served as a practice area where players could recall and practice the appropriate gestures before starting the quest. To make it easier to remember gestures, no more than seven gestures were required for each quest.

6.9.1 Usability Study

Subjects and Apparatus. A total of 25 participants (23 male and 2 female) were recruited from the University of Central Florida for participation in the investigation. Participants' ages were between 18 and 28 years. 22 participants were right-handed while three were left-handed. Each participant took 60-90 minutes to complete the entire experiment and was paid $10 for his/her time.

The experiment was conducted on a computer equipped with an Intel Core-i7-920 processor, and an nVidia GeForce 460 graphics adapter. For display, we used a 50in Samsung DLP 3D HDTV with the resolution set to 1680x1050 pixels and a refresh rate of 60Hz. The participants stood approximately 4-6 ft from the display and performed gestures while they were standing.

Experiment Procedure. The experiment was split up into two parts. Participants first performed each gesture 25 times, a random 15 of which were used for training the in-game recognizer in each session. To prevent fatigue during the data collection phase, participants were asked to take frequent breaks. Participants could delete gesture data if they discovered they had done gestures incorrectly. Additionally, a proctor monitored participants at all times and pointed out mistakes, if any. The data collection session lasted 30-40 minutes.

During the second part of the experiment, participants were required to play Wizard of Wii. Two gameplay session were required from each participant. Each session lasted approximately 10-15 minutes with a few minutes break between sessions. After each session, participants were asked to fill out a questionnaire. After the first gameplay session, the questionnaire included a few extra questions because it asked participants to compare their performance between sessions and also included a few free-response questions seeking suggestions from participants.

Gameplay Metrics. Five metrics were analyzed in each gameplay session. Actual recognition accuracy and the number of recognition errors were automatically logged during each player's session. A recognition error denotes an instance where the expected gesture does not match the recognized gesture. This occurs in two ways. Either a player performed a gesture incorrectly or the recognizer failed to recognize a correctly performed gesture. Given our experimental setup, it is impossible to distinguish the two cases. We attempted to minimize the first possibility by using visual cues to aid gesture recall and by limiting the number of gestures in each quest, but it may not have eliminated all cases of player error due to incorrect recall. Actual recognition accuracy in each round was computed as the ratio of gestures correctly recognized in the first attempt to total gestures performed. Three other metrics were collected via player responses in gameplay questionnaires: perceived recognition accuracy, perceived gameplay performance and ability to recall gestures. A 7-point Likert scale was used for these metrics.

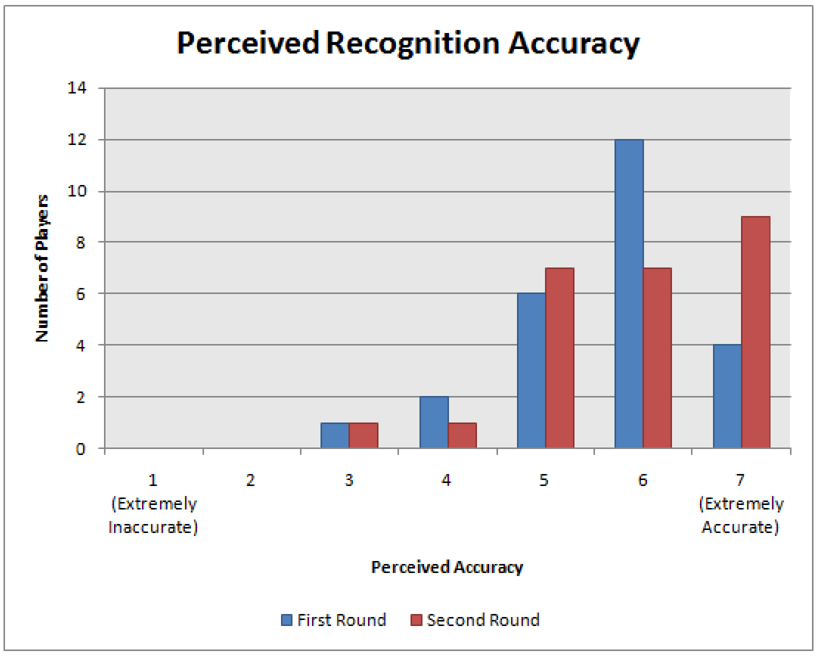


Figure 33: *Frequency Table showing how the players' perception of recognition accuracy changed between the two gameplay sessions.*

6.9.2 Results

We performed pairwise Wilcoxon Signed Rank tests on gameplay metrics, in order to see if the change in each metric's distribution was significant between gameplay sessions. In the first session, mean recognition accuracy across all players was 69.33%, which improved to 79.2% in the second session ($z = -3.664, p < 0.01$). The mean number of recognition errors per player decreased from 19.8 to 14.24 between the two gameplay sessions ($z = -2.649, p < 0.01$).

Perception of Recognition Accuracy. Figure 33 shows the change in players' perception of gesture recognition accuracy between gameplay sessions. The mean perceived recognition accuracy changed only slightly from 5.64 to 5.88. Although the increase is not statistically significant ($z = -1.255, p = 0.210$), Figure 33 indicates an increase in the number of players reporting higher perceived accuracy in the second session. It is interesting to note that although actual recognition accuracy increased significantly between sessions, the increase in mean perceived accuracy was not significant, indicating that most players' perception of recognition accuracy varied little from their initial impressions formulated during the first session.

Perception of Gameplay Performance. The mean perceived gameplay performance increased from 5.20 to 5.92 between sessions ($z = -2.578, p < 0.05$). A deeper look at the decrease in recognition errors also revealed that players needed fewer attempts in the second session before a gesture was recognized correctly. This may explain why players perceived better gameplay performance during the second session, even though they were not able to perceive a significant increase in recognition accuracy.

