Creating Next-Gen 3D Interactive Apps with Motion Control and Unity3D

SIGGRAPH 2014 Workshop

Lecturers:

Daniel Plemmons
UX Researcher and Software Engineer
Leap Motion

David Holz
CTO and Co-Founder
Leap Motion

Abstract

Modern native and web-based games and applications continue to push the boundaries of production quality and interactive complexity. Tools and techniques traditionally reserved for large teams building state-of-the-art gaming applications are now becoming the tools of choice for small game and application developers. Thanks to recent advances in motion control, developers can immerse users in a natural interface controlled directly by their bodies, which can be mapped with stunning real-time accuracy into the virtual space.

This course combines the powerful and versatile Unity3D engine with the Leap Motion Controller to demonstrate a modern toolkit for developing consumer-accessible next-gen 3D applications. Through a series of live-coded examples, attendees will be grounded in the fundamentals of using the Unity3D engine for game and application development, integrating the Leap Motion Controller into an application, and designing and developing a next-generation 3D interactive application with motion control. Attendees need not have advanced knowledge of 3D game engines or motion control, though an understanding of C# or similar languages, along with some knowledge of basic linear algebra concepts, will be useful. The course will also cover broader issues around motion-control interaction design.

As natural interface technologies are on the rise, the question of input is taking on an increasingly important role. In this diverse interface landscape, devices like the Oculus Rift, Sifteo, Leap Motion Controller, PS Move, Kinect, and others present unique challenges to developers, as choices about interaction aesthetics begin with the crucial question of input hardware.

About the Lecturers

Daniel Plemmons

UX Researcher and Software Engineer

Leap Motion

Daniel is an interactive media creator who is passionate about exploring the border between the physical and digital worlds. A graduate of the Savannah College of Art and Design in Atlanta, his work spans mainstream mobile games played by millions, site-specific games for gallery shows, an IndieCade finalist physical installation game, Twitter bots, and natural user interface design for a variety of new and emerging technologies. He tweets @randomoutput and his work can be found at RandomOutputDesign.com.

David Holz

CTO and Co-Founder

Leap Motion

David is an inventor and mathematician who loves to solve fundamental problems. After reading Stephen Hawking's A Brief History of Time in eighth grade, he devised a simple way to test the theory of general relativity. While studying Applied Math at the University of North Carolina at Chapel Hill, he worked for NASA. Frustrated by the limits of 3D design – where it takes an hour to create what a five-year-old can do with a lump of clay in five minutes – David left his PhD program to start Leap Motion.

Course Overview

- **15 Minutes: Setting Up Our Development Environment**
    - Downloading and installing Unity3D
    - Setting up the Leap Motion Controller
    - Setting up the Leap Motion SDK
    - Setting up a new Unity3D project with the Leap Motion Controller

- **10 Minutes: The Modern Motion Control Landscape**
    - A brief overview of the current state of motion-control application technologies and evolving design principles
    - Motion control in games and interactive media
    - Key questions – when is motion control the right decision, and where does each technology fit in the interactive ecosystem?

- **10 Minutes: An Introduction to the Leap Motion API**
    - *This section will be presented as live-coded examples and annotated in the notes*
    - The Leap Motion Starter Kit
    - Understanding the coordinate space
    - How to interpret and think about sensor data
    - Visual feedback and presence using rigged hands

- **20 Minutes: Live Coding – 3 Techniques for Motion Interaction**
    - 1:1 2D mapping
    - Direct manipulation
    - Gestural abstraction

- **30 Minutes: Understanding and Building Feedback Loops in Interactive Applications**
    - The limitations of motion control
        - Motion control interfaces lack the foundational subconscious audible and haptic feedback that users and developers have become reliant on
    - Building foundational feedback for user interactions
        - Motion control applications must develop their own foundational language for feedback with the user

Table of Contents

Section 1: The Modern Motion Control Landscape

While popularly associated with science fiction and high-tech interfaces, motion controls have played a role in the human-computer interaction space for decades. Historically, they have fallen within the purview of large, expensive art installations, or professional and industrial applications. Over the past decade, however, the convergence of modern hardware and computer processing advancements has made consumer use of motion controls a realistic proposition.

This course will focus on using Unity3D and the Leap Motion Controller – hardware and software solutions squarely focused on the consumer market – to create applications for this market. Before approaching application development, a cursory exploration of the recent history of the motion control space is necessary.

1.1 Primary Contexts of Motion Control Applications

In futurist popular culture and media, motion control is often depicted as a ubiquitous part of everyday life. Though this may someday be the case, today there exist only a few specific contexts in which motion control has been widely applied, with varying levels of success. Developing the next generation of motion control applications in these spaces, or extending motion control into new domains, requires an understanding of how each context has impacted the design and adoption of its respective motion control applications.

The following sections will discuss, at a high level, the following set of contexts for motion control applications: gaming, artistic installations, and the new wave of consumer virtual reality (typified by the emergence of the Oculus Rift). These are not intended to be exhaustive, but to serve as a common set of touchpoints for the further critical examination of motion control design, and to ground the rest of the course discussion in the state of the art as it exists today.

1.2 Selected Works Using Motion Control in Modern Gaming

There are far too many motion-controlled games to discuss the full range of experiences in this section. Instead, we will discuss two common goals of motion control in gaming, and present modern examples of games which typify these goals.

Player immersion is often touted as a prime goal of motion-controlled games. Underlying much of this goal is the reasoning that if a player's body is more involved in the play experience, and simulating real-world actions, then they will feel more immersed in the experience. Take, for example, this quote from the Amazon.com page for the Playstation Move motion controller:

> PlayStation Move redefines motion gaming with the most immersive and realistic gaming experience only possible on the PlayStation3 system. The simple, easy-to-use controller captures a full range of motion giving you ultimate control over how you play the game. With a diverse

> selection of games and new ones launching all the time, you can enjoy hours of fun with friends and family.[1]

Dance Central. One of the most successful and direct applications of this thinking is the *Dance Central* series for the Microsoft Xbox 360 using the Kinect 3D motion tracking camera.[2] *Dance Central* was preceded by a wide variety of rhythm and dance games, including *Dance Dance Revolution*, *Rock Band*, and *Guitar Hero*. The Kinect sensor made it possible for *Dance Central* to make use of real-time full body tracking.

In *Dance Central*, an on-screen avatar performs a repetitive set of dance moves, set to popular music. Players must mirror the on-screen avatar and are scored on the accuracy of their movements via the data from the Kinect sensor. As players improve their execution, the difficulty of moves increases, along with the accuracy demanded by the game's scoring system.

