FinePoint: A Novel Technique for Object Manipulation that Bridges the Gap Between Virtual Reality and the Desktop

Sam Torzewski
torze004@umn.edu

ABSTRACT

We present FinePoint, a technique for positioning and orienting objects in three-dimensional space using a six-degree-of-freedom (6DOF) tracking device. In conventional desktop-based techniques, 2D interfaces such as the mouse and keyboard are used, but with these tools rotation tasks are difficult because the rotation has to be broken down into a few 1- or 2D rotations, and translation tasks are difficult due to lack of depth cues. This puts a greater burden on the user to orient the object correctly. Standard VR interfaces focus on free one-to-one movement with 6-DOF input devices, but their accuracy is limited by the un-steadiness of the user's hands. FinePoint bridges the gap between the full one-to-one manipulation of a 6-DOF tracker, often described by users as the most natural user interface, and the precision of a conventional mouse and keyboard by giving the user many ways of manipulating an object that vary in speed and accuracy. These methods can be switched between on the fly, and new methods can be added on a per-application basis. We use a hierarchical control scheme to select these methods. We report on a pilot test comparing a partial implementation of FinePoint to a standard one-to-one freehand technique. Counter to our initial hypothesis, we found that FinePoint was slower than the one-to-one approach when performing 3D docking tasks at a similar level of accuracy. This pilot has helped us to refine our hypotheses about which aspects of FinePoint will be most useful in practice and in which tasks FinePoint will be most useful. Specifically, we find that separating movement methods into logical groupings is useful, but that further work is needed to make it practical. These methods would be most useful for translation tasks rather than rotation. Ultimately, we believe a FinePoint technique revised and refined according to the lessons described in this thesis will allow more free exploration of data in scientific and medical research, and allow for faster iteration on design tasks in engineering fields.

1. INTRODUCTION

Positioning and orienting objects in a virtual three-dimensional space is challenging. These placement tasks are usually approached in one of two ways: Desktop-based 2D interfaces, or VR-based interfaces that use six-degree-of-freedom (6DOF) input devices. Desktop-based interfaces employ 2D input devices, usually the mouse and keyboard. These interfaces typically use 3D widgets for manipulating the object [6], decomposing each task into a series of 1- or 2D tasks. The VR approach uses a 6DOF 3D tracking device as input to manipulate their 3D objects. The tracker can both translate and rotate in three dimensions, and their motions are mapped one-to-one with the tracker so that changes to the tracker's position and orientation are duplicated on the object. Past research by Bérard et al. [12] and Hinckley et al. [13] suggest that while translation tasks are faster and more precise with a mouse over a 3D tracker, the reverse is true for rotation tasks.

To overcome the challenges present with each of these methods, we present FinePoint, a novel technique for manipulating an object in 3D space using a pen shaped 6DOF tracker. The FinePoint technique aims to connect the modular, tool-based approach of mouse-and-keyboard manipulation techniques with the flexibility afforded by a tracker, in an attempt to leverage the strengths of each. The basis of this technique is the idea that different methods of moving an object are constrained in different ways and they have different levels of precision and speed. The constraints allow the user to only have to think about a few degrees of freedom at a time, without affecting the others. By using several movement methods in succession, the user makes increasingly granular changes to the object's position and orientation. The FinePoint technique provides a way for all of these movement methods to coexist through a state-based system that separates the methods by type, and then allows method selection using gestures.

With the introduction of FinePoint, the 3D tracker could prove to be an important tool for applications that require movement these positioning and rotation tasks. These tasks are often required in design, and the increase in speed could dramatically increase the rate of turnaround for rapid prototyping of new devices and tools. This would not only benefit design tasks, but also aid in the analysis of 3D data. This breadth of possible applications also suggests that there may be useful application-specific movement methods, and thus FinePoint is designed to allow these manipulations to coexist in one technique. The modular nature of the FinePoint

technique means that further research into new manipulation methods can be integrated directly into this technique, and each application can have FinePoint tailored to its specific needs.

2. RELATED WORK

2.1 Comparing Mouse and Tracker

We chose the tracker for its increased freedom of motion, but we cannot ignore the advantages of using the mouse for this placement task. This approach is faster and more accurate than 6DOF trackers for translation tasks [12]. A major problem with this method, though, is that it also increases the cognitive load on the user. When rotating an object, this method requires the user to convert their desired rotation into a series of component rotations, often about specific axes. These limitations are difficult to overcome because the device itself can only move in two dimensions.

Using a 6DOF tracker avoids this limitation, as the tracker can rotate in three dimensions and the entire rotation can be done freely in one step. Rotation tasks are performed more efficiently using 3D trackers than when using the common mouse-based Virtual Sphere and Arcball rotation methods [13]. In fact, when the rotation of the tracker is mapped one-to-one with the rotation of the object, tasks involving rotation can approach the speed of rotating a real-world object with one's hands [3]. The tasks performed faster using 3D trackers, and the error rates were similar across the rotation methods, roughly 6 degrees offset from the intended position. This is due to the problems that real-world manipulation has: the accuracy of the movement is limited by the range of motions of the input device and of the hand—shaky hands make placement incredibly difficult.

2.2 Design Guidelines from 3D User Interface Research

In Shumin Zhai's seminal work The Influence of Muscle Groups on Performance of Multiple Degree-of-Freedom Input [1], it is shown that when using a 6 DOF input device, the use of the user's fingers adds both speed and precision to docking tasks. We therefore choose to use a pen-shaped input device. This gives the user a range of manipulations, largely involving the rotation of the pen, that primarily focus on the motion of the fingers. In designing the FinePoint technique we take into account the affordances of the pen and the fine motor control it provides. Specifically, we use the pen's rotation to affect the object, as the pen can be rotated in multiple directions using finger motions.

When using the tracker pen as an input device, it is easy to implement the tracking in an absolute mode: Each possible position of the pen directly corresponds to some point in virtual space, and to move the object to that location the pen must be moved there. Similarly, to rotate an object to a specific orientation the pen must end up in a similar specific orientation. This absolute mode of tracking can be difficult to operate, as certain locations are outside of the user's natural range of motions. For example, Bi et al. [9] found that the usable rolling angle of a pen device is 90 degrees in each direction when used on a surface. The pen tracker here is being held in the air, so the limitations are different, but the wrist and arm together still have locations and orientations that are difficult to reach, especially if both need to be done at once.

2.3 Constrained 3D Motion

The increased efficiency of rotation with a 3D tracker is also offset by the fact that for translation-tasks, standard one-to-one mapped 3D tracking is much slower and less accurate than a mouse [12]. Teather and Stuerlitzer [10] have found that constraining 3D translation to a 2D plane produces faster and more accurate results in repositioning tasks when compared to fully free one-to-one 3D translations. Similarly, Bérard et al. [12] find that if the object's change in position is based on the initial position of the cursor and how the pen changes relative to when the object was grasped, speed and accuracy are higher than when the position of the pen always maps absolutely to some position in space. That is to say, efficiency is greater when using the pen in a relative rather than an absolute mode. In fact, the speed of these actions approaches that of mouse-based positioning, though the error rate is still higher. Together, these findings show that trackers perform better when the mapping between the pen's location and the object is not necessarily one-to-one. It's important to note that both of these constraints are standard in the use of the mouse: By restricting the tracker to only 2 dimensions when translating, they mimic the mouse's inherent limitations. Furthermore, moving the mouse a large distance in one direction requires re-grasping the mouse, choosing a new starting position each time without moving the object—the relative movement mode is built into the mouse. We adopt both the relative positioning mode and the use of constraints for FinePoint.