A majority of players reported that their gameplay performance had improved between sessions. When asked about the impact of recognition accuracy on gameplay performance, responses varied from player to player. A common complaint was the frustration players felt when the recognizer failed to correctly recognize gestures, resulting in distraction and decreased immersion for a small number of players. Some reported that good recognition accuracy increased their immersion in the game and they actually felt as if they were wielding a sword or a wand. Interestingly, a few players reported that the accuracy of recognition did not impact their gameplay performance at all.

Perception of Gesture Recall. There was a significant increase in player's perception of their ability to recall gestures in the second session. The mean response changed from 5.52 to 6 between sessions ($z = -2.178, p < 0.05$). This finding is in agreement with earlier results that indicate that gesture recognition accuracy improved between sessions while both the number of recognition errors and attempts for correct recognition decreased.

Gesture Preferences. After each session, players were asked which gestures were easy or difficult to recall, and which gestures were suitably adapted to the game environment. Analysis of the responses indicates a few common themes. Most participants felt comfortable using simple and smooth motions, e.g. wrist flicks ('Left', 'Right', 'Down', 'Stop') and lines ('Line To Left, Right, Up, Down'). Some participants disliked gestures describing geometric patterns or shapes ('Circle','Open Door', 'Lasso', 'Twister', 'Triangle', 'Square' and 'Figure 8'). Several players reported confusion and frustration with gestures involving similar motions. Examples of such gestures include 'Lasso', 'Circle', 'Open Door', 'Twister', all involving one or more circles. Other gestures such as 'Triangle', 'Square' or 'Figure 8' were disliked by some participants on the grounds that geometric patterns didn't suit the fantasy setting of Wizard of Wii and therefore detracted from the experience.

The majority of participants reported they would like to see 3D gestures used in video games if recognition errors could be eliminated. When asked which games could benefit from having 3D gestures, most of the participants responded with similar answers: First Person Games, Adventure Games, Role Playing Games, Fighting Games, and Sports Games.

6.9.3 Discussion

We were able to achieve a maximum recognition accuracy of 79.2%, which is far below the expected accuracy of over 95% shown in Section 6.7, which was achieved over gestures collected in a data gathering environment. The decrease in recognition accuracy can be attributed to stress induced by the game environment, which may impair gesture recall and also introduce more variation in the gestures themselves. Our results indicate that players were able to better recall gestures with repeated play and also perceived an improvement in their gameplay performance. With repeated play, players' gestures were correctly recognized in fewer attempts. This may have contributed to a perception of higher gameplay performance in the second session. Gesture recognition accuracy improved significantly between gameplay sessions, possibly due to improved gesture recall. Interestingly, *Players were unable to perceive the increase in recognition accuracy*, indicating that first impressions were fundamental to their perception of recognition accuracy. This is an important finding from a game designer's perspective because it seems to imply that games using 3D spatial input should attempt to make a good initial impression in terms of recognition accuracy.

7 Future 2: Mix and Match Spatial Input

One of the key themes that we believe will emerge in the future of spatial interfaces is a mixing of many of the techniques, technologies, and applications that we have discussed in this course. Haptics might be combined with low-cost depth cameras, tangible props might be combined with multi-touch input, scientific applications might be combined with art and games. In this chapter of the course notes, we highlight some recent research activities that suggest these trends and discuss some of the open research questions in supporting mix and match spatial input.