Dance Central and its sequels hit upon a confluence of game and technology to create immersive interactions. A broad set of realistic dance actions are reasonably trackable by the Kinect sensor, and the scoring system is opaque enough to render tracking inaccuracy a non-issue. The player interactions so closely mimic real world actions that they create a greatly enhanced sense of immersion in comparison to previous motion-control dance titles.[3]

Johann Sebastian Joust. In contradistinction to the goal of immersion, many successful motion-control titles instead draw attention to the motion controls themselves – focusing on players performing silly or difficult actions to create entertaining experiences. One excellent example is Die Gute Fabrik's critically acclaimed motion-control game *Johann Sebastian Joust*:

> "The rhetoric goes that motion control is supposed to be like the Holodeck or virtual reality," says [Die Gute Fabrik game designer Doug] Wilson. "People expect it to be more immersive, because you're actually doing the actions. But it's never like that, because all these controllers are kind of goofy and the technology is actually pretty crude, and will continue to be crude for years.... We're years away from being Hamlet on the Holodeck."[4]

JS Joust draws from the legacy of folk and field games to create a unique motion-control experience.[5] Each player holds a PlayStation Move controller as an electronic rendition of Johann Sebastian Bach's *Brandenburg Concerto #2* plays with a varying tempo. The goal of the game is to force other players to

---

[1] Sony, "Playstation Move Motion Controller," *Amazon.com*, accessed June 7, 2014, http://www.amazon.com/Playstation-Move-Motion-Controller-3/dp/B002I0J51U.
[2] "Dance Central 2," *MetaCritic*, accessed June 7, 2014, http://www.metacritic.com/game/xbox-360/dance-central-2.
[3] Julye Huggins, "Dance Central 2: Review from IGN," *The Dance Current*, January 2, 2012, http://www.thedancecurrent.com/video/dance-central-2-review-ign.
[4] Kris Graft, "Three Factors that Stand in the Way of Motion Controls' Future," October 8, 2013, http://www.gamasutra.com/view/news/201841/Three_factors_that_stand_in_the_way_of_motion_controls_future.php.
[5] Douglas Wilson, "Designing for the Pleasures of Disputation, or How to Make Friends by Trying to Kick Them!" manuscript for Ph.D. dissertation at the IT University of Copenhagen, August 13, 2012, http://doougle.net/phd/Designing_for_the_Pleasures_of_Disputation.pdf.

move their controller quickly. If the accelerometer in the controller detects too high a spike, the player is out. The tempo of the music determines how sensitive the game is to movement.

In this way, *JS Joust* draws the player's attention directly to the motion-control experience. The resulting game dynamics include players tiptoeing about each other as if caught in slow motion. Elegant spins, quick feints, gambits, and more than a small amount of shoving are common. The simplicity and physical freedom of the game also afford interesting social behaviors.[6] As a result, the game focuses more on drawing attention to the bodies and movements of the players and the playspace than to immersion in any particular simulation or fantasy.

Each of these games sits on opposite ends of a continuum ranging from immersion to self-awareness. Though proponents of each design style may speak ill of the other (as seen in Wilson's quote above), where a work lies along the continuum is not so much a quality judgement as a lens through which to understand and critically evaluate the work.

1.3 Selected Works Using Motion Control in Art Installations

Motion control, particularly in the form of computer vision – such as the works of Golan Levin[7] and Brian Knep[8] – has been at play in the new media arts world for many years. In this section, we will examine two pieces of interactive art in which motion control was used to create a unique physical and multimedia experience, while at the same time lowering the entry barrier for public performances.

Growth. Craig Winslow's 2013 work *Growth* is an interactive motion-controlled installation featuring projection mapping. In *Growth*, viewers interact with an abstract virtual jungle, which forms an environment that can be modified by the viewer's actions.

[6] Griffin McElroy, "Folk Lore: How Johann Sebastian Joust is Defining a New Gaming Genre," *Polygon*, February 22, 2012, http://www.polygon.com/2012/10/12/3494404/folk-lore-how-johann-sebastian-joust-is-defining-a-new-gaming-genre.
[7] See http://www.flong.com.
[8] See http://www.blep.com.

According to Winslow:

> The most powerful moment for me was seeing a mother and her two boys interact with complete awe. Once they knew they were in control of the experience, they waved their hands, wiggled their fingers – but in a very respectful way. It reminded me of a quote by Robert Irwin I was told near the beginning of the project, which influenced our intent more than I knew: "You can't plan nature; you court her."[9]

The use of motion control in *Growth* allowed viewers to interact in a unique, natural fashion, not otherwise afforded by traditional interfaces:

> Embracing the natural way we would expect people to interact with the device, we made slow soothing movements augment lighting, while aggressive swipes brought in black recursive animations.... Leap Motion amplified the story we were trying to tell, as the viewer's human interaction contributed to impact dynamically on the installation.[10]

While *Growth* allows for natural interactions from individual viewers, Rafael Lozano-Hemmer's *Frequency and Volume* is a prime example of how motion control can be used to draw the public into a piece of performance art. In *Frequency and Volume*, the viewers' shadows on the gallery wall generate the work.[11] According to Lozano-Hemmer:

[9] Kate Mitchell, "Growth: Art Installation Powered by Leap Motion," *Leap Motion Blog*, October 4, 2013, https://www.leapmotion.com/blog/growth-art-installation-powered-by-leap-motion.
[10] Ibid.
[11] "Lozano-Hemmer wants you to interact," *Metro.co.uk*, October 12, 2008, http://metro.co.uk/2008/10/12/lozano-hemmer-wants-you-to-interact-24615.

> *Frequency and Volume* enables participants to tune into and listen to different radio frequencies by using their own bodies. A computerised tracking system detects participants' shadows, which are projected on a wall of the exhibition space. The shadows scan the radio waves with their presence and position, while their size controls the volume of the signal. The piece can tune into any frequency between 150 kHz and 1.5 GHz, including air traffic control, FM, AM, short wave, cellular, CB, satellite, wireless telecommunication systems and radio navigation.[12]

1.4 Use of Motion Control in Virtual Reality

The emerging world of consumer virtual reality offers a new set of opportunities for motion control. Much like the introduction of motion controls to traditional gaming, virtual reality experiences often seek to create an additional sense of immersion. Though heated discussions are taking place, the larger development community has yet to standardize on any particular set of input devices.