As a guideline for which constraints to use, we turn to more recent, touch-screen based object manipulation techniques. In one such paper [11], there are two main primitives for constraining the motion of the object. These are the pivot, around which the object is rotated, and a second primitive that can switch between an axis, along which the object can be moved forward and back, and a plane orthogonal to that axis, in which the object can move anywhere in that plane. In both cases the widget allows for rotation about the original axis. We use a similar structure for our constraints.

2.4 Existing Methods for 3D Motion

The error rate for 6-DOF translation is still higher than mouse-based methods, and the error for rotation is large enough that there is a need for enhanced precision for both of these manipulation types. Constraints reduce the number of things the user must control at once, but even with these in place the motions that must be made for accurate placement are difficult. Furthermore, the restricted range of motion of the user's hand and arm makes large changes in rotation and translation difficult.

To provide the user with the efficiency we desire, there must be additional ways to move the object beyond the constrained, relative one-to-one mapping. One way to provide this is with the addition of non-isomorphic movement: Making large changes to the object's position or orientation with smaller changes to the pen, or making small adjustments using large changes to the tracker. This was shown to provide a 13% speed increase in large rotation tasks [5]. Another way is to allow for abstracted movement methods. When rotat-

ing objects with a mouse, the mouse is translated rather than rotated, and on-screen widgets are often used to describe how the motion of the mouse maps to the rotation of the object [6]. A similar approach can be used with the 6-DOF tracker, allowing for both continuous and discrete movements to be made.

2.5 Design Guidelines from Desktop Graphics Research

To handle many different manipulation modes we need some way to select between them. We base our approach on similar research using 2D input devices. Our main inspiration was from Zeleznik and Forsberg's UniCam technique [4] in which camera controls were manipulated using only a mouse and a single button. Eight different camera controls could be used to manipulate the camera, and were differentiated by the position of the cursor when clicking, whether the button was tapped or held down, and the initial direction of motion after holding down the button. We expand this approach to a full 6DOF tracking device, taking advantage of its ability to both translate and rotate to specify many different ways to manipulate the object.

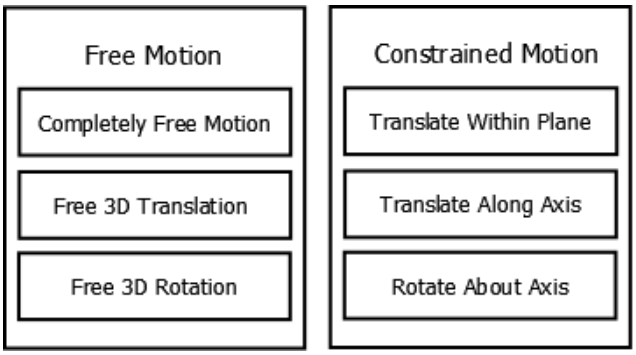


Figure 1: The FinePoint technique defines six movement types in two groups. Three types, those in the Constrained Motion group, are used in relation to a Constraint Widget that defines an axis and plane. The other three do not require this widget.

3. FINEPOINT

The goal of the FinePoint technique is to integrate existing methods for precise placement into one technique so that the user can switch between these methods on the fly. We divide these methods into two groups: Movement freely in space, and movement about some axis or within some plane. To correctly constrain the object's motion about that axis or plane, we use a Constraint Widget that defines a point in space, an axis out of that point, and a plane orthogonal to that axis. These two groups of manipulation methods are further broken up into 6 main types of motion, as seen in Figure 1. Each type of motion has multiple specific movement methods because different methods will be useful in different situations. Some are good for the initial rough placement of the object, while others are made for fine-tuning, giving the user a range of options that can be used in sequence. FinePoint allows the user to change both the movement type used, and the way the object's motion is being constrained, at any time. This lets the user switch rapidly between any desired set of manipulation methods. To interact with the world, we use a pen-shaped 3D tracker with a button on it. Through simple gesture recognition, the user chooses the specific method of interaction and can move the object.

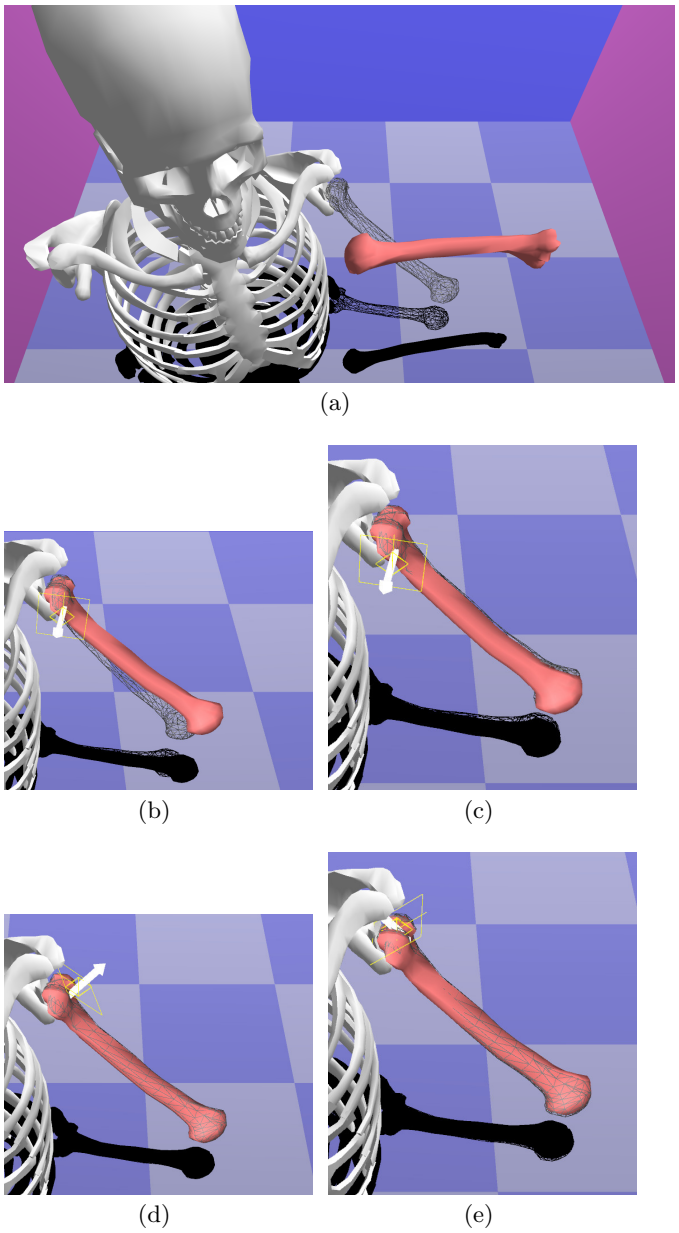


Figure 2: In this example use case, the bone must be re-oriented and placed into position. The start state is shown in Figure 2(a). The bone is roughly placed using the freehand one-to-one movement method, as depicted in 2(b). The bone is rotated around this axis in 2(c). The bone is then rotated away from the user around the axis shown in 2(d). The end result is shown in 2(e).

