Integrating Leap Motion with Autodesk® AutoCAD®

Kean Walmsley – Autodesk

DV1442: There is a great deal of interest in how the Leap Motion™ Controller is going to change the way we interact with 3D modeling and visualization tools. This class focuses on the steps that are needed to define a simple, gesture-based user interface to Autodesk AutoCAD software using the Leap Motion SDK. We look at a way to enable AutoCAD users to navigate 3D models using gestures, as well as different techniques for allowing geometry creation and manipulation. We take a look at the relative pros and cons of gesture-based interaction with 3D systems in an attempt to assess whether this mode of interaction is likely to prove popular in our industry.

Learning Objectives
At the end of this class, you will be able to:

- Describe how Leap Motion can be used to control software applications
- Use the capabilities of the Leap Motion SDK
- Integrate Leap Motion into AutoCAD using C# or JavaScript
- Explain how this technology can be applied successfully to 3D modeling

About the Speaker

Kean has been with Autodesk since 1995, working for most of that time in a variety of roles—and in a number of different countries—for the Autodesk Developer Network organization.

Kean’s current role is Software Architect for the AutoCAD family of products, and he continues to write regular posts for his popular development-oriented blog, Through the Interface. Kean currently lives and works in Switzerland.

kean.walmsley@autodesk.com
Introducing Leap Motion

History

The Leap Motion controller was announced in May, 2012. The excitement – and arguably marketing hype – that surrounded the initial announcement was intense: people were thrilled by the possibilities presented by this device.

This rationale can be found on the Leap Motion web-site:

How we came to be: A brief history of the future.

It all started when one of our founders, David, was 3D modeling. What took 10 seconds by hand often took 30 minutes with a computer. Technology was actually getting in the way of technology. Fortunately, he’s a super-smart mathematician who figured out an incredibly brilliant way of tracking natural movement down to the fingertip. So now we really can mold pixels as easily as clay.

It’s clear than much of the motivation for this technology has been driven by the design industry. In this session we’ll take a look at how this potential seems to be playing out, and get a sense for the near- and longer-term applications for the technology.

The Technology

The Leap Motion controller is a sleek device with a Gorilla Glass upper surface and brushed aluminum casing. It measures 80mm (length) by 30mm (depth) by 11mm (height) and has the feel of an Apple product, perhaps understandably given the background of members of the company’s executive team. The device is being sold for $79.99 in the US.

It connects to your PC or Mac via a USB 2.0 cable and requires a runtime component (including a device driver) to be installed in order to work.

The controller contains three infra-red LEDs that generate a pattern of dots of IR light (similar to the way a Kinect 1.0 the device uses structured light to detect depth). It works at an incredibly high frame-rate of 300 fps, gathering data to be analyzed in software by the device driver (which is where all the cleverness happens, apparently).

The device works incredibly accurately in a hemispherical volume of around 1 meter/3 feet in radius, so it’s clearly currently suited for capturing close-range hand gestures by a single person.
Getting to Know Your Leap

The best way to get a feel for the type of data your application can receive from a Leap Motion controller is to run the visualizer app. There are two visualizers available with the software: one is for show and one if for genuine diagnostics.

The first can be launched directly from the Leap Motion status bar (whether on a Mac or a PC):

This brings up a window that should display a simplified view on data coming from the device, albeit with fancy physics effects:
It’s interesting to note that this menu also gives access to the AirSpace store, which is Leap Motion’s App Store. You’ll find a number of titles available here, including at least one from Autodesk:

The more useful visualizer is available from the Leap Motion control panel (accessible via the Settings option on the status bar menu):
Clicking Diagnostic Visualizer will bring up the advanced visualization dialog:
This allows you to get some very useful data on what’s coming from the controller – I strongly recommend playing with the various options when considering how to implement your gesture-based UI.
The Leap Motion SDK

A number of different language bindings are available to Leap Motion developers:

- C++
- C# (also for Unity)
- Objective C
- Java
- Python
- JavaScript (https://github.com/leapmotion/leapjs)

There are some common concepts that cross these various language bindings:

- **Controller**: interface between Leap and the application
- **Listener**: used to handle events dispatched by the Leap
- **Frame***: contains a set of hand and finger tracking data
- **Hand***: contains tracking data for a detected hand
- **Finger**: contains tracking data for a detected finger
- **Pointable***: either a detected finger or a detected tool
- **Vector**: represents a 3D position or directional vector
- **Gesture***: represents a recognized gesture

* These also apply to the JavaScript implementation, which is otherwise somewhat simplified.

There are samples provided with the Leap Motion SDK that demonstrate how to interface with Leap Motion from each of these environments.

The general approach – for languages other than JavaScript – is to implement a custom Listener class with overrides for the various messages: OnInit(), OnConnect(), OnDisconnect() and OnFrame(). The most important being OnFrame(), as that’s where your application will receive information on what’s happening on a per-frame basis.

This Listener needs to be instantiated and used to create a Controller object. When the event handlers in the Listener are called, they have the Controller as an argument. This object provides access to the various objects representing the things detected in the scene, whether Hands, Fingers, Pointables or Gestures.