We can identify two varieties of motion control for VR – physical controllers, such as the Razer Hydra, and Wiimote; and in-air controllers, such as the Kinect and Leap Motion Controller. Each category has its own advantages and drawbacks. Physical controllers have the advantage of tactile feedback, buttons, and additional analog inputs. They can be more generalized and abstracted to many uses.[13] In-air devices allow for more natural input with less hardware overhead, but are more specialized for particular interactions as they are optimized to imitate particular real-world interactions. As VR technologies approach the consumer market in the near term, motion-control interfaces are approaching a crossroads with huge implications for the long term.

[12] Rafael Lozano-Hemmer, *Frequency and Volume*, accessed June 7, 2014, http://www.lozano-hemmer.com/frequency_and_volume.php. Image from Jessica Lack, "Artist of the week no 11: Rafael Lozano-Hemmer," *TheGuardian.com* (Oct. 15, 2008), accessed June 7, 2014, http://www.theguardian.com/artanddesign/2008/oct/15/art1

[13] For example, see Reed Albergotti, "Virtual-Reality Company Survios Gets $4 Million in Venture Funding," *Digits*, May 19, 2014, http://blogs.wsj.com/digits/2014/05/19/virtual-reality-company-survios-gets-4-million-in-venture-funding.

1.5 When is Motion Control the Right Decision?

No one wants to write a novel using just a mouse, or create a 3D design with nothing but a keyboard. Similarly, not every project is the right fit for motion control, and developers must contrast their goals with the relative advantages and drawbacks of the input options available.

Motion control is best suited to interactions and experiences where developers wish to: increase immersion; bring attention to the players' bodies and the play space; create multimedia experiences that respond to natural interactions; make complex or abstract pieces more accessible; or provide natural inputs for virtual reality. In all cases, it should augment the experience (e.g. connecting with abstract nature in *Growth*) or remove a fundamental obstacle (e.g. the inability to see your real hands in virtual reality).

Section 2: Three Techniques for Motion Control Interaction

In this section, we will cover three techniques for interacting with virtual spaces with motion control. The details of setting up the development environment are best explained by Leap Motion's developer resources,[14] while an explanation of Unity's development environment is detailed in their documentation.[15] We will briefly review Leap Motion and Unity3D for those who are unfamiliar, and in this section will discuss the implementation theory and the pros and cons associated with each technique. A few key code snippets are included in these notes; additional relevant code snippets will be provided at course time. These implementations will assume the use of Leap Motion's Version 2 Tracking SDK.

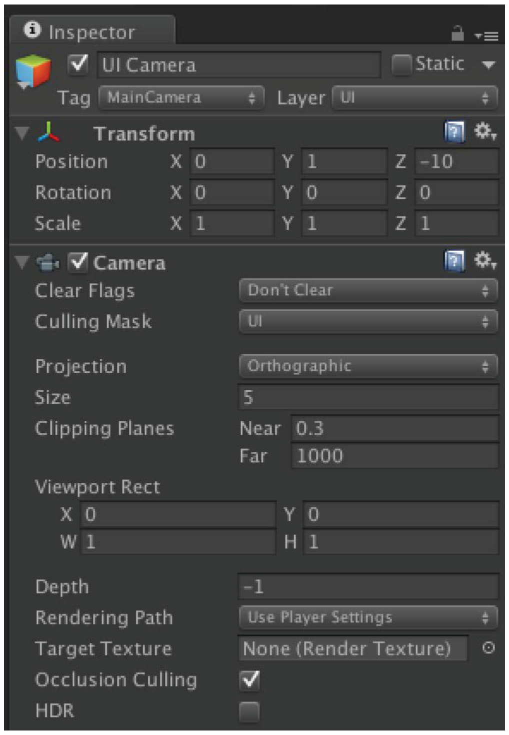
2.1 1:1 2D mapping

One-to-one 2D mapping is the simplest of the three techniques discussed in the course. This refers to taking the X and Y space position of the hand or finger, normalizing that location within a defined interaction space, and mapping that location onto the 2D space of the screen.

For a variety of reasons, many UI design problems lend themselves well to 2D interactions. A designer may need to adapt an existing interface to gestural control, or the information design of a set of relevant interactions lends itself strongly to 2D interaction. Implementing 2D mapping in Unity3D with Leap Motion is relatively trivial.

We will use a common technique for separating UI rendering in Unity from world space rendering. We will then create a new camera in the scene, and set it to render an orthographic view above the main world camera. We also change the clearing settings to allow the world camera to be viewed through any blank areas of our UI camera.

[14] *Leap Motion Developer*, accessed June 7, 2014, https://www.developer.leapmotion.com.
[15] *Learn with Unity*, accessed June 7, 2014, http://unity3d.com/learn.



```
Inspector
✓ UI Camera                         Static
Tag MainCamera          Layer UI
Transform
Position          X 0      Y 1      Z -10
Rotation          X 0      Y 0      Z 0
Scale             X 1      Y 1      Z 1
✓ Camera
Clear Flags            Don't Clear
Culling Mask           UI
Projection             Orthographic
Size                   5
Clipping Planes        Near 0.3
                       Far  1000
Viewport Rect
  X 0                  Y 0
  W 1                  H 1
Depth                  -1
Rendering Path         Use Player Settings
Target Texture         None (Render Texture)
Occlusion Culling      ✓
HDR
```

Next, we are going to be mapping a 2D cursor onto our UI layer. First, we create the cursor and set it to be rendered by our UI camera. Then we will add a script to the cursor to move it based on the user's hand. Unity allows developers to attach individual scripts describing behavior to an object as "components." This component model allows for small simple scripts to be combined to create more complex behaviors, as will be shown later in the course.

Using Leap Motion's hacker helper, we convert the Leap Motion data into a Unity3D Vector3 type:

```
Vector3 handPosition = mainHand.PalmPosition.ToUnityTranslated();
```

We can then convert this data to Unity world coordinates. This uses a very simple normalization technique, as each axis of data is given a minimum and maximum value in the configuration. Incoming Leap Motion frame data is then fed in and normalized according to these configurations.

```
float normalizeValue ( float value, float min, float max ) {
        return (value - min) / (max - min);
}

//Normalize
handPosition.x = normalizeValue( leapPosition.x, _leapMin.x, _leapMax.x );
handPosition.y = normalizeValue( leapPosition.y, _leapMin.y, _leapMax.y );
handPosition.z = normalizeValue( leapPosition.z, _leapMin.z, _leapMax.z ) * -1;

//Set to world coordinates
handPosition.x = _worldMin.x + (handPosition.x * (_worldMax.x - _worldMin.x));
handPosition.y = _worldMin.y + (handPosition.y * (_worldMax.y - _worldMin.y));
handPosition.z = 10;
```

Given these world coordinates, and a camera, we can project these screen coordinates onto the camera, and then back into the world along a particular plane. This plane will be rendered as our user interface. As we have an orthographic camera, moving game objects forward and back in space will cause them to be rendered in front of or behind other items in our UI.

```
screenPosition = _leapManager._mainCam.WorldToScreenPoint( handPosition );
uiPosition = _leapManager._mainCam.ScreenToWorldPoint ( new Vector3( screenPosition.x, screenPosition.y,
uiPlane) );
```

Given these world space coordinates, we simply set the location of the cursor game object.

```
transform.position = uiPosition;
```

Running the project and placing a hand over the Leap Motion Controller should show the hand in space.