3.1 User-Experience Example

We now describe a typical user experience for the task of orienting and positioning one object relative to another. In this example, the user wishes to align a humerus with a

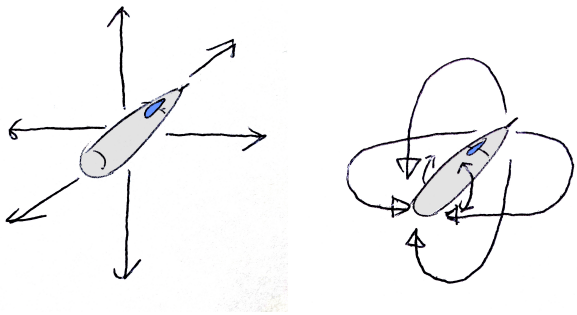
skeleton torso. The two objects are shown in their starting positions in Figure 2(a). The gray wireframe bone indicates the desired position and orientation of the humerus, and thus the task is to align the actual bone as closely as possible to this wireframe.

We start with the humerus already selected. The user then uses the tracker-pen to move the object. The user holds down the pen button, and then moves the pen horizontally. This makes the system enter the one-to-one movement mode. In this mode, the humerus will mimic the movements of the pen. When the pen rotates, so will the bone. The same goes for translation. The user then roughly positions the object such that one end of the bone is aligned roughly with the wireframe, and releases the pen button. Once the button is released, the object will no longer move with the pen. The new position of the bone is depicted in Figure 2(b). Note that the bone is aligned using the shadows as a guide.

Next, the user wants to rotate the bone down to the mesh. The user moves the cursor to the aligned end of the bone and clicks once, creating a constraint point widget, then drags a line out from it, towards the user. This puts the constraint widget in Axis/Plane mode. The widget is shown in Figure 2(b) near the arm socket. The object can be rotated about that axis by holding down the 'R' key to enter the 'Axis Rotate' mode. Now, the user presses down the pen button, and moves the pen horizontally again. Since the user is in this Axis Rotate mode, this puts the system in the Winding state. By moving the pen cursor up and down, the object is spun as if attached to the cursor by a stick. The user rotates it such that the bone is as close as possible to the wireframe, and releases the pen button. This gets the bone near its desired orientation, as seen in Figure 2(c), but not near enough.

To finish aligning the bone, the user must rotate it about other axes than the constraint axis. Specifically, the elbow-end of the bone is too far forward, even though the other end is aligned properly. The user deletes the constraint widget by clicking, and recreates it with a new orientation. The user then presses down the pen button and makes a yawing motion to the left. This puts the system in the 'Flick' state. Each time the user performs a flick to the left, the object rotates one degree counterclockwise about the axis, aligning it more closely with the wireframe mesh using the shadow as a guide. The user does this once more before releasing the button, and this result is shown in Figure 2(d). At this point, the end of the object needs to be rotated away from the user. At this point the object is only misaligned by 3 degrees total. This primarily comes from misalignment around the major axis. Once again, the user recreates the Axis/Plane constraint widget and uses the flicking gesture to align the object. The end result, as shown in Figure 2(e), is aligned with the wireframe with a translational error of one millimeter and a rotational error of half of a degree.

It's important to note that the bone in Figure 2(d) might look closer to the wireframe, as can be observed by the fact that less of the wireframe is visible in the shadow, and the object seems to encompass the wireframe itself. This is for two reasons: first, the object as it is aligned in Figure 2(d) is wider than the actual object—has a larger silhouette from the user's perspective—and thus the errors are covered up. Second, the translational offset is masking the deficiencies in the rotational offset. Note the disparities at the shoulder-end of the humerus in Figure 2(d), and that those disparities are much less apparent in Figure 2(e).



(a) (b)

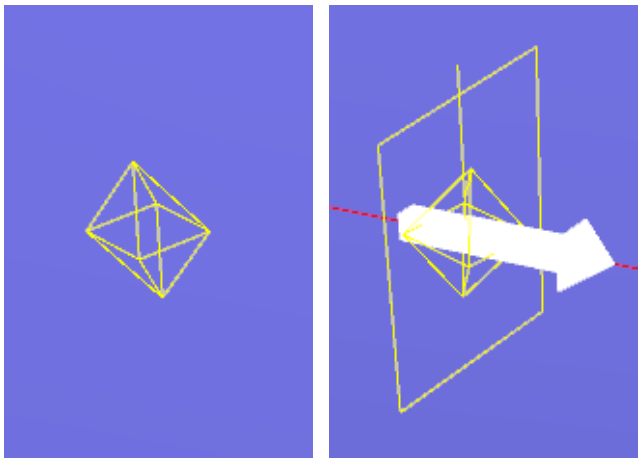
Figure 3: The gestures allowed by the pen all involve either translating along an axis of the pen or rotating about an axis of the pen, but not both. These axes are defined as the pen's major axis, the up direction (where the button is) and the axis orthogonal to the previous two. Each gesture involves rotating or translating in one dimension. This gives 12 possible gestures.

3.2 The Pen

In order to interact with the object, we use a 3D tracker pen. The specific technology used is not of utmost importance, it just needs to have a definite position in 3D space and a button on the pen. The pen is chosen as a tool that can be adeptly manipulated by the user's wrist and fingers, rather than just the arm. Furthermore, it also has an easily-identifiable major axis and a forward direction. A major feature of the FinePoint technique is the ability to constrain motion about some axis, so we can use the pen's shape to correspond with this axis. This is especially important for the gesture recognition mentioned above.

We define all of the gestures in terms of the pen's initial position and orientation at the start of the gesture. These gestures are shown in Figure 3. The basic guidelines for these gestures are that the pen should only either rotate or translate, and that they should only have to move along/around one axis. For example, the user may roll the pen about its main axis, or pitch the pen upwards, but not do both in one gesture. This reduces the complexity of the gestures. These are all in relation to the pen's direction: if the pen is being held sideways, the "forward motion" gesture is chosen by moving the pen along its major axis, rather than moving it forwards towards the screen. This stipulation prevents the pen from having to be oriented in the correct way for a given gesture to make sense. For example, the roll gesture will always require rolling the pen, not just rotating it around some arbitrary axis in space.

3.3 The Constraint Widgets



(a) (b)

Figure 4: This depicts the Constraint Widget in its two forms. On the left, in Figure 4(a), is the Constraint Point. The object rotates about this point. On the right, in Figure 4(b), the Axis/Plane constraint form is shown. In this form, the axis is drawn as white arrow, and the plane orthogonal to it is drawn as a square around it. The constraint point is still drawn here. The vertical line orthogonal to the arrow denotes the "up" direction of the plane.

