Virtual Environment Interaction Techniques

Mark R. Mine

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175

mine@cs.unc.edu

1. Introduction

Virtual environments have shown considerable promise as a natural (and thus it is hoped more effective) form of human-computer interaction. In a virtual world you can use your eyes, ears, and hands much as you do in the real world: move your head to set your viewpoint, listen to sounds that have direction, reach out your hands to grab and manipulate virtual objects. Virtual worlds technologies (such as head-tracking and stereo, head-mounted displays) provide a better understanding of three-dimensional shapes and spaces through perceptual phenomena such as head-motion parallax, the kinetic depth effect, and stereopsis. Precise interaction, however, is difficult in a virtual world. Virtual environments suffer from a lack of haptic feedback (which helps us to control our interaction in the real world) and current alphanumeric input techniques for the virtual world (which we use for precise interaction in the computer world) are ineffective. We are unfamiliar with this new medium we work in; we do not fully understand how to immerse a user within an application. Before we can create virtual world solutions to real world problems we must learn how to interact with information and controls distributed about a user instead of concentrated in a window in front of him. We must identify natural forms of interaction and extend them in ways not possible in the real world.

The purpose of this paper is to provide the reader with a good understanding of the types of interaction that are possible in a virtual environment. The main body of the paper consists of a discussion of the fundamental forms of interaction and includes numerous examples of interaction techniques that can be used as building blocks in the development of virtual worlds applications.

Included as an appendix to this paper is an overview of coordinate system transformations and examples of using coordinate system diagrams in the implementation of virtual worlds interaction techniques.

Though every effort has been made to avoid a bias towards a particular type of virtual environments system, the paper does assume some form of *display* to present images to a user, a *tracking system* which can be used to measure the position and orientation of the user's head and hand, and some form of *input device* such as a hand-held button device or an instrumented glove which can be used to signal the user's intentions.

2. Interaction in a Virtual World

The goal of this paper is to introduce the reader to the fundamental forms of interaction in a virtual world: *movement, selection, manipulation, and scaling.* From these are derived a fifth form, *virtual menu and widget* interaction. Included for each mode is a description of the interaction task and a listing of the key parameters that must be specified for each mode.

Though there are countless techniques that can be used in implementing each of these modes of interaction, there are several major categories from which the reader can choose:

Direct User Interaction. This includes the use of hand tracking, gesture recognition, pointing, gaze direction, etc. to specify the parameters of the interaction task. Direct user interaction depends upon natural intuitive mapping between user action and the resulting action in the virtual world.

Physical Controls. This includes buttons, sliders, dials, joysticks, steering wheels, etc.. Using physical controls to interact with a virtual world (such as a steering wheel in a driving simulator) can greatly enhance a user's feeling of presence in the virtual environment. Physical device are also well suited for the precise control of an interaction task (fine positioning of an object, for example). Physical controls, however, often lack the natural mappings that facilitate an interaction task. Physical devices also have the drawback that they can be difficult to find while wearing a head-mounted display.

Virtual Controls. Just about anything you can imagine can be implemented as a virtual control. This great flexibility is the key advantage of virtual controls. The disadvantages include a lack of haptic feedback and the general difficulty of interacting with a virtual object. Proper attention to the design of the virtual control and the proper choice of interaction dimensionality is essential.

2.1 Movement

One of the simplest and most natural ways for a user to move through the virtual world is to map movement in the physical world, such as walking, into corresponding motion through the virtual world. The correspondence between physical motion and virtual motion may be one-to-one, or it may be highly exaggerated (for example, one step in the physical world corresponding to a move of 500 light years in a galactic simulation).

The mapping of physical motion to virtual motion is one of the most intuitive means of movement through the virtual world; it requires no special action on the part of the user and provides proprioceptive information which can help the user maintain a better mental model of his current location in the virtual world. Other modes of movement, such as the examples discussed below, require special gestures or actions on the part of the user, lack the proprioceptive feedback of physical motion, and in general are less intuitive for the user (often leaving him lost and disoriented).

The disadvantage of using physical motion to move through the virtual world is that the range of user motion through the virtual world is directly dependent upon the tracking technology in use. Most current systems have usable working volumes of 1 to 2 meters in radius. Even with the development of large area tracking systems (see for example [Ward 1992]), it is likely that the size of an application's virtual space will exceed the physically tracked working volume. This means that some alternate way to move through the virtual world, independent of motion in the physical world, is required. Typically this involves some form of flying, though depending upon your application it may take some alternate form such as driving, or even instant teleportation.

There are two key parameters which must be specified to fully define the user's movement through the virtual world: *speed* and *direction of motion*. Though these could be thought of in terms of a single parameter *velocity*, they are considered here separately to reflect the fact that different mechanisms may be used to specify each parameter.

2.1.1 Direction of Motion

Some of the alternatives for controlling the direction of a user's motion through the virtual world include:

- Hand directed
- Gaze directed
- Physical controls
- Virtual controls
- Object driven
- Goal driven

Hand Directed
In hand directed motion, the position and orientation of the hand determines the direction of motion through the virtual world. There are several variations of hand directed motion.