2.2 3D Direct Manipulation

Direct manipulation represents a more skeuomorphic design approach for virtual space. Skeuomorphic designs use references to real-world objects and experiences to help a person interacting with a system craft a mental model as to how that system behaves.[16] These sorts of orientation aids are often helpful when introducing people to new technologies or interfaces. As an interface matures, however, skeuomorphic designs will often take a back seat to visual styles more suited to the nature of the technology or interaction medium, as has been seen with the evolution of the iOS mobile operating system:

> When we sat down last November (to work on iOS 7), we understood that people had already become comfortable with touching glass, they didn't need physical buttons, they understood the benefits.... So there was an incredible liberty in not having to reference the physical world so literally. We were trying to create an environment that was less specific. It got design out of the way.
> — Jonathan Ive[17]

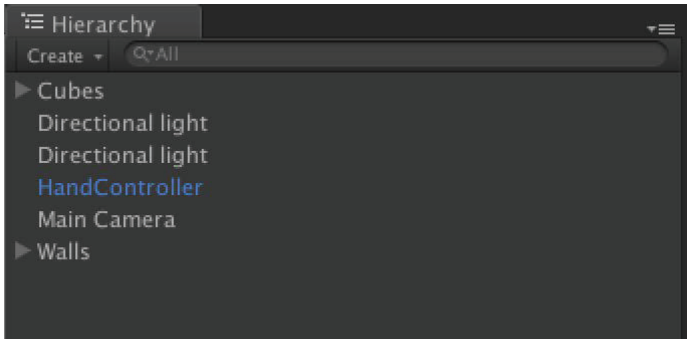
As discussed previously, natural user interfaces have been in use for decades in various contexts and applications. Whether or not a particular interface element is in need of skeuomorphic treatment depends on audience, context, and the purpose and behavior of the interface element itself. It is worth noting that the skeuomorphic properties of direct manipulation dovetail nicely with the goals of immersion and presence – common to virtual reality applications – that we discussed in Section 1.2.

To explore direct manipulation, this course will use some prebuilt Unity3D components. Leap Motion has released a set of hand controllers and models intended to act as a starting framework for Unity3D developers building direct manipulation applications. Not only does the hand controller use Leap Motion

[16] Clive Thompson, "Clive Thompson on Analog Designs in the Digital Age," *Wired.com*, January 31, 2012, http://www.wired.com/2012/01/st_thompson_analog.
[17] Buster Hein, "Jony Ive Explains Why He Decided To Gut Skeuomorphism From iOS 7," *Cult of Mac*, September 19, 2013, http://www.cultofmac.com/246312/jony-ive-explains-why-he-decided-to-gut-skeuomorphism-out-of-ios.

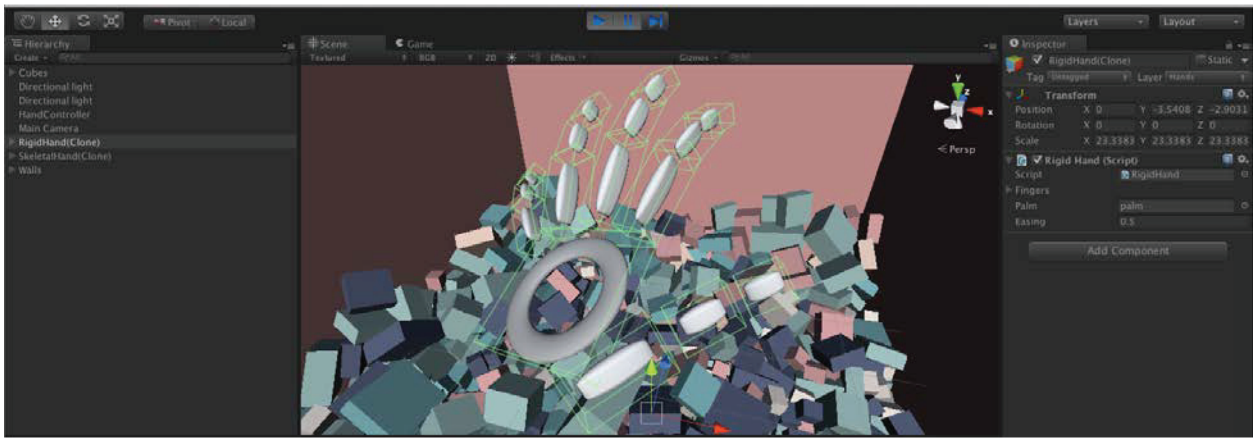
data to drive a visual representation of the hand, it also drives a set of colliders tied into Unity's physics engine. Both the visual and physical components of the hand controller are fully customizable by the end developer.

To begin exploring the hand controller, we will open the Block Pit example distributed with the Leap Motion version 2 tracking SDK.[18] As we can see in the scene hierarchy, in addition to an interaction space and a number of blocks, the scene includes a HandController component.



Cubes
Directional light
Directional light
HandController
Main Camera
Walls

To see how this works, we can place the scene and place a hand over the Leap Motion Controller. A skeletal representation of a hand then appears in the scene and is able to interact directly with the blocks. By pausing the player with our hand still in the scene (sometimes a tricky operation) and switching to Unity's scene, we can examine the structure of the hand.



Cubes
Directional light
Directional light
HandController
Main Camera
RigidHand(Clone)
SkeletalHand(Clone)
Walls

RigidHand(Clone) Static
Tag Untagged Layer Hands
Transform
Position X 0 Y -3.5408 Z -2.9031
Rotation X 0 Y 0 Z 0
Scale X 23.3383 Y 23.3383 Z 23.3383
Rigid Hand (Script)
Script RigidHand
Fingers
Palm palm
Easing 0.5

In this way, we can see that the HandController object spawns two new GameObjects. The first listed is a RigidHand, which contains the generated physics objects and colliders being driven by the Leap Motion Controller data. The second is the SkeletalHand GameObject, which is the visual representation of the hand. As the visual and physical representations of the hand are independent, each can be separately customized or removed, depending on the needs of the end developer.