One major aspect of the FinePoint technique is the ability to constrain the object to a smaller number of degrees of freedom. This allows the user to make adjustments to some components of the object's position or orientation, without affecting the others. To control what constraints are being placed on the object, we use the Constraint Widget, shown in Figure 4. As we see, this widget takes on two forms: a single point in Figure 4(a), and a combined axis/plane constraint in Figure 4(b). The user first creates the constraint-point by clicking the button on the pen once. This will place the point at the cursor's position in space. Normally, an object will rotate about its centroid. Once the constraint-point has been placed, however, the object will rotate about that point instead.

By placing the cursor on the constraint point, then holding down the button and dragging away from the point, the constraint widget will go into Axis/Plane mode. This mode has an axis coming out of the constraint point, and a plane orthogonal to that axis. When the user is dragging, the arrow will point towards the position of the cursor, and will remain oriented in that direction after letting go of the button. With this axis defined, all rotations will happen about that axis. When the user clicks on empty space when the widget is in Axis/Plane mode, it will be deleted. This allows the user to quickly discard the constraint without having to maneuver the cursor to a potentially awkward position.

3.4 Movement Types

The FinePoint technique is designed to combine many different methods of manipulating objects into one technique. These methods are divided into two main groups: Methods that translate or rotate the object freely, and methods that move the object in relation to some axis or plane. We call these two groups "Free Motion" and "Constrained Motion". We then divide each group into three types of motion, as seen in Figure 1. To determine which set of methods are available, we use a state machine based on these six types: When the user is in the correct state, they can use all manipulation methods associated with the corresponding type.

In the Free Motion group the three types of motion are "Free Translation", in which the object can be translated in any direction in 3D space but will not rotate, "Free Rotate", where the object can be rotated in all three directions freely, but has its position locked in place, and a "Completely Free" type in which the user can both translate and rotate the object, for a full six degrees of freedom. In the Constrained Motion group, we require that an Axis/Plane constraint is defined with the constraint widget. The three types of motion here are "Axis Rotation", in which all rotation is about the axis, "Axis Translation", in which the object can only move back and forth along the axis, and "Planar Translation", in which the object's movements are locked to a plane orthogonal to that axis. While Free Rotate and Free Translate are constrained to only three degrees of freedom apiece, only rotation or only translation, they are grouped in the Free Motion category because they still have more freedom of movement than the Constrained Motion states, which only have one or two degrees available to them.

3.5 The State Machine

The six states are organized as shown in Figure 5. We first start by looking at the Free Motion states. When the user begins the overall movement task, they will start with nothing constraining the object's movement, and can choose more specific constraints as necessary. Thus, the user starts in the Completely Free state. From there, the user can switch to the Free Rotate and Free Translate by holding down the 'R' and 'T' keys respectively. They are only in these states as long as the corresponding keys are held, and releasing the keys returns the user to the Completely Free state. This allows the user to rapidly toggle between these three states.

Now we look at the Constrained Motion group. We can think of the two groups as being parallel to one another, with one state as the baseline and the other two states entered by holding down a keyboard key. We place the Axis Rotate and Axis Translate states in the same positions as the Free Rotate and Free Translate states, and correlate the Plane Translate state with the Completely Free state, at the top. Thus, the user starts in the Plane Translate state and enters the Axis Rotate and Axis Translate states by holding down 'R' and 'T'. This is chosen because planar translation allows for two degrees of freedom, rather than just one. By pressing the 'R' or 'T' keys we further restrain its motion.

To switch from the Free Motion group to the Constrained Motion group, the user must define an axis/plane constraint. Later, the user can delete that constraint to switch back to the Free Motion group. Since the constraint widget can be created or deleted at any time, the user does not need to release the state-changing keys to do so. Thus, the user can switch from the Axis Rotate state to the Free Rotate state

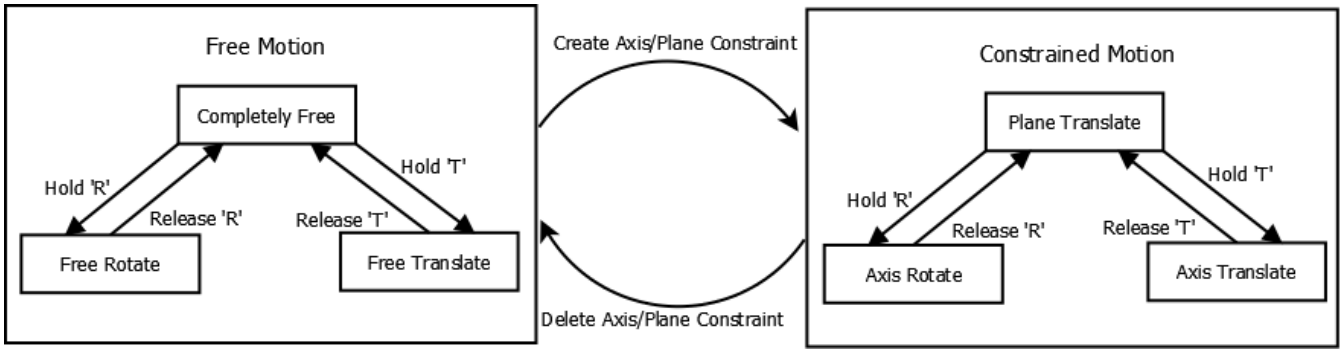


Figure 5: The state machine. Within each of the two groups, Free Motion and Constrained Motion, the state is based on whether the R or T keys are held. When neither are held, FinePoint is in the top-level state, either Completely Free or Plane Translate. To switch between groups, the user creates or deletes an Axis/Plane Constraint Widget.

by deleting the constraint, without having to return to the Plane Translate state first. When only the point-constraint exists, the object will be in the Free Motion states.

3.6 Object Manipulation

Each state corresponds to one of the six movement types. In a given state, the user can choose one of several movement methods. Each of these methods is of the same movement type, but they vary in terms of speed and precision. This allows the user to first roughly align an object with a fast method, and then switch over to a more deliberate method for fine-tuning. For example, a user wants to move an object along an axis. The user may first use a continuous movement method, where the object slides along the axis until it reaches the correct place. Then, the user switches to a discrete "nudging" method to put the object exactly in place. We differentiate between these actions using simple gesture recognition, as mentioned above in Section 3.2. To begin, the user holds down the button on the pen. Then, they make an initial translation or rotation. Once the gesture is recognized, the object will move. In the example, the continuous sliding method is selected by making a poking gesture with the pen, causing the object to move one-to-one with the pen. The user is locked into that movement method until the button is released. This allows a movement method to interpret several motions in different ways, without worrying about accidentally switching to a different method by making a wrong gesture.