In *pointing* mode, the direction of motion through the virtual space depends upon the current orientation of the user's hand or hand held input device (see figure 1). i.e. the user simply points in the direction he wishes to fly. The advantage of this mode is that it is extremely flexible and allows arbitrary motion through the virtual world (such as flying backwards while you look around). The problem with this mode is that it can be confusing for novice users who often don't fully understand the relationship between hand orientation and flying direction.

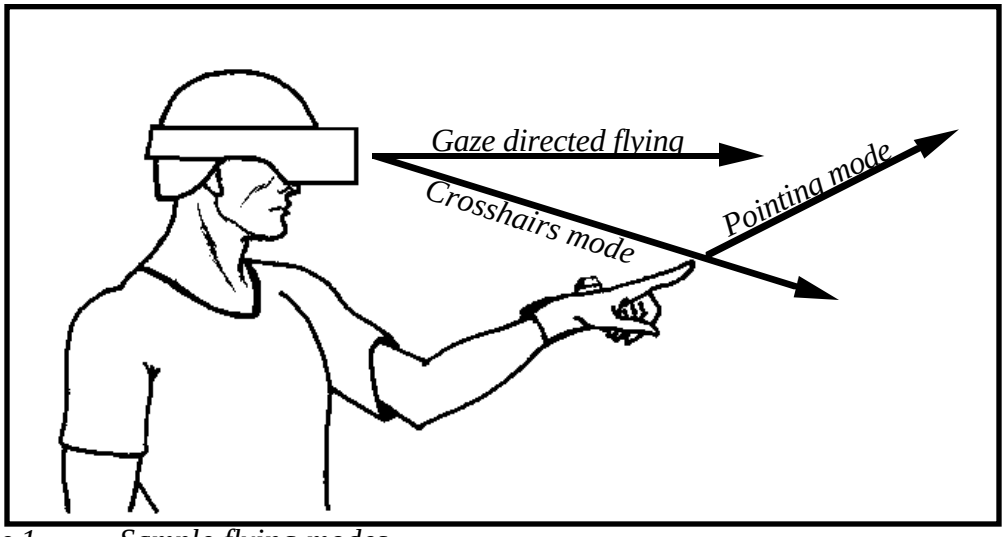

Figure 1. Sample flying modes.

Crosshairs mode was conceived of in the hopes of overcoming some of the difficulties encountered with pointing mode. This mode is intended for novice users who are used to interacting with desktop workstations and personal computers using mice. In crosshairs mode the user simply positions the cursor (typically attached to the user's hand) so that it visually lies on top of the object that he wishes to fly towards. The direction of flight is then determined by the vector from the user's head through the crosshair (see figure 1). Though somewhat simpler to use than pointing mode, this method has the disadvantage

that it requires the user to keep the cursor in line with the desired destination and can lead to arm fatigue.

Dynamic scaling is a clever use of scaling to move through the virtual world (see section 2.4 below for more discussion on scaling). Assuming the user possesses the ability to scale the virtual world up and down (or himself up and down), motion through the virtual world can be accomplished by: 1) Scaling down the world until the desired destination is within reach. 2) Moving the center of scaling (the location in three-space that all objects move away from when scaling up and move towards when scaling down) to the desired destination . 3) Scaling the world back up again. The net result will be a change in location to the destination specified by the location of the center of scaling. The advantage of this technique is that the scaled down world provides a virtual map of the entire environment making it possible to locate your final destination without having to navigate through the virtual world.

Gaze Directed

As an alternative to the hand, the head can be used to specify direction of motion through the virtual world. This is what is known as gaze directed flying (see figure 1). In gaze directed flying, the user flies in whatever direction he is currently looking (typically approximated by the direction his head is pointing). This is very easy to understand and is ideal for novice users. Unfortunately it eliminates the possibility of turning your head to look around while you fly through the virtual world (like looking out a car's windows), since you continually move in the direction you're looking.

An alternative form of gaze directed flying which can be used to view objects from arbitrary angles is known as *orbital mode* [Chung 1994]. In orbital mode, the object of interest is constrained to always appear directly in front of the user, no matter which way he turns his head. The side of the object facing the user depends upon the current orientation of the user's head. Look up and you see the objects bottom, look down you see its top (see figure 2). The object, in effect, orbits the user (thus the name), with its position relative to your head based upon your gaze direction.