7.1 Spatial input meets multi-touch

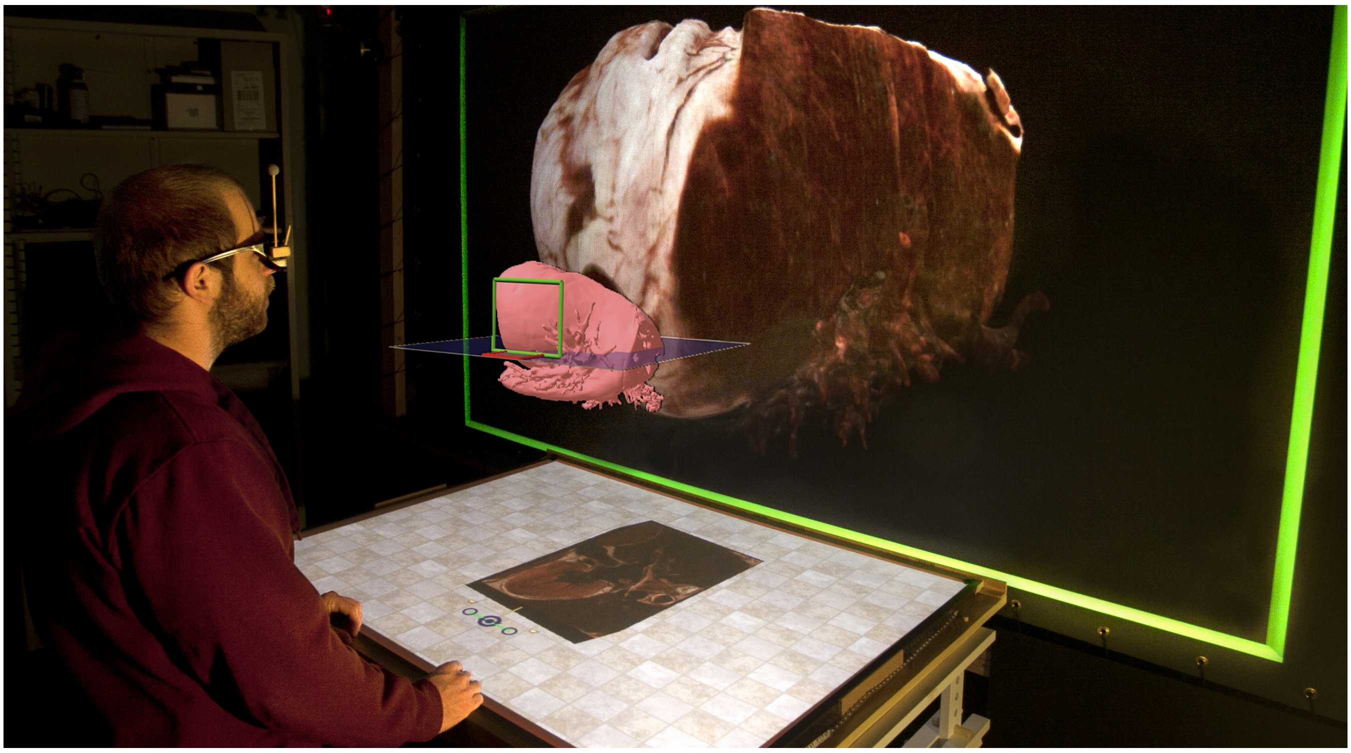


Figure 34: *The Slice WIM interface mixes 2D and 3D hardware and interaction techniques. The vertical screen displays a stereoscopic view, and using the stereo projection, the small world-in-miniature (WIM) model is positioned so that it appears to float in the air above the table (digitally superimposed on the photograph here to demonstrate the effect). The table is a multi-touch display surface. The user performs actions to manipulate the 3D scene by touching the shadow of the WIM on the table. In this way, multiple simultaneous 2D inputs on the table surface are used to control a variety of 3D visualization techniques, including navigation and selection within the volume data.*

Earlier in the notes we discussed Slice WIM, an extension to WIM interfaces to support navigating through volumetric data. Figure 34 illustrates the interface for this WIM technique, which combines multi-touch input with head-tracked stereoscopic projection. The interface is based on the metaphor of interacting with the shadow of a 3D object that floats in the air above a table. Multi-touch gestures made on the table are used to manipulate the shadow, which in turn causes the 3D object to update (translate, rotate, etc.) to maintain a consistent connection with the shadow. A series of widgets on the table are used to control slicing through the data (e.g., setting vertical cutting planes), which is an important technique for visualizing dense volume data. All of the interactions are 3D actions in the sense that the user's view is stereoscopic and the entire environment is 3D, but in this case, the input all occurs on a plane, thus, it can be viewed as a 2D input – albeit a very rich 2D input since multiple simultaneous touches are recorded by the table device.

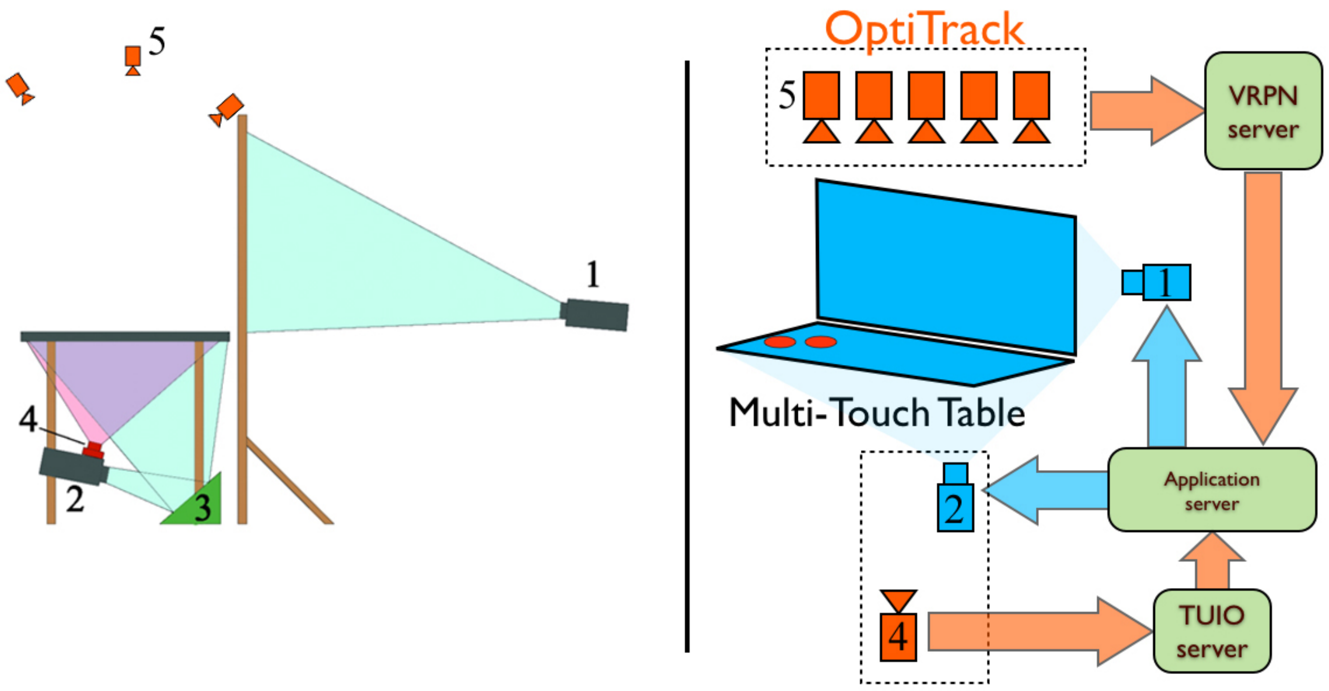


Figure 35: *Currently, custom hardware configurations are often required to perform the type of mix and match spatial input that we believe will become commonplace in the future. This left side of the figure describes the hardware configuration utilized for the Slice WIM interface shown in the previous figure: The rear-projected image on the vertical screen originates from a 3D projector* **(1)** *mounted behind the screen. The horizontal table's image originates from a projector* **(2)** *mounted under the table and must be bounced off a front surface mirror* **(3)** *to reach the surface. An IR camera* **(4)** *is mounted on top of the table's projector and is used to track points of contact on the table. A set of NaturalPoint OptiTrack cameras* **(5)** *are used to provide headtracking data to update the stereoscopic display. The right side of the figure provides some additional system information, including the use of VRPN and TUIO protocols for handling data from the various devices.*

In mixing 2D and 3D and multi-touch on a surface with stereoscopic imagery, this system is an example of the type of mixing of interface and graphics technologies that we believe will become increasingly common in the future. Today, these systems must typically be constructed via custom hardware configurations, such as the one diagrammed in Figure 35. In fact this display can be built for relatively low cost, at least given what we have become used to in the virtual reality community. The following describes some notes on materials and construction to build a similar FTIR multi-touch table and stereoscopic wall display (adapted from [Coffey et al. 2010]).

