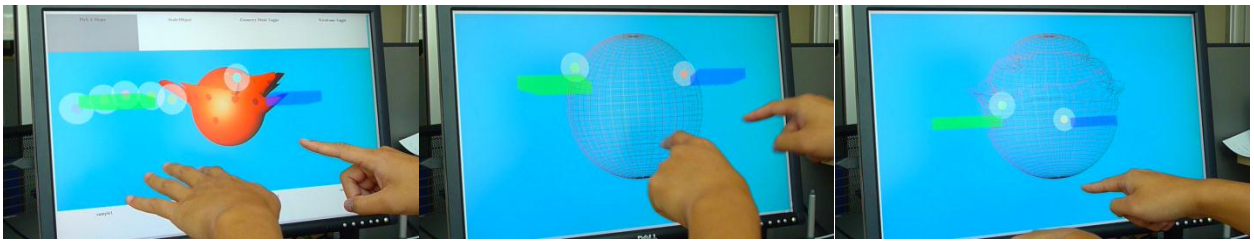
Sculpting in Real-Time Using a Leap Motion

Thinh Nguyen, Jürgen P. Schulze, Kiyoshi Kiyokawa

University of California, San Diego; Osaka University



Abstract

We investigate the suitability of the Leap Motion device for high precision activities such as sculpting. The system lets the users choose between a variety of initial polygon meshes and allows the user to interactively make changes to the shapes as they choose. Our approach uses only the position and orientation of our hands to navigate through menus and perform sculpting tasks. Created using WebGL and HTML5, the application is also able to run on any operating system without the need to use executable files.

Introduction

Sculpting in the real world is one of the most full hand interaction activities and to imitate sculpting using the leap motion seemed intuitive as it is one of the newest consumer friendly devices to allow potential users to use both of their hands as a user input device without wearing anything cumbersome. We made a 3D engine from the ground up to let users test out the leap motion as a way to feed hand and finger data to the application.

Previous Work

Previous work on sculpting applications involved some sort of selection of the sculpting region, as well as a way to enable the mesh manipulations. This usually involved some sort of button to enable mesh changes. Different levels of sculpting software were designed; some for consumer use such as 123D sculpt and TrueSculpt - A large majority of which tended to use a flashlight selection scheme to select play areas.

In contrast to this we used a ray casted selection approach to select the play area and distance factors for enabling mesh manipulations. In contrast to the normal keyboard and mouse routines, the user is able to navigate menus and manipulate geometries without the needing to use any other peripheral device. The user also gets a much more one to one selection scheme with the use of the fingers as a way to select things than a normal mouse cursor.

Leap Hardware

The Leap Motion controller is a small USB peripheral device which is designed to be placed on a physical desktop, facing upward. It uses two monochromatic IR cameras and three infrared LEDs. The LEDs generate a 3D pattern of dots of IR light and the cameras generate almost 300 frames per second of reflected data, which is then sent through a USB cable to the host computer, where it is analyzed by the Leap Motion controller software. It generates 3D position data by comparing the 2D frames generated by the two cameras, presumably with some sort of edge detection algorithm. It is designed to track fingers (or similar items such as a pen) which cross into the observed area, to a spatial precision of about 0.01 mm. From online demos it is presumed there may be some point cloud data of hand depth, but it is not in any current consumer APIs.

Leap Input

The leap motion provides a frame packet that contains data on the hands present in the field of play above the leap motion. The packets hold things like hand position, hand direction, finger position, finger directions, and basic gestures. There are seemingly no limits to the number of hands that that leap can detect, however in the application we are only expecting the users hands to be detected to function properly. There is also no built in way to distinguish left and right hands so we implemented. The leap can detect basic gestures like horizontal swiping, spins, relative amount of hand clenching, and keytaps. The leap however does not detect finger or hand objects occluded vertically by any objects, which will make fingers objects occasionally flicker or hand objects to suddenly disappear from time to time.

Selections (Mesh)

Our system uses ray casters on each finger object to do selections. The direction of the ray caster is based off the direction that we get from the leap data about the finger. The origin of the ray caster is placed at the relative positions of where the finger objects are seen. Selection is done by checking whether any of the rays intersect with the bounding box of the current

mesh object; if so, find out which surface the ray caster is intersecting and send it to the manipulation handler.

Mesh Manipulations

We use the relative distance between the finger object and the collided mesh to do mesh manipulations. When the finger object gets within a certain user defined threshold, the mesh will conform based on what sculpting mode it is in. Mesh manipulations are also only done with the right hand's finger objects, while the left hand's fingers serve as a way to enable the menu and perform simple gestures to move the camera around. The mesh will either be pulled or pushed relative to the direction of where the raycaster was. The mesh will be pulled towards the position of the raycaster, or pushed away in the direction of the raycaster.