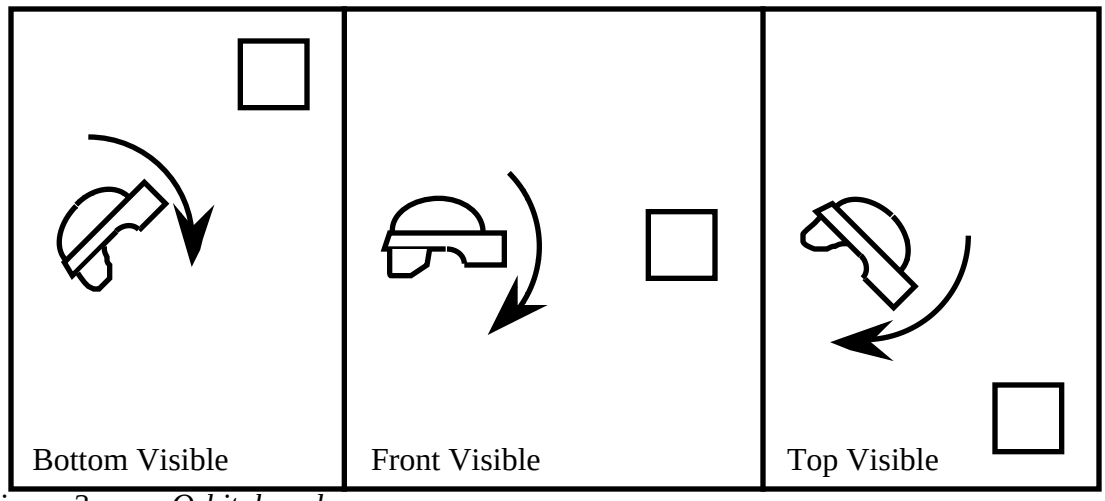

Figure 2. Orbital mode.

Physical Controls

Physical input devices such as joysticks, trackballs, buttons and sliders can be used to specify direction of motion through the virtual world. Though readily available and easy to incorporate into an application, these devices often lack a natural mapping between movement of the device and motion in the virtual world (e.g. should I twist the knob clockwise or counter-clockwise to move up?). Physical devices such as joysticks and dials

are useful, however, if more precise control of the orthogonal components of motion is required.

For certain applications (such as driving simulators) an effective technique is to use actual steering devices, such as steering wheels or handlebars, to effect changes in direction in the virtual world. The addition of haptic feedback greatly enhances the feeling of presence in the simulation, however, registration of the physical controls and their graphical representation is problematical. The virtual mountain bike at the University of North Carolina is an example of this type of application.

Virtual Controls
Instead of physical input devices, one alternative is to implement virtual devices (such as virtual steering wheels or flight sticks) to control your motion through the virtual world. This has the advantage of great flexibility, but interaction with these devices is difficult due to the lack of haptic feedback. One potential application of virtual control devices is in the evaluation of control layouts in actual vehicles such as planes and automobiles.

Object Driven
Sometimes a user's direction of motion is not controlled by the user directly but is influenced instead by objects present in the virtual world. These objects include *autonomous vehicles, attractors, and repellors*. An autonomous vehicle is a virtual object, such as an elevator, which, once entered by the user, automatically moves the user to a new location in the virtual world.

An attractor is an object which draws the user towards itself wherever it presides in the virtual environment (such as a planet with a simulated gravity). A repellor is the opposite of an attractor. A repellor pushes the user away from itself, for example in a direction of motion which extends radially from the center of the repellor. Attractors and repellors may be autonomous (like a planet revolving around the sun) or they may be under user control. If under user control, an attractor can be used to move the user through the virtual world. To do so, the attractor is moved to the desired destination in the environment, is turned on, and the user is then drawn to that location. The disadvantage of this scheme is that it requires some additional means of controlling the position of the attractor.

Goal Driven
In a goal driven system, the user is presented a list of destinations which are displayed either as text or as a set of icons. To fly to a destination, the user simply picks the destination from the list, at which point he is automatically moved to that location. This necessitates some form of virtual menu interaction (see section 2.5 below).

Instead of displaying the destinations in a list, the potential destinations can be displayed graphically in the form of a virtual map. Using the virtual map, the user can point directly at the desired destination and then be transported there under program control.

2.1.2 Specification of Speed of Motion

In addition to specifying direction of motion, the user must also specify speed of motion along that direction. Several options for the specification of speed of motion exist and are discussed in detail below.

- Constant speed
- Constant acceleration
- Hand controlled
- Physical controls
- Virtual controls

Constant speed:
The simplest mechanism for specifying speed of motion through the virtual world is to provide no option but rather have the user move at a constant speed whenever he is flying. This speed can be based upon the size of the virtual space the user must traverse (adjusted such that the user can traverse the space in some reasonable amount of time). This technique allows you to control the speed with which a user can traverse a space, but makes it difficult to successfully navigate through the environment (leading to a jerky, start/stop kind of motion and a tendency to overshoot your destination).

Constant acceleration:
Instead of a constant speed, the user can fly with a constant acceleration. The user begins at a slow speed that is reasonable for short flights through the local environment, but as long as the user continues to fly, he continues to accelerate. This enables the user's speed to grow exponentially with flight duration, and depending upon the rate of acceleration, will allow the user to quickly reach tremendous speeds. This type of speed control is useful if the size of the environment is very large and yet contains lots of interesting local detail that would be interesting to fly around. Determination of the proper rate of acceleration is a tricky aspect of implementing this type of speed control. Set it too high and you quickly zoom away from objects in your immediate neighborhood. Set it too low and it may take an unacceptably long time to reach objects on the other side of the environment.