7.1.1 Building the Minnesota Interactive Visualization Lab 3D Multi-touch Workbench

Parts Listing:

- Table hardware, including projector
  - Viewsonic PJD5351 projector
  - Unibrain Fire-i BW board camera with 107° horizontal FOV lens
  - Acrylic sheet (4' x 3' x 3/8")
  - Reel with 5 m adhesive-backed ribbon of surface-mounted 850 nm IR LEDs
  - Power supply for LEDs
  - Rubber spacers for LEDs (4" setting block, cut as necessary)
  - Aluminum channel (2 pieces 3/8" wide, 3/4" deep, 98" long each)
  - Drafting film (two 3' x 4' sheets)
  - Clear silicone sealant and Xylene or comparable solvent
  - First surface mirror (408 x 608 mm, 4-6 wave)
  - Schott RG830 filter (1" diameter)
  - Adjustable table legs
- Stereoscopic wall hardware
  - DepthQ DQ-WXGA stereoscopic projector
  - Synchronization device and shutter glasses
  - Rosco Grey (9' x 4.5' sheet)
  - Wood beams for screen frame
  - Bungee cord (1/8" diameter) and grommets (1/2" inner diameter)
- Computer workstations/servers
  - Community Core Vision and OptiTrack servers (Any desktop machine will suffice)
  - 3D application workstation (with stereo capable graphics card)
- Head-tracking system
  - Optitrack camera server (any desktop machine will suffice)
  - Optitrack NaturalPoint 6 camera system

Design and construction notes:

The NUI Group forums (e.g., [Tinkerman 2008]) are an excellent source of information for developing these types of systems, and provided much guidance in designing the multi-touch system documented here. Some specific construction notes adapted from the forum, include: (1) It is strongly advised to prototype the placement of the interactive surface projector, the camera, and the mirror using paper triangles (cut out to scale) that closely estimate beam shapes and focal ranges. The paper triangles can be folded to the required mirror size and angle for the projector beam. (2) Acrylic sheet edges may need to be polished. (3) A metal frame can be used to house the IR LEDs used for the FTIR multi-touch table. The LED's must be placed inside the channel formed by this frame; using a strip of pre-mounted LEDs greatly facilitates assembly. (4) To provide an opaque projection surface on top of the table, a piece of laminated drafting film is placed on top of the table surface. To improve the capture of touch events through this medium, a medium thickness mixture of clear silicone sealant and xylene is painted onto the bottom side of the laminated drafting film in two coats. The NUI group forum refers to this as the compliant surface, and provides several reviews of different options for creating an effective touch and display surface.

Today, we believe the design discussed here is one of the best options for mixing touch input with 3D stereoscopic displays; however, we expect this will continue to change in the future. As 3D TV displays continue to improve, it's likely that one or both of the projectors used in the system described here may be able to be replaced by a commodity TV. The current design does not utilize a 3D TV because we found that when stereoscopic images were presented so that the virtual objects appeared far in front of the display, the ghosting on the TV displays was too distracting, ruining the stereo effect.

7.1.2 Future Outlook

As 3D input and graphics technologies become increasingly available to the public at reasonable cost, we believe configurations such as this one, which combine touch with 3D or other technologies will become increasingly common, and perhaps hold a key to enabling emerging technologies to be used effectively in applications to science, art, design, medicine, and other areas to complement increasingly compelling games and entertainment.

7.2 Tangible spatial input

Today, 3D interfaces that include a sense of touch tend to be limited to full active haptic simulations (e.g., utilizing devices similar to the SensAble PHANTOM) or passive interfaces, such as interacting with a tablet prop within a virtual environment. In the future, we believe a number of other hybrid styles of tangible spatial interaction will emerge. Already today, we see multi-touch interfaces being implemented on cubes and curved 3D surfaces, and there appear to be many new opportunities on the horizon for mixing spatial input with touch and other tangible styles of interaction.

One area where we see some early work in this direction is in interacting with physical rapid prototype models. For example, Kruszyski et al. [Kruszynski and van Liere 2009] developed a smartly calibrated pen interface to interact with high-resolution 3D rapid prototypes of sea corals. This work extended early work in prop-based interaction (e.g., [Hinckley et al. 1997]) to demonstrate the current potential impact on scientific workflows of designing 3D interfaces for interacting with very accurate rapid prototype 3D printouts. Other researchers have continued in this direction, investigating how 3D gestures made relative to accurate, scientifically relevant 3D printouts can be used as effective visualization interfaces [Konchada et al. 2011; Jackson and Keefe 2011].

Building on the idea of interacting in the 3D space surrounding an object or a display, the style of tangible 3D interactions that we might come to expect in the future may be closely related to research results, such as the work by Grossman et al., a 3D gestural interface for bimanual interaction with emerging true 3D display technologies that render a true 3D light model inside a hemispheric dome [Grossman et al. 2004].