Camera movement is done via left hand gestures. Simple spinning finger gestures will rotate the camera based on the direction of the spin (counter clockwise will spin the camera to the left, and clockwise spins will spin the camera to the right). Finger swipes also will move the camera based on the direction of the swipes done. The relative speeds of the camera movement can also be changed by the user by putting in their own values for rotation speeds.

Geometry view and sculpting modes are also able to be toggled using the left hand as well. Screentap gestures with a left finger will cause the geometry to go into a wireframe mode, while a screentap gesture with a right finger will cause the geometry to go into a fully rendered mesh mode.

Menu Navigations

Selection for menu items is done by a relative direction of the ray caster (as opposed to checking for collisions in the mesh handler case). Choosing of individual menu items is done using a single right finger as a selector ray. The finger (usually an index finger) can choose between eight different menu items at one time. The menu items are fully selected after leaving the finger on the menu item for a three second timer.

Opening of the menu items are done using the left hand. The open palm of a left hand object will cause the menu to appear and stop all mesh manipulation routines. The open palm will keep the menu up, unless you fully finished going down a menu item path. Closing the left hand will immediately close the menu popup if you were not already in the middle of going down a menu item path.

The menu is done using an HTML5 overlay over the WebGL canvas object. The menu is fully customizable to put whatever the user wants to quickly access or toggle by changing the HTML headers around.

Visual Selection Aids

We used a number of visual selection aids to help the user accurately pick the surfaces that they would want to manipulate. This is a larger problem without the use of any buttons, and that the users tend to not have completely still hands while using the leap motion device.

Basic shadows were used to give the working mesh a better feel for height. The shadows also helped show the user relatively how close the fingers were to the object, by giving it another field of depth. Shadows were done using a basic shadow mapping from two light sources, which are able to be changed in the code. The user can also change the values of the shadow intensity, as well as store a larger or smaller texture for the shadow map.

A basic silkscreen around the finger objects was used to allow the user to know whether the finger objects were colliding with any nearby objects. The silkscreen was done by having a semi-transparent sphere around the finger object, which did not cast any shadows. The screen is essentially a wider bounding sphere around the finger objects that warned the user when the finger objects were about to manipulate any geometries, since the silkscreen sphere was the initialized to be the same size as the manipulation distance for the finger ray casters.

Extended surface color mapping was also used to show which part of the working mesh would be altered if the user would continue moving their fingers towards the working mesh. The color of the potential play surface would increase in white-color intensity as the user would get closer to the manipulation distance. The further away the finger objects were to the working mesh, the darker the intensity would be on that potential play surface.

Results

The application provides a way for anyone with a leap motion device to quickly play around with and manipulate 3D meshes. Compared to other existing 3D sculpting applications that use the traditional mouse and keyboard there is less need to learn convoluted menus to move the camera around. By using a 3D input device there is also a more inherit one to one relationship with the selection using finger objects rather than the mouse cursor that only moves in a 2 dimensional plane. Allowing selections using a 3 dimensional space overall is more intuitive, but has less accuracy as the hands have to be floating in midair as opposed to a mouse that can be easily set still.

The application provided a way to quickly manipulate 3D geometries in an intuitive way, however the lack of precision caused by that increased speed also went up. The lack of the leap motion's ability to detect vertically aligned fingers also caused problems with some aspects of sculpting (like sculpting the sides of a pot for example). The manipulation algorithm also does

not create new geometries when the working meshes are more skewed, causing it to not be able to append small and precise structures to the image.

The speed of the application can also vary depending on the graphics capability of the machine it is currently running on. The application requires a decent graphics card to process more complicated geometries with higher densities of triangles. The application can however run the sphere as a beginning mesh for a large number of low powered machines.

The leap device also has no inherit way to keep track of preexisting hand objects, which causes a lot of flickering in the data. IDs of hand objects are not preserved upon lost of a hand object, which will occasionally cause flicker. The leap device will also occasionally detect objects that are not hand objects and say that there are more hands in the scene than there really is. (Things such as faces for example). This causes problems in this application which is expecting only two hands in the scene, one left hand and one right hand, as most.

The user also tends to have a lot of hand fatigue when using the leap motion for extended periods of time as well. Holding up your hands for a long period of time causes muscle strain and ultimately causes shaky hands which will make precise sculpting much more difficult to do.

Conclusion

There can be more work done, specifically in the mesh manipulation algorithms. By allowing more vertices and faces to be generated while the application is running, there can be much more precise sculpting. The application can also be altered to use an actual 3D display which would help the user sculpt more precisely as well. There are also many open menu items available for use.

The ability to save the final sculpted model to a file to the user's computer is also lacking, which would give more incentive for people to use the application as well.

Future work can also be done on having a multi-user experience in collaborative sculpting of a 3D model, since the application is already run in the web browser using WebGL and HTML5. The application can be run on a server and the users would be able to make 3D models with the friends without the need to install any more software than just the leap motion device drivers.