This gesture-based selection allows the user to pick from the different methods on a moment-to-moment basis. This gives the user to rapidly switch between these methods in orders of increasing granularity for a result that is precise, without wasting time making the large, rough movements with a more precise tool. The specific methods implemented will vary from application to application, however. For example, in some applications the difference between very large movements and very small ones is just a few orders of magnitude: when the large movements are still relatively small to begin with, as with the bone placement example above. In this case the user may only need two different movement methods to achieve the desired level of precision. In other applications, the user may be tasked with re-positioning small objects over very large distances, and thus several tools may be necessary to make increasingly fine-grained adjustments.

4. IMPLEMENTATION AND RESULTS

To demonstrate the FinePoint technique, we developed a sample implementation and pilot tested it with a docking task inspired by 3D medical imaging applications. For our pilot test, the user's task is to reposition a humerus relative to a skeleton torso, as in the example user experience section above. The goal of this task is to align the humerus so that it matches with the gray wireframe humerus in the shoulder-blade socket, as seen in Figure 2(e). This task allows us to measure the accuracy of different placement methods. The bone starts in the pictured default position and must be moved into position. When the desired level of accuracy is met, the console indicates that the task has been completed. The accuracy constraints used in this example are a maximum of one millimeter error in translation and one degree of error in rotation.

Our discussion begins with a description of our implementation. We did not implement a full set of movement methods for each state, instead choosing to focus on creating a robust set of manipulations for just one state. For this we chose the Axis-Rotate state. We also implemented a basic one-to-one movement method for each of the other states. Following this, we describe informal user feedback and results from our pilot testing, which has so far included just ourselves and our lab members as users.

4.1 Axis-Rotate State Implementation

In the Axis-Rotate state, we implement three different rotation methods: a twisting method, a flicking method, and a winding method. In all cases, when the pen button is released the object jumps back to the Axis-Rotate state, and any of these three gestures can be performed again to select that movement method.

The first method, twisting, is shown in Figure 6. To enter the mode, the pen is rolled in either direction. Once it has rolled far enough, FinePoint will recognize this rolling as a gesture. Once this happens, the object will start twisting along with the pen. The pen will twist in either direction, so in order to lock it in place the user must let go of the

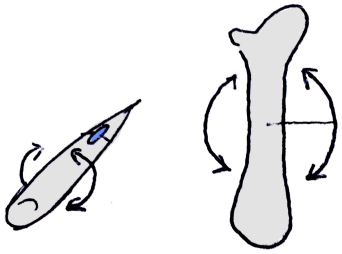
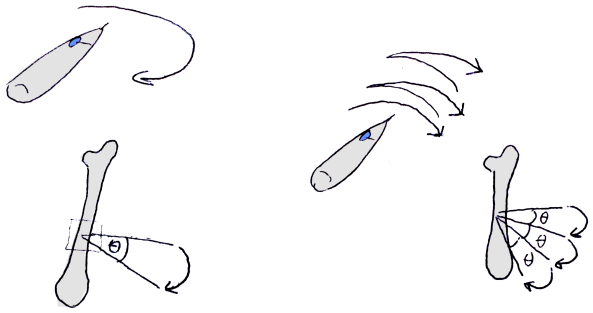


Figure 6: The twisting method of rotation. The pen is twisted, rolled about its main axis, and the object also twists. We assume the constraint axis is placed at the centroid for this example, but it need not be—the object will rotate about the constraint axis, wherever it is placed.

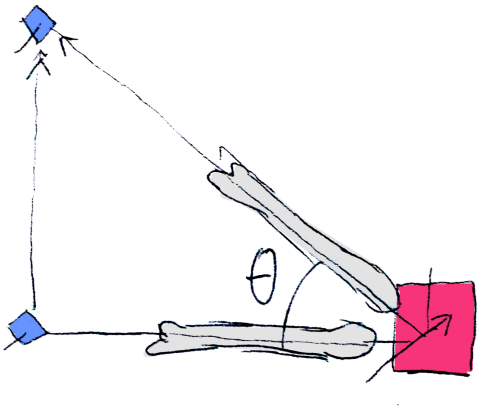
button. From there, the user can reorient their hand and start twisting again, as if turning a screwdriver. This is an example of intuitive, one-to-one motion. The user enters into this method by making a rolling action about the pen axis, and the object then rolls around the constraint axis.



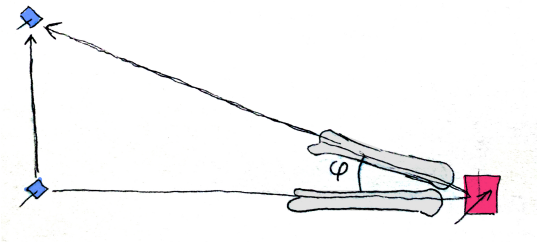
(a) (b)

Figure 7: The flicking method of rotation. On the left, we see the result of a single flick: The object is rotated by a set amount, theta. On the right, multiple flicks are depicted: by making multiple gestures in a row, the object continues to rotate in discrete increments.

The second method, flicking, is shown in Figure 7. The user makes a flicking gesture in one of four directions: up, down, left, or right. More specifically, the user must either pitch or yaw the pen while keeping it generally in the same position. This puts the object in the Flicking state. From there, as long as the pen is held down, the user can keep making flicks in the same direction to rotate the object. Each flick rotates the object by a discrete amount. In our implementation, this is 1 degree. This allows the user to make very fine adjustments to the object's orientation once it is mostly oriented correctly. This, and other discrete-increment methods, allow for absolute precision numerically, rather than looking for precise placement visually.



(a)



(b)

Figure 8: The winding method of rotation. We consider the Figure 8(a). The cursor is depicted in blue, while the Axis/Plane constraint is colored red. The user first pulls out an arm from the axis by dragging horizontally. Then, by repositioning the cursor, the object will rotate as if attached to the arm. The arm can be dragged in a complete circle around the Axis/Plane constraint. Comparing both images, each of these examples depict the arm being moved upwards by the same amount. In the second image, the cursor is farther from its start point, so the angle of rotation is smaller.

The third method, winding, is demonstrated in Figure 8(a). The user pulls out an arm horizontally from the object. Once the translation is large enough, the winding gesture will be recognized. After that point, moving the pen up and down will cause the object to rotate. More specifically, once the winding state is entered the arm is locked to a specific orientation of the object. By moving the arm to a different point around the object, the object will turn to face it. Thus, the object can be fully rotated by dragging the pen around in a circle from where the gesture first started. The other important thing about the winding gesture is the arm's length. This is demonstrated by comparing Figure 8(a) and Figure 8(b). When the pen is close to where the gesture began, the arm is short. Thus, moving the pen a small amount will change the angle of difference a lot. When the arm is long, a similar vertical movement of the pen will only change the angle by a small amount. This is an example of a more

abstracted movement method, since the pen does not need to be rotated in order to rotate the object. These sorts of abstracted methods can be applied to other situations: For example, the winding method can be used for axis translation as well: By winding around the object, it can be moved very slowly but continuously.