Hand Controlled:
The use of hand position as a throttling mechanism is an adaptable form of speed control. For example you can set your speed based on how far your hand is extended in front of your body (see figure 3). Hold your hand in close and you move slowly, extend it fully and you fly at top speed. Any mapping function can be used to convert the intermediate positions of the hand to flying speed, typically linear or exponential mappings are used. The main limitations of this method include fatigue from having to hold your arm at the required distance and limitations in the dynamic range that can be compressed into the possible range of motion of the arm and yet still be controllable.

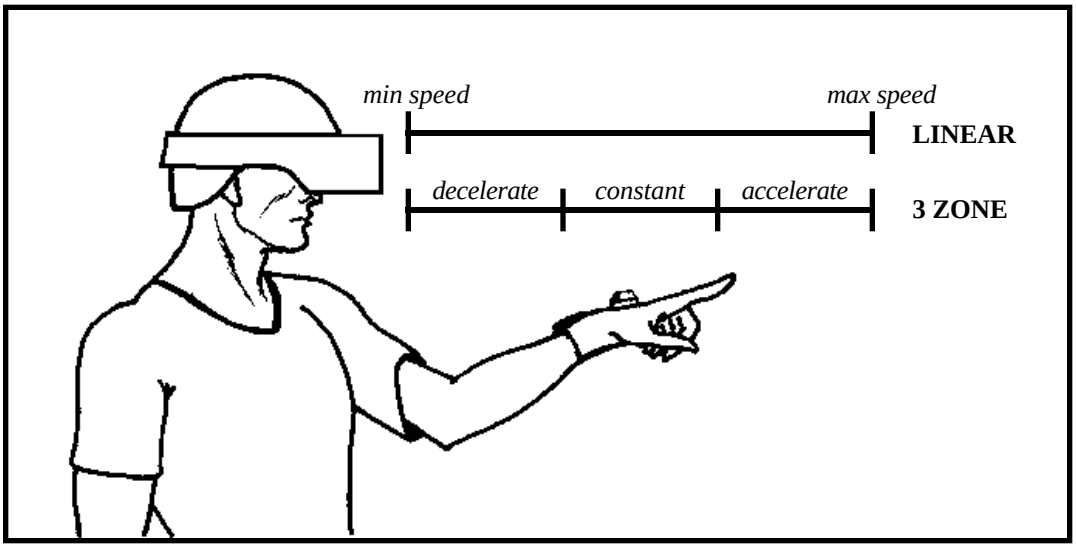

Figure 3. Hand controlled flying speed.

An alternative mode of gestural control is to define three zones at varying reach distances. One zone will maintain the user at a constant speed, another will maintain a constant acceleration, and the third will result in constant deceleration (see figure 3). In this scheme,

greater dynamic range is obtained at the cost of ease of use, the system being somewhat confusing for novice users.

Physical controls:
External input devices can be used for speed control. Alternatives include: keyboard input at a workstation, voice control, actual speed controls such as an accelerator pedal or a slot-car controller, or some form of dial or slider mounted on the hand-held input device. The treadmill used in early virtual environments research at UNC is an example of a physical control device. The speed the user moved through the virtual environment depended upon how fast the user walked on the treadmill. [Airey 1990]

Virtual Controls:
Not surprisingly, speed control can be implemented using some form of virtual control device. The device may emulate some physical analog (a throttle lever for example) or may be in the form of virtual menus/sliders which can be controlled by the user. Again, these devices suffer from a lack of haptic feedback.

2.1.3 Implementation Issues

One of the most important factors to consider when dealing with the specification of motion is the number of degrees of freedom that are under user control. Too few and the user will have difficulty reaching his desired destination, too many and the user can quickly lose control of his motion through the virtual space.

The fundamental problem that must be dealt with is the fact that three-dimensional space is big! There are many ways, when navigating around a virtual world, to get lost and disoriented. The way to deal with this is to constrain the types of motion possible by the user through the virtual space.

In many applications, for example, it is beneficial to restrict the user to changes in position and to not allow changes in orientation as he flies about the environment. In doing so, the relative orientation of the virtual world and the physical world will remain fixed and will remain consistent with the user's sense of "down" in the real world (due to the pull of gravity). This is like standing on a platform that can move anywhere in the virtual space, but is constrained to remain level relative to the virtual world. The user is still free to walk around on this platform, to look in any direction, and to tilt his head in any orientation. The platform's orientation, however, will remain fixed in the virtual world. Giving the user total freedom to change his orientation relative to the virtual world can be great fun (allowing you to perform barrel rolls and loops in a virtual airplane, for example), but it makes it very difficult to re-orient yourself later on.

Even the ability to translate in all directions may be excessive. In an architectural walkthrough, for example, it might be preferred that the user remains at some typical head height rather than zooming up towards the ceiling and down towards the floor. In this case, some form of "sliding" motion should be implemented that forces the user to remain at a constant height in the virtual world. Similarly, the user may be forced to remain at a fixed distance above a surface regardless of its height and orientation. This "terrain following" is excellent for outdoors simulations and landscape flythroughs. Finally, in certain situations it is preferable to take control of the user's motion, guiding him through the virtual world under program control.