Emerging optical devices, such as the kinect, enable a variety of freehand 3D interactions, but we know from years on research in the 3D user interaction community that this type of freehand unconstrained interaction can often be very difficult to control, much more so than techniques guided by even passive haptic feedback. We expect that a critical challenge for the future will therefore be to determine how best to leverage the dramatic new capabilities for 3D input that we have with the advent of commercially-available devices, such as the kinect, while combining these capabilities with the advantages of tangible spatial input. In the research community, we have seen many successful examples of physical input devices; the cubic mouse [Fröhlich and Plate 2000] for example. How can we combine the success of these tangible spatial input devices with the exciting emerging trends in commercially available hardware, and how will this impact the many new applications that are becoming possible? We believe these are key questions that will define the future of spatial interaction.

7.3 Cognitive and perceptual issues

The final emerging theme for future investigations of spatial interfaces that we will highlight here is the importance of taking cognitive and perceptual issues into account during interface design. In emerging 3D environments, there is great potential for illusion, and many researchers have found that this can be both good and bad. Some of the most compelling examples of how perceptual illusion can be used to advantage include work in using redirected walking [Razzaque 2005] and portals [Steinicke et al. 2009] to move through virtual environments.

Studies of touch-based stereoscopic 3D environments have recently explored the perceptual issue of touching objects that exist the the space in front of or behind a surface display [Valkov et al. 2011]. Similarly, the impact of the realism of the shadow widgets used in the interface in Figure 34 has been explored and unrealistic shadows have been shown to often be advantageous [Coffey et al. 2011].

These are just a few examples of how human perception can impact user interface design. As interfaces move increasingly in the direction of mixing 2D and 3D inputs, working in tandem with 3D displays, and mixing tangible and freehand modes of action, perceptual and cognitive issues are sure to become increasingly important factors to consider in interface design. The exciting possibility is that, as some preliminary work has already demonstrated, it is often the case that perceptual illusion can actually be used to advantage in designing effective spatial user interfaces.

Acknowledgements

A special thanks to Doug Bowman, Ernst Kruijff, Ivan Poupyrev, Chad Wingrave, Richard Marks, Salman Cheema, Kris Rivera, and members of the Interactive Systems and User Experience Lab for assisting in the development of the material for this course. Also thanks to co-authors, collaborators, and students whose work in scientific visualization, art, and 3D user interfaces features prominently in many examples within the course notes: Dane Coffey, Nick Malbraaten, Fedor Korsakov, David Laidlaw, Fritz Drury, Robert Zeleznik, Andy Forsberg, Jason Sobel, and the research groups in the University of Minnesota Interactive Visualization Lab and the Brown University Visualization Research Lab. The preparation of this course was supported in part by the NSF (CAREER award, IIS-1054783).

References

AKERS, D. 2006. CINCH: A cooperatively designed marking interface for 3D pathway selection. In *Symposium on User Interface Software and Technology*, 33–42.

ANDERSON, L., ESSER, J., AND INTERRANTE, V. 2003. A virtual environment for conceptual design in architecture. In *Proceedings of the workshop on Virtual environments 2003*, ACM, New York, NY, USA, EGVE '03, 57–63.

AZUMA, R., AND BISHOP, G. 1994. Improving static and dynamic registration in an optical see-through hmd. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 197–204.

BOTT, J., CROWLEY, J., AND LAVIOLA, J. 2009. Exploring 3d gestural interfaces for music creation in video games. In *Proceedings of The Fourth International Conference on the Foundations of Digital Games 2009*, 18–25.

BOWMAN, D. A., AND HODGES, L. F. 1997. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 35–ff.

BOWMAN, D. A., KOLLER, D., AND HODGES, L. F. 1997. Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques. In *Proceedings of the Virtual Reality Annual International Symposium*, 45–52.

BOWMAN, D. A., WINEMAN, J., HODGES, L. F., AND ALLISON, D. 1998. Designing animal habitats within an immersive ve. *IEEE Comput. Graph. Appl. 18*, 5, 9–13.

BOWMAN, D. A., KRUIJFF, E., LAVIOLA, J. J., AND POUPYREV, I. 2004. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

BRODY, B., AND HARTMAN, C. 2000. Painting space with BLUI. In *SIGGRAPH '00 Conference Abstracts and Applications*, 242.

BROOKS, F. P. 1988. Grasping reality through illusion: interactive graphics serving science. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, CHI '88, 1–11.

BROWN UNIVERSITY LITERARY ARTS PROGRAM, 2011. http://www.brown.edu/departments/literary_arts.

BRYSON, S., AND LEVIT, C. 1991. The virtual windtunnel: an environment for the exploration of three-dimensional unsteady flows. In *Proceedings of the 2nd conference on Visualization '91*, IEEE Computer Society Press, Los Alamitos, CA, USA, VIS '91, 17–24.

BUTTERWORTH, J., DAVIDSON, A., HENCH, S., AND OLANO, M. T. 1992. 3DM: A three dimensional modeler using a head-mounted display. In *Symposium on Interactive 3D Graphics*, 135–138.

CHARBONNEAU, E., MILLER, A., WINGRAVE, C., AND LAVIOLA, JR., J. J. 2009. Understanding visual interfaces for the next generation of dance-based rhythm video games. In *Sandbox '09: Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, ACM Press, 119–126.

CLARK, J. H. 1976. Designing surfaces in 3-D. *Communications of the ACM 19*, 8, 454–460.

COFFEY, D., KORSAKOV, F., AND KEEFE, D. F. 2010. Low cost VR meets low cost multi-touch. In *Proceedings of International Symposium on Visual Computing*, 351–360.

COFFEY, D., MALBRAATEN, N., LE, T., BORAZJANI, I., SOTIROPOULOS, F., AND KEEFE, D. F. 2011. Slice WIM: a Multi-Surface, Multi-Touch interface for Overview+Detail exploration of volume datasets in virtual reality. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 191–198.