[18] "Block Pit," *Leap Motion Developer*, accessed June 7, 2014, https://developer.leapmotion.com/gallery/block-pit.

To achieve the physical interactions with the blocks we see in the demo, we simply attach rigid body and collider components to the blocks and the hand. This allows us to push, pull, and sort through each of the individual blocks in a relatively natural manner. But these interactions alone are limited. Developers will find it very difficult to simply reach in and pick up an individual block. The limitations of the physics engine, along with the lack of haptic feedback from a motion controller, stymies these sorts of interactions. We will need to give our simulation a hand. (The author requests that readers pardon this truly horrible pun.) This is where gestural abstraction plays a part.

2.3 Gestural Abstraction

Gestural abstraction represents a very different set of interaction concepts than the previous two techniques. Rather than interacting directly with an on-screen element, gestures allow particular actions or poses of the hand to be mapped to specific in-application commands. Gestures can be both motive or simple poses. For the purposes of this course, we will add a simple "pinch" gesture to our block demo, allowing the user to pick up an individual block.

To enable the ability to pinch and pick up a block, we can write a short script and attach it to the rigid hand. This demonstrates both the ease of using the Leap Motion API and the power of Unity3D's component-based system.

```
Finger index = Finger.Invalid;
Finger thumb = Finger.Invalid;
Vector3 indexPosition;
Vector3 thumbPosition;
Vector3 pinchPoint;
GameObject pinchTarget;

_hand = _rigid.GetLeapHand();

if(_isPinching) {
  foreach(Finger finger in _hand.Fingers) {
    if(finger.Type() == Finger.FingerType.TYPE_INDEX) {
      index = finger;
    }
    else if(finger.Type() == Finger.FingerType.TYPE_THUMB) {
      thumb = finger;
    }

    if(index != Finger.Invalid && thumb != Finger.Invalid) {
      indexPosition = index.Bone(Bone.BoneType.TYPE_DISTAL).NextJoint.ToUnityScaled();
      thumbPosition = index.Bone(Bone.BoneType.TYPE_DISTAL).NextJoint.ToUnityScaled();
      pinchPoint = indexPosition + (0.5f * (thumbPosition - indexPosition));
      pinchPoint = _handController.transform.TransformPoint(pinchPoint);
      pinchTarget = getNearestPinchable(pinchPoint);
      pinchTarget.transform.position = pinchPoint;
    }
  }
```

```
        if(_hand.PinchStrength <= _endPinch) {
            _isPinching = false;
        }
    }
    else {
         if(_hand.PinchStrength >= _startPinch) {
               _isPinching = true;
         }
    }
```

When the script detects a pinch, it finds the nearest reasonable pinchable GameObject (defined with a tag), and adjusts that object's transform to match the movement of the point halfway between the thumb and index finger. When the pinch value reported by the Leap Motion API is no longer within an acceptable range, the pinch is released. This pinch-and-release logic is just one simple example of a useful technique known as hysteresis – the application of a previous application state upon future application logic. In this case, we use slightly different values to define a pinch start versus a pinch end. This creates a dead zone between the two values that reduces the chances of someone accidentally beginning or ending a pinch.

2.4 Combining Techniques

These interaction paradigms need not be used separately. Leap Motion's creative application *Freefrom* uses all three techniques to allow the user to interact with various elements of the interface. *Freeform* is a sculpting application in which users can use direct manipulation to shape the 3D surface of the clay. The application also contains a variety of tools and options, available through a set of radial menus, which are controlled in 2D via a 1:1 mapping of world space to application space. Finally, *Freeform*'s menus can be brought in and out of view via a gesture – the spreading or coming together of the user's hands. Each of these interactions is well-adapted to each particular feature.

Section 3: Understanding and Building Feedback Loops in Interactive Applications

3.1 The Need for Dynamic Feedback

At Leap Motion, we've found that when considering motion control design, it can be helpful to focus more on the transitions between various application states than the states themselves. Due to the analog nature of the interactions inherent to motion control, constant visual feedback is critical to usability. To account for this, designers must reconsider the structure of their visual and auditory feedback. Just as our controls use motion, so must our feedback. We have previously referred to this as *dynamic feedback.*[19]

[19] Daniel Plemmons, "Jumping Head First into Motion Control Design," *Gamasutra*, May 28, 2014, http://gamasutra.com/blogs/DanielPlemmons/20140428/216072/Jumping_Head_First_into_Motion_Control_Design.php

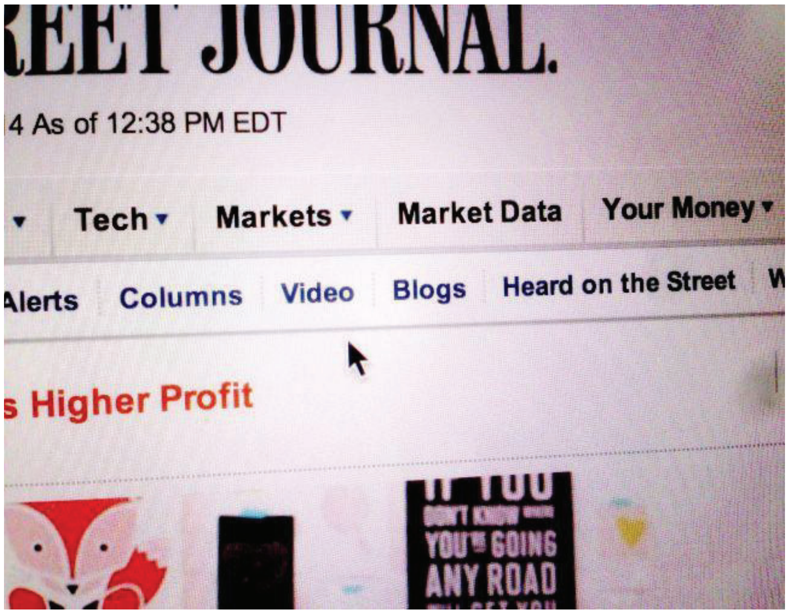
As a user moves their body in the interactive space, the application should constantly respond to their motions – communicating what the interface “cares about” at any given time. This is in contrast to most traditional desktop and mobile design, where the interface only changes when the user directly interacts with the game, because motion control lacks the obvious disengagement states of touch-based interfaces. The nearest design analog on the desktop is hover effects on buttons, and it may help to think of dynamic feedback as “super hover.”