As mentioned earlier, JavaScript – while basically comparable – is a bit simpler: you just call Leap.loop() with the callback function you wish to be executed each time there’s a frame to process. LeapJS works by listening to a WebSocket broadcast on a particular port: the Leap Motion controller sends JSON packets out describing the tracked objects and detected gestures.
Integrating Leap Motion Using C#

We will focus on using Leap Motion from C# and JavaScript, starting with C#.

After installing the SDK, which can be downloaded from the Leap Motion web-site, you will need to add a project reference to the appropriate assembly for your targeted .NET runtime version, whether LeapCSharp.NET3.5.dll or LeapCSharp.NET4.0.dll. These can be found in the lib folder of the Leap Motion SDK.

You’ll also need to copy across a few additional DLLs from the appropriate sub-folder (whether x86 or x64) into your application’s folder: Leap.dll, Leapd.dll & LeapCSharp.dll.

At this stage you’re pretty much ready to start coding. We won’t step through the code here, but do take a look at the information in this blog post and download the sample project from here.

Strategies for Integrating Leap Motion with AutoCAD

There are a number of approaches possible for integrating Leap Motion into AutoCAD. I’ve listed the main buckets here with my own labels assigned.

**Generic**

The first approach I chose to implement to get Leap Motion working inside AutoCAD was completely generic in nature. I simply moved the standard system cursor whenever the user’s hand was moved in the capture volume. I then implemented a simple check for independent acceleration of one of the digits on the hand. If one was accelerating faster than the hand itself, I considered that to be a “click” and would simulate a click of the left mouse button. Very simple.

This approach works – and is a straightforward way to integrate Leap Motion with any Windows app – but it’s fundamentally unsatisfying. It doesn’t raise the bar over what’s possible with a mouse (and is, in fact, a very basic mouse replacement).
In the sample project two commands have been added to manage this mode: LEAPMSGS enables it, will LEAPMSGXS turns it off. The mode is independent of a command, which is useful.

**Specialized**

The next approach I took was to implement a specific command that waits for gestures to be performed and then calls into AutoCAD’s API to control the view, etc. I’ve called this specialized, as it’s a way to integrate in a much more specific way. It takes more effort but is ultimately a much more interesting approach.

The sample provided has a LEAP command for this mode. It runs a jig that waits for the user to make hand movements and gestures. Basic lateral hand movements will pan the current view, vertical movements will zoom in or out, tilting of the hand will rotate the view in one of four directions (left/right/up/down).

If a single finger is detected then a spherical cursor is displayed using AutoCAD’s transient graphics mechanism. This is made easy as the whole processing loop is inside a jig, allowing visual feedback to be provided to the user. This sphere is centered on the point at which drawing will occur if the “s” (for spline) or “p” (for polyline) keys are pressed. Yes, I chose not to use gestures for that.

The user can then draw in 3D, rotating the view with hand gestures, as needed.

**Optimized**

It’s of course possible to consider a much more integrated approach, where there isn’t a special command needed to consider input from the Leap Motion controller. The user might be able to manipulate objects in 3D, for instance, picking them up and moving them around.

This, however, would involve a significant re-architecture of AutoCAD, and is way beyond the scope of this particular prototyping/research project.
Implementing Gestures

Leap Motion supports a number of basic gestures “out of the box”. It can detect swipes, taps and circular movements, for instance. Test them out with the Diagnostic Visualizer to see how they work.

For the two approaches I developed initially – the Generic and Specialized items, above – I essentially implemented my own fairly crude gestures. I’ve tried to keep things simple by mostly using full hand movements. Fine-grained gestures that depend on specific fingers are tricky to implement for a number of reasons. Fingers aren’t always tracked perfectly – they may be occluded by the palm, for instance – and it’s tricky/dangerous to make assumptions about how finger IDs are assigned and maintained.

The third approach that I implemented, which I’ll call Specialized+, does make use of circular gestures, for instance.

Over time I expect the array of gestures that are available to developers to expand, whether by Leap Motion directly or 3rd party gesture libraries.

Findings

The Leap Motion controller is extremely accurate and responsive: the developers have done an impressive job delivering on that promise, with a high frame-rate and low latency. The technology can certainly be used to perform intricate operations, and I fully expect it to be used in niche scenarios such as the medical field where using standard input peripherals is problematic.

The precision itself is not as interesting for engineering design tasks, for instance: just as with the early days of digitizing tablets, grids, snaps and numerical input will still be needed needed to have perfect accuracy. But there are certainly creative fields where expression trumps precision, as with clay sculpting applications. This is one of the reasons the Maya plug-in was one of the early ones to be delivered by Autodesk.

While heavy use of a mouse and keyboard can often lead to repetitive stress injuries, it’s not clear that using a poorly-designed “natural user interface” (NUI) is much better. Hovering your hand a few inches from a tabletop is hard, especially without haptic feedback.

Some things might mitigate this issue: having the Leap Motion controller in a vertical position, scanning your fingers on a tabletop, or placing a glass surface above the device and using that as a touch surface are just two possibilities.

Right now we’re at the beginning of a journey, one that I expect to continue as this and other gesture-based input technologies iterate quickly. It will certainly be interesting to see how the field evolves over the coming years.