CONNER, B. D., SNIBBE, S. S., HERNDON, K. P., ROBBINS, D. C., ZELEZNIK, R. C., AND VAN DAM, A. 1992. Three-dimensional widgets. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 183–188.

CRASSIDIS, J. L., AND MARKLEY, F. L. 2003. Unscented filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics 26*, 4, 536–542.

DEERING, M. 1992. High resolution virtual reality. In *SIGGRAPH '92 Conference on Computer Graphics and Interactive Techniques*, 195–202.

DEERING, M. F. 1995. Holosketch: A virtual reality sketching/animation tool. *ACM Transactions of Computer-Human Interaction 2*, 3, 220–238.

DUDA, R. O., HART, P. E., AND STORK, D. G. 2001. *Pattern Classification.* John Wiley and Sons.

FEINER, S., MACINTYRE, B., HAUPT, M., AND SOLOMON, E. 1993. Windows on the world: 2d windows for 3d augmented reality. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 145–155.

FERGUSON, M., 2003. June. Film created with the SANDDE animation system. National Film Board of Canada http://www.nfb.ca.

FERGUSON, M., 2006. Falling in love again. Film created with the SANDDE animation system. National Film Board of Canada http://www.nfb.ca.

FRIEDMAN, J. H. 1996. Another approach to polychotomous classification. Tech. rep., Department of Statistics, Stanford University.

FRÖHLICH, B., AND PLATE, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, CHI '00, 526–531.

GALYEAN, T. A., AND HUGHES, J. F. 1991. Sculpting: An interactive volumetric modeling technique. In *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, vol. 25, ACM, 267–274.

GIANSANTI, D., MACELLARI, V., AND MACCIONI, G. 2003. Is it feasible to reconstruct body segment 3-d position and orientation using accelerometric data. *IEEE Trans. Biomed. Eng 50*, 2003.

GREGORY, A. D., EHMANN, S. A., AND LIN, M. C. 2000. inTouch: Interactive multiresolution modeling and 3D painting with a haptic interface. In *Proceedings of IEEE VR 2000*, 45–52.

GREY, J. 2002. Human-computer interaction in life drawing, a fine artist‘s perspective. In *Proceedings of the Sixth International Conference on Information Visualisation*.

GROSSMAN, T., WIGDOR, D., AND BALAKRISHNAN, R. 2004. Multi-finger gestural interaction with 3d volumetric displays. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, UIST '04, 61–70.

GUNTURK, B. K., GLOTZBACH, J., ALTUNBASAK, Y., AND SCHAFER, R. W. 2005. Demosaicking: color filter array interpolation. *IEEE Signal processing magazine 22*, 44–54.

HENTSCHEL, B., TEDJO, I., PROBST, M., WOLTER, M., BEHR, M., BISCHOF, C., AND KUHLEN, T. 2008. Interactive blood damage analysis for ventricular assist devices. *IEEE Transactions on Visualization and Computer Graphics 14*, 6, 1515–1522.

HINCKLEY, K., PAUSCH, R., DOWNS, J. H., PROFFITT, D., AND KASSELL, N. F. 1997. The props-based interface for neurosurgical visualization. *Studies in Health Technology and Informatics 39*, 552–562. PMID: 10168950.

HOFFMAN, M., VARCHOLIK, P., AND LAVIOLA, J. 2010. Breaking the status quo: Improving 3d gesture recognition with spatially convenient input devices. In *IEEE Virtual Reality 2010*, IEEE Press, 59–66.

HUA, J., AND QIN, H. 2004. Haptics-based dynamic implicit solid modeling. *IEEE Transactions on Visualization and Computer Graphics 10*, 5, 574–586.

JACKSON, B., AND KEEFE, D. F. 2011. Sketching over props: Understanding and interpreting 3d sketch input relative to rapid prototype props. In *Proceedings of IUI 2011 Sketch Recognition Workshop*.

JULIER, S., AND UHLMANN, J. 1997. A new extension of the kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*.

KEEFE, D. F., FELIZ, D. A., MOSCOVICH, T., LAIDLAW, D. H., AND LAVIOLA JR., J. J. 2001. CavePainting: A fully immersive 3D artistic medium and interactive experience. In *Proceedings of I3D 2001*, 85–93.

KEEFE, D. F., ZELEZNIK, R. C., AND LAIDLAW, D. H. 2007. Drawing on air: Input techniques for controlled 3D line illustration. *IEEE Transactions on Visualization and Computer Graphics 13*, 5, 1067–1081.

KEEFE, D. F., ACEVEDO, D., MILES, J., DRURY, F., SWARTZ, S. M., AND LAIDLAW, D. H. 2008. Scientific sketching for collaborative VR visualization design. *IEEE Transactions on Visualization and Computer Graphics 14*, 4, 835–847.

KEEFE, D. F., ZELEZNIK, R. C., AND LAIDLAW, D. H. 2008. Tech-note: Dynamic dragging for input of 3D trajectories. In *Proceedings of IEEE Symposium on 3D User Interfaces 2008*, 51–54.

KEEFE, D. F. 2007. *Interactive 3D Drawing for Free-Form Modeling in Scientific Visualization and Art: Tools, Methodologies, and Theoretical Foundations*. PhD thesis, Brown University.

KEEFE, D. F. 2008. Free-form vr interactions in scientific visualization.

KEEFE, D. F., 2009. Creative 3d form-making in visual art and visual design for science. CHI 2009 Workshop on Computational Creativity Support: Using Algorithms and Machine Learning to Help People Be More Creative, April.

KEEFE, D. F. 2010. Integrating visualization and interaction research to improve scientific workflows. *IEEE Computer Graphics and Applications 30*, 2, 8–13.

KEEFE, D. F. 2011, In press. From gesture to form: The evolution of expressive freehand spatial interfaces. *Leonardo*.