To align the humerus with the wireframe, the user can first use the twisting or winding methods to achieve a decent level of accuracy, then use the flicking method to narrow in on the most accurate possible orientation. Multiple levels are provided because, even though the flicking method is the most accurate and discrete, making big changes to the orientation requires a very large number of flicking motions. For changes larger than a few degrees, it is much faster to use one of the other two first, then switch to flicking. This can be used in conjunction with the one-to-one methods for translating within the plane and along the axis to move the bone from its starting position into place.

4.2 Object Movement and Pen Orientation

The most important observation made is that object motion should not always be relative to the pen's motion. For example, consider the scenario where the object is in Free Translate mode using a one-to-one movement method. If the user moves the pen toward the screen, the object should follow suit, regardless of object orientation. The reasoning is this: Consider the case where the opposite is true. The object will move straight backwards only if the pen is moved in a straight line along its axis, rather than straight back in world space. If the user is unable to perfectly position the pen to face towards the screen, then the object will move at a slightly wrong angle when the user moves the pen towards the screen. Thus, it is often better to make translation actions correct from the user's perspective, rather than from the perspective of the pen's orientation.

The above advice does not necessarily always apply, especially in the case of rotation. We next return to the axis rotate mode: Trying to rotate the object around an axis should only require twisting, or else the axis of rotation no longer lines up with the axis of the pen, and understanding how to rotate the pen becomes more difficult. One caveat to this is that if the axis of rotation is facing towards the user, and the pen is facing away, the object will appear to rotate in the opposite direction of the pen. For example, if both the pen and object are rotated clockwise, the object rotates counterclockwise from the direction of the user. This can be fixed by reversing rotation direction when the axis is opposite to the user. This also applies to translations in the plane orthogonal to the axis. The takeaway from this is that, while the developer can implement many methods of interacting with the object, not all of them will make sense even when following the same set of rules.

4.3 Graphical Elements

The implementation as it stands is lacking in graphical cues as to which state the system is in. We consider a few possibilities here, and discuss the trade-offs between them. It is easy to add cueing information to the axis/plane form of the constraint widget. For example, the widget might have the arrow highlighted in the axis-translate state, and the plane highlighted in the plane-rotate state. A ring signifying rotation could be added for the axis-rotate state. The difficulty here is that there is not always a constraint widget. The user could instead have a widget around the object itself, but the user loses the connection between the widget and the object's motion. One possible solution would be to draw the state information around the constraint widget when it exists, and around the object when it does not. This is internally consistent, since the object rotates about its centroid when no constraint point exists so the constraint point is considered to be at the center of the object.

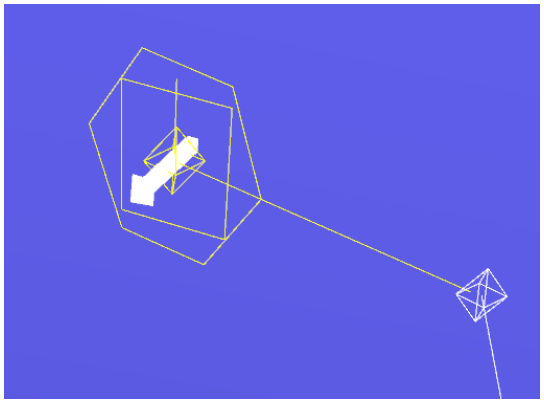


Figure 9: The winding arm is depicted her, extending from the constraint widget within the plane of the widget. A hexagon also appears around the constraint widget that will rotate with the object, and the arm will change in length based on the displacement of the cursor from its initial location.

Another factor to consider is graphical elements that differentiate manipulation methods within the same state. These will help the user identify when they have accidentally selected the wrong method through a misinterpreted gesture. Gesture recognition is not a perfect science, so we can expect this to happen with some frequency, though we do try to minimize it with our stipulations for simple gestures. By developing a different graphical element for each method, we can let the user identify the state they are in even if they do not know what sequence of actions led to this state. Our example here is the winding method from our implementation, as pictured in Figure 9. We draw an arm that depicts the distance the pen is from the initial grabbing point, which helps the user identify where they can move the pen to get which results. We also draw a hexagon in the plane orthogonal to the axis, to give a better understanding how a flat object rotates in the plane.

4.4 Results from Pilot Testing

Results from Pilot Testing To test the validity of the FinePoint technique, we tested the docking task described in the User Experience Example. This is not a formal user study, simply an experiment to see if the technique was worth pursuing further. The bone started in a set position, and had to be re-oriented and re-positioned to match the mesh. This was run with only one user under three sets of parameters: First we ran the docking task with FinePoint. While all methods were available to the user, freehand movement was only used for an initial rough alignment. After that, all

translation and rotation was done using the Axis/Plane constraint widget. Next, the task ran using a Freehand technique, using only the one-to-one free-movement method with the constraint widget disabled. Finally, a middle-of-the-road approach was tested, where the Completely Free, Free Rotate, and Free Translate movement types were available, but the constraint widget was disabled. Each task was run 5 times, and the number of seconds required was recorded. After 5 minutes, the trial was aborted if not yet complete. In order to count as 'aligned', the user had to release the button so that the object was at rest in the correct position. The results of this test are shown in Table 1.

Technique	FinePoint	Freehand	Middle-of-the-road
Trial 1	122.81	46.368	20.975
Trial 2	177.163	106.777	ABORTED
Trial 3	173.233	90.228	37.532
Trial 4	87.136	28.076	235.828
Trial 5	166.511	19.95	181.575
Average	11.268	58.280	155.182

Table 1: A comparison between the time spent placing the humerus in the wireframe guide using FinePoint, a freehand placement method, and a middle-of-the-road placement method.

Thus, the fastest technique for alignment was the Freehand technique. Furthermore, this method generally saw increasing returns from repeated tests, while the others did not. The Middle-of-the-Road technique was the slowest, but also had the largest variance. In some instances, alignment went very fast, but in one case the user was unable to align the object and ran out of time. FinePoint was much slower than the freehand technique, but did not ever take so long that the trial had to be aborted. We discuss the ramifications of these results, and what this means for the development of the FinePoint technique in the Results Discussion section below.

5. DISCUSSION

The FinePoint technique has been implemented in this paper—or rather, a subset of it has been. This technique is meant to be one tool in a toolbox for whatever task the user is performing, rather than a standalone application. Here we discuss the design decisions made and possible directions for expanding on the FinePoint technique in the future. Any future iterations on this research would also include a formal user study that compares the FinePoint's effectiveness with mouse-based techniques in terms of both rotating and translating.

5.1 Initial Design Goals and Unsuccessful Implementations

The driving force behind FinePoint is the ability to manipulate an object with speed and precision, and as we have seen in the Related Works section, the methods for improving efficiency differ between translation and rotation. Thus, instead of trying to find a movement paradigm that is superior in all regards to both desktop manipulation methods and VR trackers, we simply allow for both. By combining this with the axis-and-plane widget idea from Schmidt et al. [11], we can have a system where constraints can be placed and then the user can either rotate or translate. Where we depart from that initial widget is in deciding whether to translate or rotate. The Schmidt et al. paper uses a 2D cursor-based interface, which requires the user to click on a part of the widget in order to either rotate or translate. This is more difficult in 3D, and also lacks some of the features we'd like to allow for, such as non-isomorphic, scaled movement and discrete, incremental motions.