2.2 Selection

Interaction with virtual objects (such as grabbing) requires some form of object selection, i.e., some way to indicate the target of the desired interaction. Object selection assumes

that the model database has been divided into identifiable components, each with some form of unique ID. In all cases, object selection requires 1) some mechanism for the identification of the object to be selected, and 2) some signal or command to indicate the actual act of selection. This signal is typically in the form of a button press, a gesture, or some kind of voice command.

2.2.1 Selection Techniques

There are two primary selection techniques: *local* and *at-a-distance.* In a local technique, the desired object is within reach and the user can interact with it directly. When the user can not reach an object, action-at-a-distance selection is required.

Local:
In a local selection mode, objects are chosen for selection by moving a cursor (typically attached to the user's hand) until it is within the object's selection region (for example a minimal bounding box), see figure 4. Once chosen, the object can be selected using some pre-defined signal such as a gesture, button press, or voice command as discussed above.

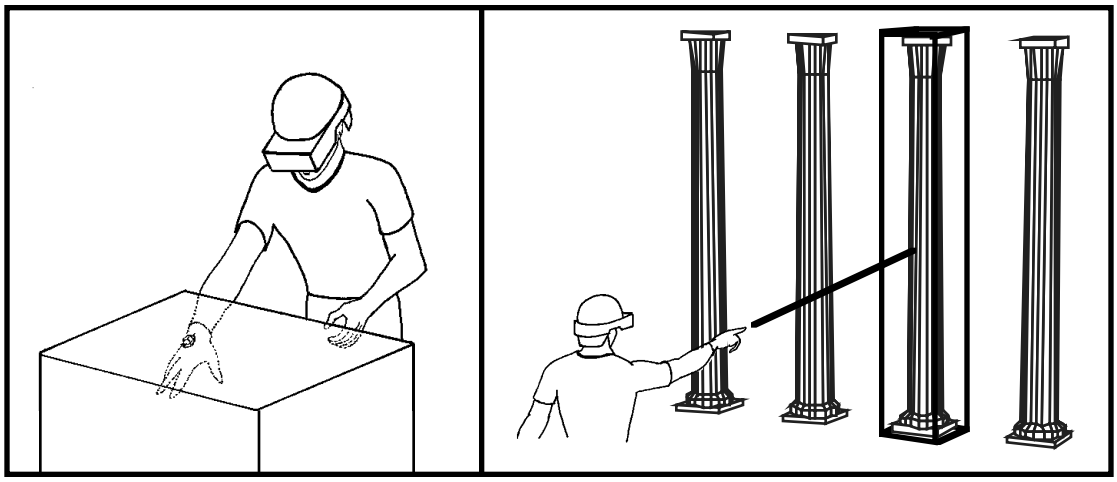

Figure 4. Local vs. At-a-distance selection

At-a-distance:
The selection of objects which fall outside of the immediate reach of the user can be accomplished using laser beams or spotlights which project out from the user's hand and intersect with the objects in the virtual world (see figure 4). Alternately, some form of virtual cursor or drone can be moved by the user through the environment until it is within the selection zone of the desired object (at which point the object can be selected using gesture, button press, or voice input).

Gaze Directed:
Object selection can be based upon the user's current gaze direction; the user merely looks at an object to be selected and then indicates his selection via the standard selection signal. In the absence of a reliable form of eyetracking this can be approximated using the current orientation of the user's head (e.g. the user turning his head to line up a target object and a cursor floating in the middle of his field-of-view).

Voice input:
If each object has some form of name or ID which is known to the user, voice input can be used to identify objects to be selected and to signal actual selection. To select an object the user would issue a command such as "Red Box - select". The main issues to be dealt with in such a system include the mental overhead of having to remember the names of all

objects (or increased clutter in the virtual world due to the need to label all objects) and the reliability of current voice recognition systems. Put-That-There [Bolt 1980] was an interesting example of using voice input combined with pointing to select objects.

List selection:
As an alternative to voice recognition, some form of virtual selection list may be presented to the user. Once again, in order for a user to select an object he must know its name. The advantage of this system, however, is that the user does not have to be able to see the object to select it.

2.2.2 Implementation Issues

When implementing object selection, it is essential to incorporate adequate feedback. The user must know when he has chosen an object for selection (perhaps by highlighting the object or its bounding box), must know when he has successfully performed a selection action (via both audible and visual cues) and must be able to determine the current selection state of all objects (for example using color coded bounding boxes or changing object color).

The selection of small or distant objects can be facilitated by several enhancements. The selection of small objects can be simplified by allowing the user to work at different scales, expanding the world to select a small subcomponent for example. Spotlights, whose cross-section is proportional to distance from the user [Liang 1994], can help in the selection of distant objects. Hysteresis can overcome difficulties in both local and at-a-distance selection that result from noise in the tracking system.

2.3 Manipulation

One of the most important forms of interaction is the specification of an object's position and/or orientation in the virtual world. This interaction can be realistic, the user grabs and moves a virtual object as he would grab and move objects in the real world, or the user can move objects in ways that have no analog in the physical world.

