Interface with 3D Data by Touchless Control via a Leap Motion Controller

Victor Geislinger

December 10, 2013

Abstract

Using python scripting, we have created a user interface with the Leap Motion Controller to manipulate 3D data via Paraview. We implemented four different actions controlled solely by a touchless interface. These actions are moving the data set in the view, performing a 2D slice of the data, rotating the view and scaling the view by zooming in or out. We also combined the actions slicing, rotating, and zooming together so that they can be used at any time by the user. The end result is a seamless control of interacting with 3D data via a touchless controller.

1 Introduction

1.1 Background and Motivation

As we develop more powerful technology, 3D data is becoming more prevalent in everyday fields. From scientists exploring the ocean floors to doctors looking at a patient's brain scan, 3D data can be useful in a variety of fields. As more people need to interact with 3D data, there is a call towards more intuitive interfaces. This is because it can be difficult for a user to interpret a 2D interaction of a mouse in a 3D space. Touchscreen controls have been an improvement since their motions tend to be more intuitive. However, there are limitations such as the cost of a larger screen to interact with and the fact that you're still restricted to a 2D plane.

This is where 3D touchless controls can be very useful. By allowing a user to interpret their 3D motions and actions into the 3D space their data lies within, you can create more intuitive and natural interfaces. We want to be able to have precise actions, but we also don't want the controller to be widely available and not too expense. There are a few of these touchless controllers in the market, but the Leap Motion Controller seems to best fit for this situation. Costing about $80 USD at the time of this writing, it can track fingers up to 0.01 of a millimeter.[1] It also focuses purely on hands and fingers which is what we would want for a versatile controller for a variety of fields.

We use Paraview, an open-source, multi-platform data analysis and visualization application, as our basis of interacting with the data.[2] There is a pre-compiled version of Paraview that includes a graphical user interface in which you can perform a variety of actions. We will only be looking at a handful of these functions to incorporate with the Leap Motion Controller. We will specifically be using the python scripting features of both Paraview and the Leap Motion's SDK to interpret hand motions to interactions with 3D data.

2 Methodology

In designing the touchless control actions, we must consider what will be useful and intuitive. We may want to take some form of controls that already exist for the mouse and touchscreen interface, but we must ensure that it functions well for the touchless control. We aim for a seamless, and natural interaction where a user will feel comfortable and not bogged down by the interface. Instead we want the interface to become intuitive where any user from any field can use the controls easily.

2.1 Gestures

Gestures have begun to be more common place since the wide adaption of touchscreens. The success of touchscreen is arguably because people feel very natural using their hands, more so than using a tool. This motivates us to also use a hand interface to enforce a natural and intuitive control. However, just because we are using hands as the interface it doesn't mean it will be intuitive or even comfortable.[3, 4] We must be careful then on how we define these controls so that a user does not become frustrated by the interface.

Since we are moving towards a natural control system, we also want to ensure that the controls are the least arbitrary we can make. This means we will want to avoid using a gesture that doesn't fit with the action desired and only makes sense in this application. However, we can use other gestures from other interfaces that the user is likely to be familiar with. For example, we will define a zooming action that will be somewhat related to the touchscreen equivalent. Although this action is arbitrary, it most likely familiar with a majority of users and the difficulty of the transition from touchscreen to touchless controls should be minimal.

3 Functions

There are four functions that we have implemented in manipulating a given three-dimensional data set. They are taking a slice of a data set, zooming in from a user's view, rotating a user's view and moving a given data set. We also implemented a script that allows the user to switch between slicing, zooming and rotating functions based on the hand input given.

3.1 Moving

Moving an object in 3D can be difficult. In Paraview, one must actually enter in values for some center and then render the data. One could see where a mouse could be used to move that data, but one can then quickly realize the limitations and the unnatural control. This is where the Leap Motion Controller with its touchless interface can be very intuitive. This is because we are controlling dat in a 3D space via a controller also in 3D space.

Here we have very simply map the SDK's position of a single hand as the center of the object. We have scaled the movement so that it would fit in the screen. We also defined the origin of the control as right above the Leap Motion Controller, but this choice of origin was arbitrary and can easily be defined somewhere else.