To work with the 3D tracker effectively, we need a few changes from the system by Schmidt et al. First, we want a system that where the pen does not need to be in specific position or orientation in order to start moving the object. Second, we need a system that allows for the more abstract input methods we mention above. In order to implement a one-to-one movement method, an incremental method, and a non-isomorphic method for each of rotation and translation, we need to have a way to select each movement method. In an effort to reduce the cognitive load required at any given moment we split these methods up based on how they affect the object. Thus, we want different states for translation methods and rotation methods.

Initially, on the thinking that every action should be performed with a pen where possible, all state-switching was done with gestures. Thus, the user had to perform gestures both to switch between states and to perform object manipulations. The result of this thinking was a 3-level tree, where the user starts at the root level and first makes a gesture to transition to a movement type, and then makes second to choose the specific manipulation method. Once the manipulation method was chosen, the user would use that method until they explicitly cancelled out of it. This was cumbersome and put too much burden on the user to remember which gestures did what in which situations. By collapsing all movement methods of a given type into one state, this reduced the tree to just two levels. From there, FinePoint developed into its current form, which uses the constraint widget and keyboard keys to switch states instead of gestures, separating the "manipulation method" actions from the "state change" actions.

The initial constraint widget was much more complex. It was built as a separate mode altogether that the user used a keyboard toggle to enter, and the user could then place constraint points in space. From there, the user could create an axis or a plane by connecting multiple constraint points, or just select a constraint point to use as a pivot. This made the pivot and the axis/plane two different things instead of two forms of one widget. This implementation allowed for multiple constraint widgets to exist at once, but made creating an axis/plane constraint quite complex. By switching to a constraint widget that could be created and destroyed at any time from any state, the widget is forced to be more simplistic, and this reduction in complexity led to the current system. In Section 5.3, we discuss some additional functionality that can be added back into this current model of the widget.

5.2 Design Decisions for the Current State Machine

The use of the keyboard for state-switching is based on the research of Yang et al [7], which suggests that mode-

switching is most efficient through button presses on the non-dominant hand. We are wary about putting too much focus on the keyboard, though, as it divides the user's attention between the pen and the keyboard. We want the user to be focused on the pen position, as the pen is often going to be at the center of focus between the user and the screen. With that in mind, we use gestures to select movement methods, taking advantage of the free-floating nature of the tracker to allow for many kinds of simple gestures.

Getting into the specifics of the six states chosen, we circle back around to the constraint widget. We recognize that requiring the user to set up a constraint widget with the right axis/plane in order to move the object at all is asking a lot from a first-time user. Thus, we allow for movements that are unconstrained by the constraint widget. Even without the widget, the user may wish to lock the object to only translation or only rotation, so we provide for that with the two keyboard mode-change keys, making the completely free movement mode the default. Once the constraint widget enters the picture again, integrating all of the movement types gets slightly more complex. When considering an axis, an object can either translate along that object or orthogonal to it, giving two different possible translation states. The object can also be rotated about that axis. Having a second rotation-type in relation to the axis makes little sense, as all 3D rotations can be made using a series of 2D rotations, so this can effectively be lumped into the 3D rotation-only state. Still, that leaves us with the six movement types: Completely free movement, free rotation, free translation, axis-only translation, plane-only translation, and axis-only rotation. Having six different keys to switch between these states will put too much focus on remembering which key does what, so we relegate the free-motion states to only be accessible when there is no constraint widget. This reduces it down to only three states accessible with the keyboard. In this model of the FinePoint technique, only the "Completely Free" movement mode allows for both translation and rotation. This is purposeful, to reduce the number of states the user has to keep track of, but it does somewhat limit the possible manipulation methods available. New ways of pairing the translation and rotation, such as "constrained translation but free rotation" could always be added in by using another keyboard modifier, but as it is the system is built around the current six states.

5.3 Expanding the Constraint Widget

The constraint widget as it is described in this paper is very simple. There can only be one of them, and it has to be placed by moving the cursor to the right spot, unable to move. Expanding this widget to include additional functionality will give the system much more precision and flexibility. We outline here the additions we feel are necessary to make the widget complete. The first is to allow the user to have multiple widgets at a time, with one of them active and the others disabled. This gives the system a memory of previous points that lets the user toggle between different constraints as necessary. Another important feature is the ability to move constraint points. A handle can be added to the widget that the user can grab with the cursor to move around. It could be treated like an object, and moved about using the FinePoint technique rather than only moving in a one-to-one fashion. This allows the user to make small adjustments to their constraints, just as the user would make adjustments to the object, ever-increasing the maximal level of precision available. The other important feature is the ability to connect these constraints to objects, such that when the object moves the constraint moves along with it. This allows the user to place a constraint point at a position on the object, move the object to a new location, but still be able to rotate the object about that constraint point. This idea of scoping constraints connects the other expansions on the widget by allowing the user to move anything in relation to anything else.

5.4 Standardized Gestures

The FinePoint technique does not define a fully-realized set of manipulation methods for each movement type, but we recognize that not all implementations need to have their own specialized set of gestures. Despite this, there are a few concepts in the implementation section that can be applied to more than just the Axis-Rotate state. First is the flicking gesture. This gesture can be performed in 4 directions: up, down, left, and right. This allows for positive and negative changes along 2 degrees of freedom, making this a useful gesture for both the plane-translate and axis-translate movement types. Second, the winding action can be re-used for axis translation, where making successive circles slowly inches the object in one direction or another. Third, we look at the twisting gesture. As mentioned above, the rotation of the pen has a one-to-one correspondence with the rotation of the object. A similar thought process can be applied to the other states: moving in a direction that makes sense for that mode. For example, the user can make a poking gesture in the axis-translate state to begin moving along the axis. Further work on FinePoint will involve expanding on these gestures to create a cohesive set of gestures for each movement type that allows for several kinds of motion of varying granularity. The goal of this work will be to create a set of movement metaphors, each of which apply to most of the different movement types. This will reduce the number of different movement methods the user needs to remember, keeping only one set of gestures in mind at all times rather than having a different set of three for each state. Once a set of standard methods has been developed, an application can also implement expert-level methods that provide specific functionality for a user who is already comfortable with the standard methods.

5.5 Insights on the Pilot Test Results

Reflecting on the informal pilot test, we notice a few areas in which FinePoint is currently lacking. First is something we are already aware of: only the axis-rotation methods have so far been implemented, and all other movements are one-to-one. As noted in the introduction, rotation is faster using freehand 6DOF motion than with a mouse, but translation is slower in this freehand 6DOF mode. Since we focused on using the features currently implemented in FinePoint, we used one-to-one translation (though still constrained to a plane or axis) with abstracted-out, gesture-driven rotation. This perhaps gives us the worst of both worlds, but gives a good glimpse at the potential troubles that will exist even with a fully-implemented system. The two big ones we noticed were problems with the constraint widget, and the high rate of user error with the FinePoint technique.