Three parameters must be specified when manipulating an object: its *change in position*, its *change in orientation* and its *center of rotation.*

2.3.1 Change in Position/Orientation

Hand specified:
One of the most intuitive means available to change the position and orientation of a virtual object is to allow the user to "grab" it (typically signaled by a button press or a grab gesture), and move it as though he were moving an object in the real world. Grabbed objects can move in a 1:1 correspondence with the user's hand, or to allow a greater range of object motion, an amplification factor can be applied to the motion of the object. Depending on the magnitude of the amplification factor this can result in fine or gross positioning of the object. The amplification factor can either be specified separately or can be based upon some factor such as the speed of the user's hand motion (analogous to the *control-to-display ratio* used in mouse interaction, see [Foley 1990] p 351).

Physical controls:
Object position and orientation can also be controlled via some form of external input device such as a joystick, slider or dial. Though excellent for precise positioning of objects (since each degree of freedom can be controlled separately), these methods lack natural mappings and can make it difficult to place an object at some arbitrary position and orientation.

Virtual controls:
Just as physical input devices can be used to position an object, virtual control devices such as sliders, buttons and dials in a virtual menu or toolbox can be used to position the selected object in the virtual world. Objects can also be manipulated in the virtual world through interaction with other virtual objects. A virtual golf club could be used to hit a virtual golf ball, or a virtual bowling ball could be used to knock down virtual pins.

A more complicated way to manipulate virtual objects is to implement some form of virtual device, such as a telerobotic arm, to interact with and position the object. The motion of the object will depend upon the mapping of the device's virtual controls. Another example is some form of virtual drone (like a flying tugboat) which can be used to grab and position objects within the virtual world while flying around the virtual space under user control.

2.3.2 Center of Rotation

One of the key differences between the different manipulation schemes is the location of the center of rotation. Hand centered rotation (i.e. all selected objects pivot about the hand) is the scheme most directly related to the manipulation of objects in the real world. In some cases, however, it is desirable to have some remote center of rotation. This allows the user to stand back and have a good perspective as he sets the orientation of an object. Typical alternatives include rotation about the center of an object or rotation about some user specified center of rotation (for example you could set the center of rotation of a virtual door to be the center of the hinge axis).

2.3.3 Implementation Issues

The absence of constraints is one of the biggest problems encountered in the manipulation of virtual objects. Without constraints, users are restricted to gross interactions and are unable to perform any form of precise manipulation.

Two types of constraints can be used for more controlled manipulation: virtual and physical. Virtual constraints are those in which extraneous degrees of freedom in the user's input are ignored, for example filtering out any change in orientation of the user's hand as he positions an object. Physical constraints are actual physical objects which are used to restrict the motion of the user's hand, for example a desktop on which the user can slide the input device.

Constraints should be overridable with resets. A three-dimensional mouse with a flat bottom can be used to specify user viewpoint within an architectural walkthrough, for example (see [Airey 1990]). When slid along a desktop, the input device, and thus the user's viewpoint, is constrained to move in a plane with an upright orientation (due to the mouse's flat bottom). Arbitrary view orientations and out-of-plane translations, however, can be achieved, by picking up the mouse, overriding the constraints of the desktop. The view orientation can be quickly reset to upright by placing the mouse back on the desktop.

2.4 Scaling

Another useful mode of interaction in a virtual world is scaling. Scaling can be used in the interactive construction of a virtual world to get components at their correct relative scale. Alternately it can be used in the exploration of an environment to allow a user to view some small detail by scaling up the selected object or to get a better global understanding of the environment by scaling it down and viewing it as a miniature model. The key parameters to define in a scaling operation are *the center of scaling* and the *scaling factor.*

2.4.1 Center of Scaling

The center of scaling determines the behavior of the scaling operation. It is the point which all objects move towards when you scale down and all points move away from when scaling up. In hand centered scaling, all selected objects will scale about the current location of the hand. For object centered scaling, the center of scaling is defined to be the center of the selected object.

Alternately, objects can scale about some user defined center of scaling. User defined centers of scaling can be specified via direct interaction (e.g. the user can grab some icon representing the center of scaling and move it about) or by using some remote agent which the user moves (via remote control) to specify the desired center of scaling.

2.4.2 Scaling Factor

Hand controlled:
There are several different options for the specification of scaling factor. The scaling factor can be *hand specified* where movement of the hand determines the scaling factor of the selected objects. For example, movement of the hand up could signify a scale-up action, movement of the hand down could signify a scale-down action, and the range of hand motion could determine the magnitude of the scaling action. This is particularly effective for uniform scaling. As with object manipulation the use of a *control -to-display ratio* based on speed of hand motion can be used to adjust the scaling factor.

Alternately, you can provide affordances or handles which the user can interact with to control the scaling of the selected object. This is similar to the handles found in most conventional desktop drawing programs which the user "grabs" and moves to define the scale of the selected object. These handles are typically the vertices of the selected objects bounding box. This method is particularly well suited for the implementation of non-uniform scaling

Physical controls:
Physical input devices can be used to control object scaling. Pressing a button on an input device, for example, can indicate the start of a scale up or down action, with the duration of the button press controlling the magnitude of the scale. Alternately some form of slider or dial can be used to specify the scaling factor.