The need for well-designed dynamic feedback is also apparent when we consider the limited set of feedback vectors available to software applications utilizing motion control.

3.2 Comparing and Contrasting Mouse versus Motion Applications

The following is an extended excerpt from Daniel Plemmons’ article “Jumping Head First into Motion Control Design.”[20]

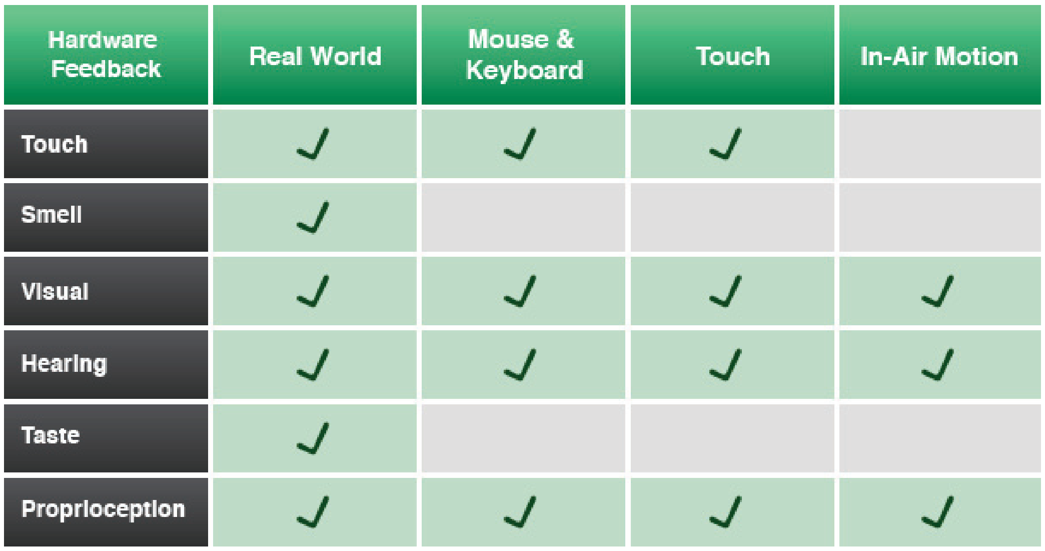
> While motion controls allow for a high degree of freedom and nuance, they lack many of the traditional signals and feedback we’re used to from our hardware input devices. For example, let's compare and contrast the physical and mental processes that take place selecting a button on a web page, with a mouse and with the Leap Motion Controller.
>
> **The mouse version.** (1) First, you put your hand on the mouse. You can feel it and you know the mouse can “detect” your input (tactile feedback). You’ve declared your intent to interact with the computer. You move the mouse along the table. It takes a moment for you to find your cursor on the screen, but as long as that cursor moves when you move the mouse (visual feedback), you know the mouse is working.



Touch, sight, and proprioception all combine to let you move your mouse pointer quickly and easily.

[20] Ibid.

(2) You move your cursor towards the button. The feeling of resistance from the table and your sense of proprioception (where the parts of your body lie in relation to each other) tell you how far you've moved your arm. The cursor on the screen simply confirms your expectations. As the cursor nears your target, your eyes focus on it, letting you correct your exact position. You're not thinking about it, but you're constantly making tiny corrections as you move.

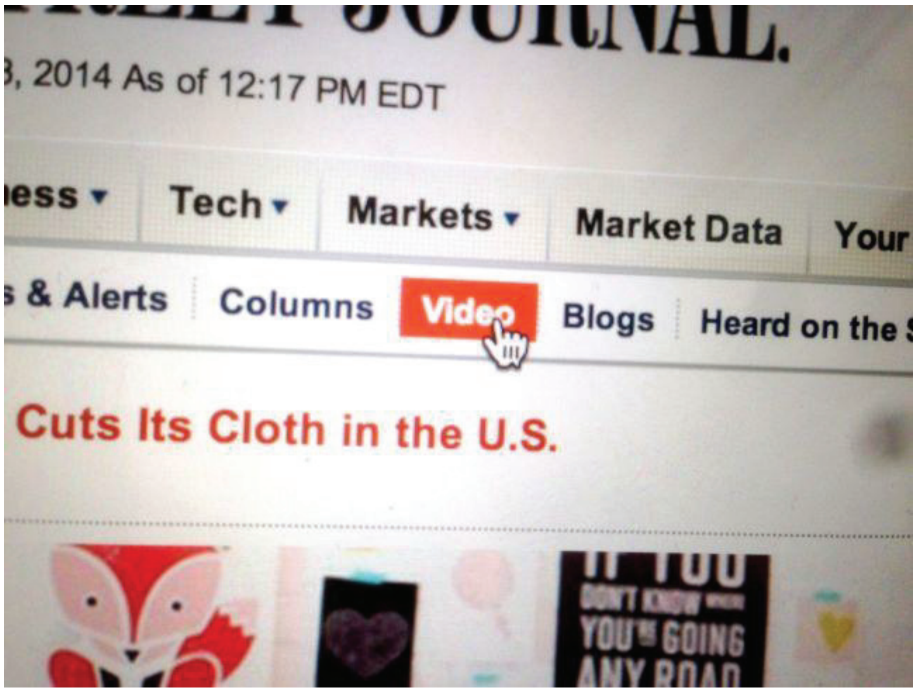


Hardware Feedback	Real World	Mouse & Keyboard	Touch	In-Air Motion
Touch	✓	✓	✓	
Smell	✓			
Visual	✓	✓	✓	✓
Hearing	✓	✓	✓	✓
Taste	✓			
Proprioception	✓	✓	✓	✓

The various feedback vectors available per platform.

(3) Your cursor crosses the boundary of the button, and it highlights (visual feedback).

(4) Your index finger presses down on the left mouse button (or left side of the mouse if you're on a Mac). You feel the resistance of the button and then the reassuring pop as you exert enough force to depress it (tactile feedback). You also hear the ubiquitous "click" sound we're all used to (hardware auditory feedback). You've used this so much you know this means the computer has registered your input. On the screen, the button confirms your input by changing color and/or shading.



Additional visual feedback communicates the system state.

(5) Within milliseconds, your finger releases its pressure on the button, you feel another “pop,” and you hear the second half of the anticipated “click.” The main content area of the webpage flashes white, the button you just pressed transitions from a light background to a dark one, and a small spinner appears next to the name of the browser tab. All this confirms that your input was registered by the website, and it is in fact navigating.