KITWARE, INC. Paraview: Open source scientific visualization. http://www.paraview.org/. Accessed May, 2011.

KONCHADA, V., JACKSON, B., LE, T., BORAZJANI, I., SOTIROPOULOS, F., AND KEEFE, D. F. 2011. Supporting internal visualization of biomedical datasets via 3d rapid prototypes and sketch-based gestures. In *Poster Proceedings of ACM Symposium on Interactive 3D Graphics and Games*.

KONIECZNY, J., MEYER, G., SHIMIZU, C., HECKMAN, J., MANYEN, M., AND RABENS, M. 2008. Vr spray painting for training and design. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, VRST '08, 293–294.

KRATZ, L., SMITH, M., AND LEE, F. J. 2007. Wiizards: 3d gesture recognition for game play input. In *Future Play '07: Proceedings of the 2007 conference on Future Play*, ACM, New York, NY, USA, 209–212.

KRUSZYNSKI, K., AND VAN LIERE, R. 2009. Tangible props for scientific visualization: concept, requirements, application. *Virtual Reality 13*, 235–244. 10.1007/s10055-009-0126-1.

LAVIOLA, J. J., AND ZELEZNIK, R. C. 2007. A practical approach for writer-dependent symbol recognition using a writer-independent symbol recognizer. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 11, 1917–1926.

LAVIOLA, J. 2000. Msvt: A virtual reality-based multimodal scientific visualization tool. In *Proceedings of the Third IASTED International Conference on Computer Graphics and Imaging*, 1–7.

LAVIOLA, J. J. 2003. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003*, ACM, New York, NY, USA, 199–206.

LUINGE, H., VELTINK, P., AND BATEN, C. 1999. Estimating orientation with gyroscopes and accelerometers. *Technology and Health Care 7*, 6, 455–459.

MÄKELÄ, W. 2005. Working 3D meshes and particles with finger tips, towards an immersive artists' interface. In *IEEE VR 2005 Workshop Proceedings*, 77–80.

MAPES, D., AND MOSHELL, M. 1995. A two-handed interface for object manipulation in virtual environments. *Presence: Teleoper. Virtual Environ. 4*, 4, 403–416.

MINE, M. R., BROOKS, JR., F. P., AND SEQUIN, C. H. 1997. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 19–26.

MINE, M. 1995. Virtual environment interaction techniques. Tech. rep., UNC Chapel Hill CS Dept.

MORSTAD, P., 2004. Moon man. Film created with the SANDDE animation system. National Film Board of Canada http://www.nfb.ca.

PAUSCH, R., BURNETTE, T., BROCKWAY, D., AND WEIBLEN, M. E. 1995. Navigation and locomotion in virtual worlds via flight into hand-held miniatures. In *SIGGRAPH '95: Proceedings of the 22nd* *annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 399–400.

PAUSCH, R., BURNETTE, T., BROCKWAY, D., AND WEIBLEN, M. E. 1995. Navigation and locomotion in virtual worlds via flight into hand-held miniatures. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM, 399–400.

PIERCE, J. S., FORSBERG, A. S., CONWAY, M. J., HONG, S., ZELEZNIK, R. C., AND MINE, M. R. 1997. Image plane interaction techniques in 3d immersive environments. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 39–ff.

POUPYREV, I., BILLINGHURST, M., WEGHORST, S., AND ICHIKAWA, T. 1996. The go-go interaction technique: non-linear mapping for direct manipulation in vr. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 79–80.

RAZZAQUE, S. 2005. *Redirected walking*. PhD thesis, Chapel Hill, NC, USA. AAI3190299.

REHM, M., BEE, N., AND ANDRÉ, E. 2008. Wave like an egyptian: accelerometer based gesture recognition for culture specific interactions. In *BCS-HCI '08: Proceedings of the 22nd British HCI Group Annual Conference on HCI 2008*, British Computer Society, Swinton, UK, UK, 13–22.

RUBIN, C. B., AND KEEFE, D. F., 2002. Hiding spaces: A cave of elusive immateriality. ACM SIGGRAPH 2002 Conference Abstracts and Applications, July.

RUBINE, D. 1991. Specifying gestures by example. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 329–337.

SACHS, E., ROBERTS, A., AND STOOPS, D. 1991. 3-draw: A tool for designing 3D shapes. *IEEE Computer Graphics and Applications 11*, 6, 18–26.

SCHAPIRE, R. E. 1999. A brief introduction to boosting. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1401–1406.

SCHKOLNE, S., PRUETT, M., AND SCHRÖDER, P. 2001. Surface drawing: creating organic 3D shapes with the hand and tangible tools. In *Proceedings of CHI '01*, 261–268.

SCHKOLNE, S., ISHII, H., AND SCHRÖDER, P. 2004. Immersive design of DNA molecules with a tangible interface. In *Proceedings of IEEE Visualization*, 227–234.

SCHKOLNE, S. 2003. *3D Interfaces for Spatial Construction*. PhD thesis, California Institute of Technology.

SCHLÖMER, T., POPPINGA, B., HENZE, N., AND BOLL, S. 2008. Gesture recognition with a wii controller. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*, ACM, New York, NY, USA, 11–14.

SCHMANDT, C. 1983. Spatial input/display correspondence in a stereoscopic computer graphic work station. In *SIGGRAPH '83: Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, 253–261.

SCHROERING, M., GRIMM, C., AND PLESS, R. 2003. A new input device for 3D sketching. In *Vision Interface*, 311–318.

SENSABLE TECHNOLOGIES, INC. 2005. *FreeForm Concept Product Brochure*.

SHEPPARD, R., AND NAHRSTEDT, K. 2009. Merging research modalities: TED (tele-immersive dance) collaboration offers a model for performance-based research and creative development. In *Proceedings of the Computational Creativity Support Workshop at ACM CHI 2009*.