Virtual controls:
Virtual controls can also be used to specify scaling factor. Menus with dials or sliders are but two alternatives.

2.4.3 Implementation Issues

Two primary factors to be considered in the implementation of scaling include: uniform vs. non-uniform scaling and user vs. object scaling.

In uniform scaling, an object is scaled along all three dimensions equally. Non-uniform scaling allows the user to control the scale of a selected object along each dimension separately. For many applications it is not necessary to deal with the added complexity of specifying non-uniform scaling (in particular where the main purpose of scaling is to allow the user to get a close look at some fine detail or to get a god's eye view of the scene), uniform scaling will suffice. Advanced applications such as immersive modeling systems [Butterworth 1992], however, may require non-uniform scaling to help in the generation of arbitrary shapes.

In certain instances it is desirable to scale selected objects, but in other cases the desired operation is the scaling of the user. Scaling of the user changes the apparent size of objects

in the world without affecting their size in the model database. As the user gets smaller, it will appear to him that the world is getting larger (and vice-versa). This allows the user to explore fine detail of an object or get a global view without modifying the actual size of the objects in the database.

2.5 Menu Interaction

The incorporation of menus and toolboxes into the virtual environment is a powerful way to add function to a virtual environment application. Menus enable the user to perform operations that may be difficult to specify using direct interaction. The primary difference between various menu schemes is the dimensionality of the selection mechanism, that is to say, the number of dimensions specified by the user to select different menu options.

2.5.1 Menu Dimensionality

One-dimensional menus are ideal for the selection of a single item from a limited set of options. In a one dimensional menu, the user moves from option to option by modifying only a single parameter. This can be as simple as clicking a button to cycle through options or a more sophisticated gestural based scheme where options are selected based upon the change of a single dimension of the position or orientation of the user's hand. In a rotary tool selector, for example, a user selects his current tool by rotating his hand about a chosen axis (like turning a dial). Rotation of the hand causes the tools, displayed in an arc about the hand, to slide past a selection box. A tool is selected when it falls within the selection box (see [Liang 1994]). The important thing to remember is that only changes along the chosen dimension are critical in the selection of the tool, all other changes are ignored. This simplifies interaction with the one-dimensional menu, but limits the number of options which can be handled in this way.

When the number of options to choose among grows larger and can no longer fit within a one-dimensional menu, the next logical step is the implementation of a two-dimensional menu. In a two-dimensional menu, hand motion results in the motion of a selection cursor in the menu's two-dimensional plane. This method of interaction is directly analogous to the interaction found in conventional desktop workstations. The two-dimensional position of the cursor can be determined, for example, by a line based upon the current orientation of the user's hand (i.e. pointing) or by a vector from the user's head, through his hand, projected onto the current menu plane (analogous to the crosshairs mode of flying)

The addition of a third dimension adds both power and problems to the task of interacting with a menu [Jacoby 1993]. On the one hand, the use of three-dimensional menus allows you to create three-dimensional widgets, objects with affordances that can be used in controlling virtual objects[Conner 1992]. Widgets help enhance object interaction and control since they are registered in three-space with the objects being controlled. On the other hand, the addition of a third-dimension makes it more difficult for the user to interact with his menus since he must correctly match his hand position with all three dimensions of the target coordinates. This is why all interaction with conventional menus (such as pull down menus and dialog boxes which are two-dimensional entities) should avoid three-dimensional interaction at all costs.

2.5.2 Implementation Issues

The principles of virtual menu design are directly related to the guiding principles of two-dimensional user-interface design: consistency, feedback, natural mappings, good visibility, etc. Virtual menus, however, have additional problems that result from having to work in a three-dimensional environment.

One of the most important of these problems concerns the placement of the menu in the virtual environment. Where should the menu be located? One option is to have the menu floating in space like any other virtual object. Whenever the user wishes to move the menu, he must grab it, and move it to its new location. This makes it easy to move the menu out of the way when you're working on something, but it suffers from the problem that it is easy to lose track of the menu's location in space. Having to search around your environment to find your menu can be quite frustrating. As an alternative, the menu can be constrained to always float in front of the user's face (no matter which way he turns his head). Though this solves the problem of having to find the menu, it has the drawback that the menu often ends up blocking the user's view into the virtual world. A hybrid solution is to have the menu floating in space like a regular 3D object with some additional means of quickly snapping the menu into view. Press on a special button on your input device or make a special gesture with your hand and the menu pops up in front of your face. Since the menu is a free-floating, 3D object, however, you can still look around the menu by moving your head.

Limitations in current virtual environments hardware also have a significant impact on the design of virtual menus. The poor image quality of current head-mounted displays, for example, makes it next to impossible to discern between various menu choices. Noise and distortion in tracking data makes it difficult to make menu selections and to interact with virtual widgets. These issues must be dealt with through the proper selection of menu dimensionality and the design of virtual widgets that compensate for limitations in the current technology. A widget for positioning virtual objects, for example, might damp out instability in the user's hand and noise in the input tracker data by forcing the user to move the object by moving a large lever arm (thus mapping large movements of the user's hand to small movements of the object) .