We experience this loop thousands of times per day as we “pick and click” our way through modern desktop interfaces. It takes a 10th of a second, but each piece of feedback is key to the efficient use of the mouse. When a piece of feedback returns an unexpected result, it tells us immediately what’s wrong. Is your cursor not moving? Your mouse must be disconnected, or the computer is locked up. Didn’t feel the button press? You’ve got a broken mouse. Did the button not highlight? It’s probably disabled.

Notice how much of this loop is tactile and auditory. When you’re designing for motion control, your interface must make up for these missing links in the feedback chain. We’re subconsciously aware of a lot of information about the state of our hardware, and the application we control with it. If we’re denied this by a lack of foundational feedback, we conclude an interface is unresponsive, dodgy, confusing, or broken.

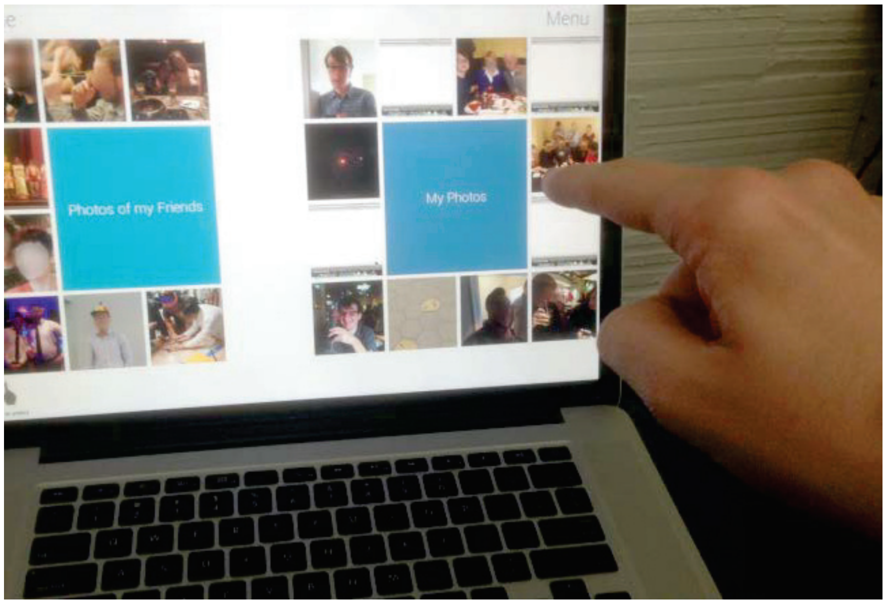
The motion control version. Now let’s take the common motion control version of these events – moving to an item and selecting it. Many applications today like *Photo Explore*,[21] *Touchless*,[22]

[21] Exteroceptive, *PhotoExplore*, accessed June 7, 2014, retrieved from https://airspace.leapmotion.com/apps/photoexplore.

and *Verticus*[23] use an in-air "screen tap" gesture for selection. They use a cursor with dynamic feedback to show the user when they've made a "click." As you read this, it's worth noting that between touch, sight, and hearing, sight is the slowest responding of our senses.[24]

(1) You start by placing your hand in the area you expect the sensor to detect you and point with your index finger. Assuming you're in the right area, a cursor appears on the screen. Just like the mouse, you may take a moment to find it (visual feedback).

(2) You move the cursor towards the button. You're relying on your sense of proprioception and watching the cursor to see when its in the right place. Each motion control application you're using has slightly different calibration, so it's difficult to get a reliable sense of motion.

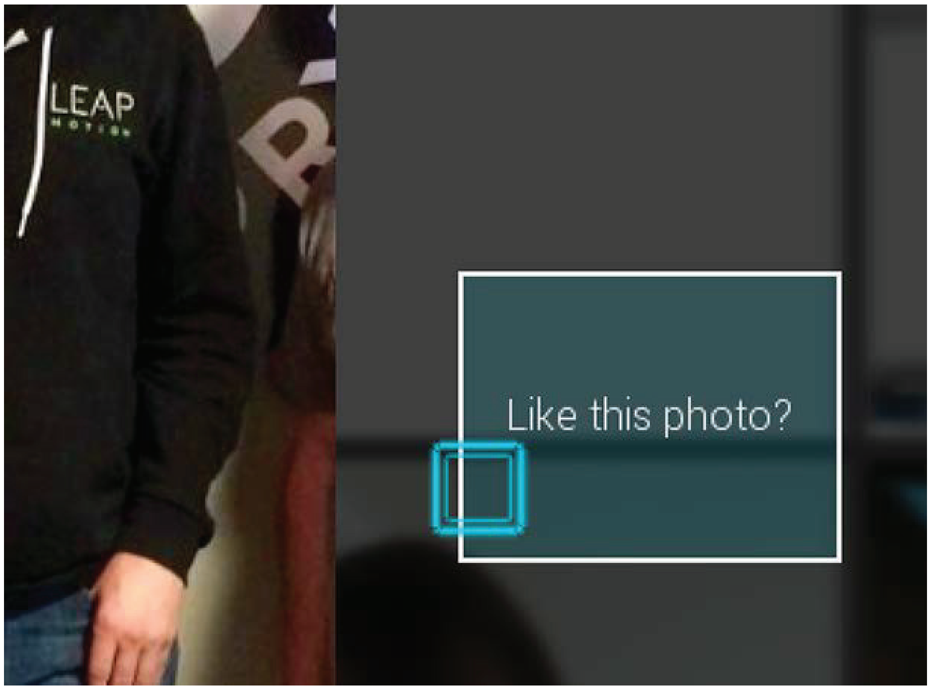


With in-air gestures you rely on sight and proprioception to help guide and steady your hand.

(3) As your cursor crosses the boundary of the button, it highlights – telling you it's an active interface element. You hold your hand steady in the air over the button. It's relatively large, so it's not too hard.

(4) You push your finger forward, watching the cursor to make sure you keep your finger steady pointing at the right item, making small adjustments as you push forward. As you move forward, an inner circle on the cursor grows to meet the outer circle, signalling a "click" (visual feedback).

[22] Leap Motion, *Touchless For Windows*, accessed June 7, 2014, retrieved from https://airspace.leapmotion.com/apps/touchless-for-windows.
[23] Moonshark, *Stan Lee's Verticus*, accessed June 7, 2014, retrieved from https://airspace.leapmotion.com/apps/stan-lee-s-verticus.
[24] Robert J. Kosinski, "A Literature Review on Reaction Time," *Clemson University Department of Biological Sciences*, last modified September 2013, http://biology.clemson.edu/bpc/bp/Lab/110/reaction.htm#Type%20of%20Stimulus.