The constraint widget was more problematic to orient than expected. The first issue was that the constraint widget could only be placed at the end of the cursor. While this extended out from the pen a few inches, this meant that constraint points could only be placed a few inches into the scene—about where the bones were located. Thus, axes could only be drawn towards the user, not away. A different method of placing constraint points may be required, or a different way of controlling the cursor's depth in the virtual space. Second, it was difficult to orient the axis in a desired direction. While the user often tried to orient the axis along the object's major axis, or orthogonal to it, perceptual errors meant that the object would actually rotate about some other, less ideal axis. Even when the user knew the steps needed to achieve the right amount of accuracy, following through with this was difficult. This issue also occurred with translation. This suggests that the constraint widget needs to expand further to allow common axes to be easily selected.

During the FinePoint technique trials, a large number of unintentional movements, mis-clicks, and other errors occurred. Some of these were caused by tracker accuracy: if the hardware thinks the tracker is oriented differently than it actually is, gestures are interpreted incorrectly. While not a FinePoint error specifically, it may be necessary to consider safeguards against these accidental errors in later iterations of the FinePoint technique. Other errors were caused by not holding down the 'R' button before starting a gesture. This caused the system to do a planar translate instead of a rotation. This occurred several times, each time requiring the user to reposition the bone within the plane. This may mean that a default 'root' state should exist that does not allow for any rotation or translation, and the plane-translate state can be entered by holding a different button. Alternatively, this might suggest that state-transitions should be done by toggling the button, not holding it down, which would make it harder to accidentally return to the plane-translate mode. We expect that these issues will be lessened by giving the user more visual feedback about which mode the system is in at any given time.

One other concern that arose during the trial was arm and wrist fatigue. In general, FinePoint caused less fatigue than the freehand approach, as the hand did not have to twist in strenuous directions. Despite this, we believe it will be worthwhile to make gestures as easy to perform as possible. This means movement methods that do not require the user to lift their arm off the table, and making sure gestures do not require high precision from the user's hands, or else the user will tense up and strain the wrist for long periods of time. Relatedly, care must be taken in how the gestures are implemented. For the twisting movement method, the threshold for recognizing as a twist gesture was set at ten degrees. Once this was reached, the object would rotate instantly to align itself with the pen's rotation. To rotate the object five degrees, it must first be rotated the ten degrees necessary to recognize the gesture, then rotated back by five degrees. This is inefficient and frustrating to deal with. To combat this, rotations must be based on the amount of rotation after the gesture is recognized.

6. CONCLUSION

The intent of the FinePoint technique is to combine the best traits of both desktop and VR systems for manipulating objects. We believe that there is merit to the idea of switching between different movement methods based on the type of movement and level of granularity, but in pilot testing this is still much slower than regular one-to-one tracking. Despite this, there are a large number of enhancements left to be made before FinePoint is fully implemented, and thus we believe it is still an exciting direction for future research. The addition of multiple constrained translation methods will increase the viability of the FinePoint technique immensely, as translation is the weak point of the 6DOF tracker.

One part of this work that shows promise is the combination of 3D tracker and keyboard. We believe that the hotkey-driven interfaces of many desktop applications can be combined with the freedom of the tracker—especially in its ability to rotate—to allow for novel interface methods that are more robust than is allowed by the mouse. Gestures provide a second set of interface methods that complement the hotkeys, allowing the application to assign different kinds of commands to each. This will also require research into ways to accurately express these controls to the user, as the increased freedom of the pen also increases the cognitive load on the user, in remembering what different motions do.

From this work we can conclude that one major factor in balancing a large number of movement methods is expressing them all differently with feedback to the user, and making sure the user can keep track of them all. Since the movement methods are selected via gesture, and different gestures are used in different modes, there must be a way of representing all of these to the user in a meaningful way. While using a mouse to navigate through a list to select different tools can be slow and cumbersome, it allows the user to see all options available at once. Since gestures are motions, this is not such an easy task. Future work in integrating the 6DOF tracker with desktop interfaces must address this problem in order to utilize the tracker effectively.

7. ACKNOWLEDGEMENTS

This work was funded in part by the NSF (IIS-1054783).

8. REFERENCES

[1] Zhai, Shumin, Paul Milgram, and William Buxton. "The influence of muscle groups on performance of multiple degree-of-freedom input." *Proceedings of the SIGCHI conference on Human factors in computing systems.* ACM, 1996.

[2] Boritz, James, and Kellogg S. Booth. "A study of interactive 6 DOF docking in a computerised virtual environment." *Virtual Reality Annual International Symposium, 1998. Proceedings., IEEE 1998.* IEEE, 1998.

[3] Ware, Colin, and Jeff Rose. "Rotating virtual objects with real handles." *ACM Transactions on Computer-Human Interaction (TOCHI)* 6.2 (1999): 162-180.

[4] Zeleznik, Robert, and Andrew Forsberg. "UniCam—2D gestural camera controls for 3D environments." *Proceedings of the 1999 symposium on Interactive 3D graphics.* ACM, 1999.

[5] Poupyrev, Ivan, Suzanne Weghorst, and Sidney Fels. "Non-isomorphic 3D rotational techniques." *Proceedings of the SIGCHI conference on Human Factors in Computing Systems.* ACM, 2000.

[6] Conner, Brookshire D., et al. "Three-dimensional widgets." *Proceedings of the 1992 symposium on Interactive 3D graphics.* ACM, 1992.

[7] Li, Yang, et al. "Experimental analysis of mode switching techniques in pen-based user interfaces." *Proceedings of the SIGCHI conference on Human factors in computing systems.* ACM, 2005.

[8] Teather, Robert J., and Wolfgang Stuerzlinger. "Guidelines for 3D positioning techniques." *Proceedings of the 2007 conference on Future Play.* ACM, 2007.

[9] Bi, Xiaojun, et al. "An exploration of pen rolling for pen-based interaction." *Proceedings of the 21st annual ACM symposium on User interface software and technology.* ACM, 2008.

[10] Teather, Robert J., and Wolfgang Stuerzlinger. "Assessing the effects of orientation and device on (constrained) 3D movement techniques." *3D User Interfaces, 2008. 3DUI 2008. IEEE Symposium on.* IEEE, 2008.

[11] Schmidt, Ryan, Karan Singh, and Ravin Balakrishnan. "Sketching and composing widgets for 3d manipulation." *Computer Graphics Forum.* Vol. 27. No. 2. Blackwell Publishing Ltd, 2008.

[12] Bérard, François, et al. "Did "Minority Report" get it wrong? Superiority of the mouse over 3D input devices in a 3D placement task." *Human-Computer Interaction–INTERACT 2009.* Springer Berlin Heidelberg, 2009. 400-414.

[13] Hinckley, Ken, et al. "Usability analysis of 3D rotation techniques." *Proceedings of the 10th annual ACM symposium on User interface software and technology.* ACM, 1997.

[14] Poupyrev, Ivan, et al. "Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques." *Computer Graphics Forum.* Vol. 17. No. 3. Blackwell Publishers Ltd, 1998.