3.2 Slicing

Viewing slices of a given data set can be especially useful when considering three-dimensional data. For example, a brain scan may be presented as a 3D data set but 2D slices of that data can be more useful and give vital information for a particular application. Mouse interfaces can be in particular relatively unnatural and difficult. This is because with this type of control, we are attempting to interact with a 3D space via a 2D-bounded tool. This is where the Leap Motion Controller is clearly superior in the sense of intuitiveness.

We begin by using Leap Motion's SDK to identify a hand's palm and give it a vector normal to the surface of the palm. It should be noted that this is easiest to define when the palm is as flat as possible and the normal vector does not lie almost completely in the xz-plane of the controller or any parallel plane. We can then map the normal of a hand in the controller's view to the normal of the slicing plane in our view. One may notice that this provides an ambiguity to whether we use the normal coming out from the palm or out from the back of the hand. Although this ambiguity exists, it is not important to this application of slicing since both above and below the plane are identical in the sense of slicing.

With this interpretation of a hand to perform a slicing function, the user is able to interact with the data in all three dimensions naturally. Compared to a mouse interface, the Leap Motion Controller is very intuitive as a user's hand is interpreted as the slice itself. This method of slicing also has some benefits over the precompiled Paraview application. With use of the Leap Motion controller, the slice is rendered immediately with no visual lag. However in the Paraview control, the user must move the slicing plane and then apply it so the user does not get an immediate rendering.

3.3 Rotating

Rotating a view will not modify the data as the moving function did, nor will it present new data to the screen as the slicing function did. However, it will allow us to view the data differently by rotating the user's view. The Paraview application has the ability to rotate the view by a mouse click-and-drag. As with many mouse interactions, this is not very intuitive and natural. This motivates a touchless interface for rotation.

We begin by starting with an intuitive gesture, turning an object. We can capture this through the Leap Motion's SDK by comparing a single hand's position from one frame to the next. We need an aspect of the hand to focus on, specifically a vector that we can track rotating between frames. We choose the palm's normal vector since we already use it for the slicing function. Now we can take the SDK's definition for the angle of rotation of the normal vector and translate that as the view's angle. Here we have only focused on rotating the view about the camera's axis, but the concept can be expanded to rotate about all three axes of the view camera.

This touchless function is more intuitive since it is like we are rotating the object within our hands. It should be noted that the data itself is not being rotated, it is the camera view's angle itself though it might feel that the data is being rotated.

3.4 Zooming

Viewing data either closer or farther away maybe beneficial in some instances. This can be done in the Paraview application by either specifying the zoom percentage or using the scrolling function of a mouse. The second function is usually more intuitive, so we would like to implement something that is similarly natural for the touchless control.

We first take a look at touchscreen zooming gestures. It is nearly universal on a touchscreen that when touching two fingers against the screen and moving them away from and toward one another zooms in and out on the image respectively. This exact motion cannot be used reliably with a touchless control since this would result in zooming in and out as the user returns to his or her initial finger position. The user would have to the hand outside the view of the controller, reposition the fingers and then reenter the viewing area. We instead defined something similar but instead of using two fingers, we use two hands.

In the python script we activate this function when two hands are present and both have at least two fingers present in the view. When the user moves their hands away and toward one another, the view will zoom in and out respectively as it similarly does with touchscreen controls. We do this by using the SDK's definition of "scaling" where it compares positions of the hands in the current frame and the previous frame.

The careful reader may realize that just by replacing fingers with hands, we still have the issue of zooming back in or out when the hands return to their initial positions. This is true so we really need a 3D equivalent of not touching a touchscreen. One might think to just remove your hands from the view but the same problem still remains. Instead we define an "empty" function where the view will not change regardless of the position of hands. In this application, we define two closed hands (no fingers) as an empty function. This now allows the user to zoom in or out, then close his or her hands and no longer update the view. The user can then remove their hands from view or reposition them and begin to zoom in and out again by presenting 2 or more fingers on each hand.

This zooming function now allows the user to naturally scale his or her view, possibly comparable to a touchscreen interface. This doesn't have a clear advantage over the scrolling of a mouse in the Paraview application. However, it will be useful when wanting to switch from different actions to perform with a touchless interface as we will later show.