SHIRAI, A., GESLIN, E., AND RICHIR, S. 2007. Wiimedia: motion analysis methods and applications using a consumer video game controller. In *Sandbox '07: Proceedings of the 2007 ACM SIGGRAPH symposium on Video games*, ACM, New York, NY, USA, 133–140.

SHIRATORI, T., AND HODGINS, J. K. 2008. Accelerometer-based user interfaces for the control of a physically simulated character. *ACM Trans. Graph. 27*, 5, 1–9.

SHIVARAM, G., AND SEETHARAMAN, G. 1998. A new technique for finding the optical center of cameras. In *ICIP 98: Proceedings of 1998 International Conference on Image Processing*, vol. 2, 167–171.

SHOTTON, J., FIZGIBBON, A., COOK, M., SHARP, T., FINOCCHIO, M., MOORE, R., KIPMAN, A., AND BLAKE, A. 2011. Real-time human pose recognition in parts from single depth images. In *IEEE Computer Vision and Pattern Recognition (CVPR) 2011*, IEEE.

SNIBBE, S., ANDERSON, S., AND VERPLANK, B. 1998. Springs and constraints for 3D drawing. In *Proceedings of the Third Phantom Users Group*. M.I.T. Artificial Intelligence Laboratory Technical Report AITR-1643.

SNIBBE, S., 1998. Bondary functions, installation by scott snibbe. http://www.snibbe.com/. Accessed May, 2011.

SOBEL, J. S., FORSBERG, A. S., LAIDLAW, D. H., ZELEZNIK, R. C., KEEFE, D. F., PIVKIN, I., KARNIADAKIS, G. E., RICHARDSON, P., AND SWARTZ, S. 2004. Particle flurries: Synoptic 3D pulsatile flow visualization. *IEEE Computer Graphics and Applications 24*, 2 (March/April), 76–85.

SONG, H., GUIMBRETIÈRE, F., HU, C., AND LIPSON, H. 2006. Modelcraft: capturing freehand annotations and edits on physical 3d models. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 13–22.

STEINICKE, F., BRUDER, G., HINRICHS, K., STEED, A., AND GERLACH, A. L. 2009. Does a gradual transition to the virtual world increase presence? In *Proceedings of the 2009 IEEE Virtual Reality Conference*, IEEE Computer Society, Washington, DC, USA, 203–210.

STOAKLEY, R., CONWAY, M. J., AND PAUSCH, R. 1995. Virtual reality on a wim: interactive worlds in miniature. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 265–272.

STOAKLEY, R., CONWAY, M. J., AND PAUSCH, R. 1995. Virtual reality on a WIM: interactive worlds in miniature. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co., Denver, Colorado, United States, 265–272.

SUTHERLAND, I. 1968. A head-mounted three dimensional display. In *Proceedings of the AFIPS Fall Joint Computer Conference*, vol. 33, 757–764.

TINKERMAN, 2008. Tinkermans method - casting textured silcone, http://nuigroup.com/forums/viewthread/2383/, July.

TRUEBA, R., ANDUJAR, C., AND ARGELAGUET, F. 2009. Complexity and occlusion management for the World-in-Miniature metaphor. In *Smart Graphics*. 155–166.

VALKOV, D., STEINICKE, F., BRUDER, G., AND HINRICHS, K. 2011. 2d touching of 3d stereoscopic objects. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, ACM, New York, NY, USA, CHI '11, 1353–1362.

VARCHOLIK, P. D., LAVIOLA, JR., J. J., AND HUGHES, C. 2009. The bespoke 3dui xna framework: a low-cost platform for prototyping 3d spatial interfaces in video games. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, ACM, New York, NY, USA, Sandbox '09, 55–61.

WAN, E. A., AND VAN DER MERWE, R. 2002. The unscented kalman filter for nonlinear estimation. 153–158.

WESCHE, G., AND SEIDEL, H.-P. 2001. Freedrawer: a free-form sketching system on the responsive workbench. In *Proceedings of VRST 2001*, 167–174.

WILLIAMSON, R., AND ANDREWS, B. 2001. Detecting absolute human knee angle and angular velocity using accelerometers and rate gyroscopes. *Medical and Biological Engineering and Computing 39*, 3, 294–302.

WILLIAMSON, B., WINGRAVE, C., AND LAVIOLA, J. 2010. Realnav: Exploring natural user interfaces for locomotion in video games. In *Proceedings of IEEE Symposium on 3D User Interfaces 2010*, IEEE Computer Society, 3 – 10.

WINGRAVE, C. A., HACIAHMETOGLU, Y., AND BOWMAN, D. A. 2006. Overcoming world in miniature limitations by a scaled and scrolling WIM. In *Proceedings of the IEEE conference on Virtual Reality*, IEEE Computer Society, 11–16.

WINGRAVE, C., WILLIAMSON, B., VARCHOLIK, P., ROSE, J., MILLER, A., CHARBONNEAU, E., BOTT, J., AND LAVIOLA, J. 2010. Wii remote and beyond: Using spatially convenient devices for 3duis. *IEEE Computer Graphics and Applications 30*, 2, 71–85.

WLOKA, M. M., AND GREENFIELD, E. 1995. The virtual tricorder: a uniform interface for virtual reality. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, ACM, New York, NY, USA, 39–40.

ZHANG, N., ZHOU, X., SHEN, Y., AND SWEET, R. 2010. Volumetric modeling in laser bph therapy simulation. *IEEE Transactions on Visualization and Computer Graphics 16* (November), 1405–1412.

ZHOU, W., CORREIA, S., AND LAIDLAW, D. H. 2008. Haptics-assisted 3D lasso drawing for tracts-of-interest selection in DTI visualization.