Dynamic on-screen feedback is critical to communicating system state.

> (5) When the two circles meet, the main content area of the webpage flashes white, the button you just pressed transitions from a light background to a dark one, and a small spinner appears next to the name of the browser tab (visual feedback). Again, this confirms that your input was properly registered. You drop your hand, relaxing the joints.
>
> This flow seems quite useable, but challenges crop up when something along the line doesn't work properly. What if you don't see your cursor? Is your hand simply too low or is it too far to the right or is the device not working? What if when you push your finger in to "click" and the click doesn't happen? Are you performing the gesture wrong? If so, how? What if the website doesn't take the click? Are you gesturing wrong or is the site at fault? Does this website even support this motion tracker?
>
> It's up to the developers of motion control software to provide users with the answers to these questions. This is where constant dynamic feedback can be a very useful tool. Don't underestimate the value of good audible feedback either. "Pops" and "clicks" can lend a sense of physicality and don't require your player to be focused on any individual on-screen element to be useful.

At Leap Motion, we explored the usability impact of even a minor implementation of this dynamic feedback, with encouraging results:

> In an early prototype of the marching menu, we used static icons to inform users what sorts of actions they could perform. However, users often didn't associate these static icons with actions. Some people tried tapping, others pinched, and others were just plain confused. When users did make a selection, sometimes they didn't know how they'd done it.

For the next iteration, we added some simple feedback, emphasizing the X-axis location of the user's finger relative to the currently selected button. In testing, users picked up on what they were supposed to do quite quickly, but were often uncomfortable flinging their finger off a menu item to select it. There was no indication of what would happen if they did.

With a little added polish though, the new users we tested with understood and started using the menus with ease. We saw a real marked improvement from some very small changes – design elements that reveal the menu's behavioral structure at a glance. This sort of basic feedback is great for all manner of menus.[25]

3.3 Heuristic Lenses for Motion Interaction Design

When designing interactions and gestures for an application, the teams at Leap Motion have used a variety of strategies to design and iterate on our designs. From this has arisen a set of critical heuristic lenses that we have found are useful for assessing gestural and motion interaction designs.

1. **Tracking Consistency.** How consistent is tracking, given one or more users performing this motion many times in a real-world environment, and given that production applications need 95%+ accuracy?

2. **Ease of Detection.** From a production and maintainability perspective, how difficult is it to detect this motion with a high degree of accuracy across many users, both in terms of false positive and false negative results?

3. **Occlusion.** Given the limitations of the device, does this interaction have a high chance of causing occlusion? (For example, an interaction in which a user may reach a hand across the field of view of the device has a high chance of causing the arm or a shirt sleeve to occlude a majority of the hand.)

4. **Ergonomics.** Given the limitations of the human body and the intended use-case and environment for the interaction, what are the ergonomic concerns? Can someone perform this interaction while relaxed, or does it create unnecessary tension? Does repeated use become stressful? Does the interaction require a vastly unnatural motion (e.g. trying to poke straight ahead in Z-space while our hands want to move in arcs)?

5. **Transitions.** Given the complete interaction set in an application, how does this interaction transition to others? Are the transitions clear and easy to learn? How easy are they to detect? What happens in the case of ambiguous transitions? Each transition could easily be re-evaluated with this same set of heuristic lenses.

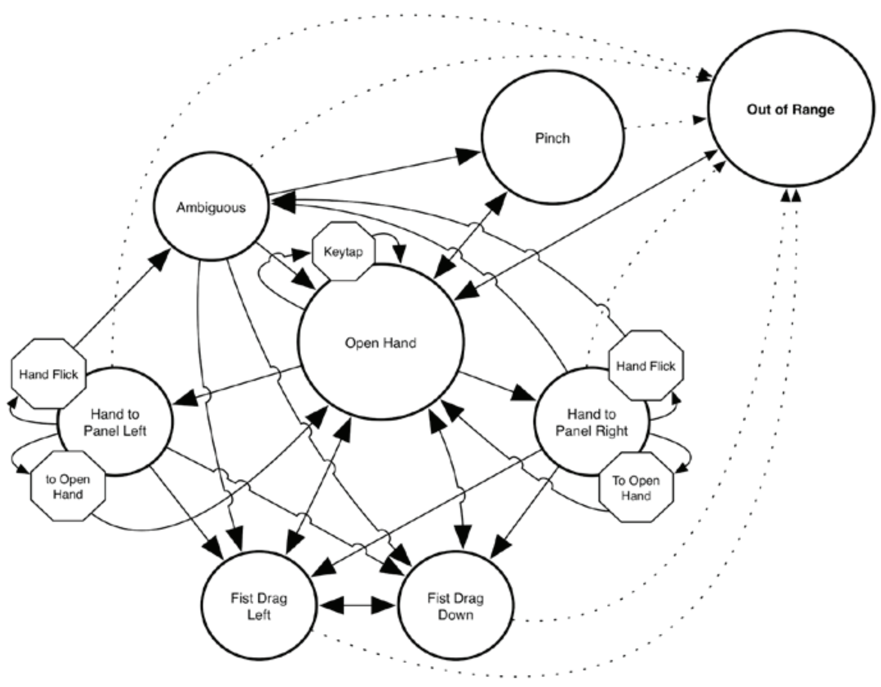
[25] Daniel Plemmons, "Rethinking Menu Design in the Natural Interface Wild West," *Leap Motion Blog*, accessed June 7, 2014, https://www.leapmotion.com/blog/rethinking-menu-design-in-the-natural-interface-wild-west.

6. **Feedback.** Does this interaction lend itself well to having good feedback from the UI? As we have seen in many of the above references, a lack of proper feedback can sink an otherwise well-designed interaction.

It is important to note that these heuristics exist as lenses through which to critique and examine an interaction, not as hard and fast rules.

3.4 Getting the Most Out of Gesture Space

Detection of gestures and analog motions is inherently fuzzy. To pack the most functionality into a single application, an application can manage and modify its gesture detection and actions based on the current state of the application. Limiting the number of possible transitions at any one time, and applying a critical lens to the transitions possible from each state, an application can significantly reduce the number of interactions that can damage the end user experience, in line with the classic Nielsen heuristic of error prevention.[26]



The above diagram shows an example analysis of the possible motion or gesture transitions in an application. Given each possible set of transitions, gesture detection code can be modified for the specific cases, better separating gestures from each other as well as reducing the probability of false negatives.

[26] Jakob Nielsen, "10 Usability Heuristics for User Interface Design," January 1, 1995, http://www.nngroup.com/articles/ten-usability-heuristics.