3.5 Simultaneous Use of Slicing, Zooming and Rotating Functions

So far we have looked at the four functions acting on their own. We would like to switch from one function to another seamlessly as possible. We could have the user press a key on the keyboard or click a button with a mouse. But these options would break the flow of using the touchless controller. We could also have something like a button click with the touchless control, or even an equivalent to a key press by using some single gesture that would signal to switch function modes. Although this allows us to remove the mouse and keyboard from the control system, it still isn't very smooth and/or natural. Instead we defined different hand shapes that are already used for the functions. This means to use a function, a user simply changes his or her hand to whatever is used for the desired function allowing for a natural flow from one function to another.

We created a script that uses the slice, zoom and rotation functions. We have kept the functions touchless controls similar to, if not the same as, the controls that were used when they were the only function available. Slicing is activated when only one hand is present and is controlled with the hand's palm's normal vector, exactly as described in the previous section. The zooming function is activated when two hands are visible to the Leap Motion Controller where each has at least two fingers visible. We again use the movement of the hands to be interpreted as scaling the user's view.

For rotation, we changed how it is done since we don't want the slicing and rotating function to be confused with one another. We therefore activate rotation when two hands are present but one has no fingers visible and the other has at least two visible. We then define the rotation as before on the non-closed hand, essentially ignoring the closed hand. This hands configuration is solely to make clear what function is being used. It was inspired by American Sign Language where there might be one hand present solely as a reference while the other hand is moving.

We also define the "empty" function again where two hands are present with neither having any fingers present in the view. This allows the user to stop updating the view at any given time. It also makes switch functions simpler by taking a small break between functions.

With these three functions and the "empty" function defined, a user can now perform slices, scaling and rotations on a given data set without worrying about leaving the touchless interface. This can be seen as how the mouse can perform different functions depending if its clicking, scrolling or click-and-dragging.

4 Analysis of Use

Although we have taken care to define the touchless controls to be intuitive and natural, there areas where we can improve the interface. We could change what gestures are needed to activate actions. We can also combine functions together with perhaps a more intuitive action. We hope that by incorporating more intuitive controls, a user will feel comfortable.

One such example is the the rotation function. The smoothness of the rotation tends to be lacking more so than the rest of the functions. We believe this could be from the calculation of the palm's normal vector between each frame. It perhaps might be better to use some other aspect of the hand to better control the rotation.

It also might also be beneficial if the gestures between zooming and rotation were combined. Zooming is defined by the relative movement between two hands, so perhaps we can combine this into rotation where rotation is defined as the rotation of both hands. One can imagine this by forming a plane between two separated hands and then rotating this imagined plane to translate as a rotation of the view. To zoom in and out, the user simply moves his or her hands away or towards one another, restricting the movement on the imagined plane.

Overall the motions seem to work relatively well, especially in the combined interaction. In the combined interaction, however, there was sometimes a misinterpretation of the action when switching from one function to another. This might be able to be fixed by looking for a threshold of how often to update the screen. But this could effect the flow of the controls, so there would have to be some sort of balance between these two needs.

5 Future Work

We would like to continue on building on this project and expand the functionality of the touchless interface. First, we would like to expand on the slicing feature where we can define not only the direction of the slicing plane as we have been, but also the position of the plane. As of now, the plane is defined to only go through the origin of the data. But it is clearly beneficial to be able to move the plane's position and then change the direction at that point. How to do both of these things seamless has yet to be seen.

The movement function can also be improved on. It seems to be intuitive but was not combined with the rest of the functions because of its lack of distinction of using one hand. A grabbing motion might do well here to separate it from the rest of the functions. We also would have to define where we are moving it in accordance to the view, that is if it was zoomed in or out or rotated we would want the touchless control to correspond to the user's view.

Lastly, we strive towards a goal of incorporating this touchless control into the the Paraview application directly. This might be achievable by via a creation of a plugin for Paraview. The ultimate goal would be to completely replace the mouse interface and solely use the touchless control. Since their are many controls and tools available to Paraview, this might mean incorporating some form of clicking equivalent for the touchless control as one does for a mouse interface to switch from different modes.

References

[1] Leap Motion, Inc. <www.leapmotion.com>

[2] Kitware. Paraview. <www.paraview.org>

[3] Ghandi, Joue, Mittelberg. “Understanding naturalness and intuitiveness in gesture production: insights for touchless gestural interfaces”. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems pp821-824.

[4] Ghandi, Wacharamanotham, Joue, Borchers, Mittelberg. "How we gesture towards machines: an exploratory study of user perceptions of gestural interaction". Extended Abstracts on Human Factors in Computing Systems pp1209-1204.