3. Conclusions

The goal of this paper has been to show that there are numerous ways to implement the fundamental forms of interaction in a virtual world: *movement, selection, manipulation, and scaling.* Each form of interaction has a unique set of parameters which can be specified in many different ways. Overall there are three main techniques of parameter specification used in virtual worlds interaction:

- Direct user interaction
- Physical controls
- Virtual controls

Direct user interaction is the use of gestures of the head and hand to specify interaction parameters. Though flexible and intuitive, direct user interaction depends on the determination of natural, intuitive mappings between user actions and the resulting actions in the virtual world. Physical controls include the use of devices such as buttons, dials, sliders, and steering wheels. Physical control devices provide haptic feedback, enhancing presence and facilitating precise control, but they lack flexibility. Virtual controls include any type of control device presented as a virtual object and thus are highly flexible. Virtual controls, however, suffer from the lack of haptic feedback and the general difficulty of interacting with virtual objects.

It is hoped that the information presented in this paper will give the reader a better understanding of the possibilities of working in a virtual environment and that it will help him to create virtual environment applications with more natural and intuitive interfaces. It is also hoped that the reader will be encouraged by the information in this paper to explore new ways of interaction in the virtual world.

4. References

Airey, J.M. (1990). *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations.* Ph.D. Thesis, University of North Carolina, Chapel Hill, NC. TR 90-027

Bolt, R. (1980). Put-That-There, *ACM Computer Graphics: Proceedings of SIGGRAPH 80,* pp. 262-270.

Butterworth, J., A. Davidson, S. Hench, and T. M. Olano (1992). 3DM: A Three Dimensional Modeler Using a Head-Mounted Display. *ACM Computer Graphics: Proceedings of 1992 Symposium on Interactive 3D Graphics,* Cambridge, MA, 135-138

Chung, J. (1994). *Intuitive Navigation in the Targeting of Radiation Therapy Treatment Beams*. Ph.D. Thesis, University of North Carolina, Chapel Hill, NC.

Conner D., S. Snibbe, K. Herndon, D. Robbins, R. Zeleznik, A. van Dam (1992). Three-Dimensional Widgets, *ACM Computer Graphics: Proceedings of 1992 Symposium on Interactive 3D Graphics*, Cambridge, MA, 197-208.

Foley, J., A. van Dam, S. Feiner, J. Hughes (1990). *Computer Graphics: Principles and Practice* (2nd ed.). Addison-Wesley Publishing Co., Reading MA. 201-226

Jacoby, R., S. Ellis. (1993). Using Virtual menus in a virtual environment. *SIGGRAPH course notes: Implementing Virtual Reality*, course 43.

Liang, J., M. Green (1994). JDCAD: A highly interactive 3D modeling system. *Computers & Graphics: Proceedings of the Conference on Computer Aided Design and Computer Graphics,* Beijing, China, 499-506

Robinett, W., R. Holloway (1992). Implementation of flying, scaling, and grabbing in virtual worlds, *Proc. 1992 Symposium on Interactive 3D Graphics,* Cambridge MA, March, 189-192.

Ward, M., R. Azuma, R. Bennett, S. Gottschalk, H. Fuchs (1992). A Demonstrated Optical Tracker with Scalable Work Area for Head-Mounted Display Systems. *ACM Computer Graphics: Proceedings of 1992 Symposium on Interactive 3D Graphics,* Cambridge, MA, 43-52

APPENDIX

Coordinate System Transformations for Virtual Environments

One of the most difficult aspects of implementing a virtual environment application is learning how to deal with the seemingly endless number of coordinate systems used in creating a virtual world. Each user has separate coordinate systems for their head, their hands, their eyes, the screens in front of their eyes, the room they're in, and the tracking system they use. There is a separate coordinate system for the world, for each movable object in the world, and for each movable part in an object. Dealing with and understanding these coordinate systems (and more importantly the relationships between these coordinate systems) can be a daunting and frustrating task.

Probably the best way to begin is by reading *Implementation of Flying, Scaling, and Grabbing in Virtual Worlds* by Warren Robinett and Richard Holloway of the University of North Carolina [Robinett 1992]. In that paper, Robinett and Holloway present a systematic approach for dealing with the multitude of coordinate systems found in a virtual world. Most importantly they teach you how to implement various actions in the virtual world (such as flying, scaling and grabbing) through the use of frame-to-frame invariants (discussed below).

Since coordinate system transformations are essential for the implementation of virtual world interaction, I have included a brief summary of their techniques as this Appendix.

A.1 Definition of a Transformation

A transformation is defined as the instantaneous relationship between a pair of coordinate systems. In other words, given two separate coordinate systems, A and B, the transformation T_{A_B} defines the relative position, orientation and scale of coordinate systems A and B. More precisely, the transformation T_{A_B} is defined as the transformation *from* coordinate system B *to* coordinate system A. Thus given a point P_B in coordinate system B, the transformation T_{A_B} can be used to convert this point, P_B, into a point P_A in coordinate system A.

$$P_A = T_{A\_B} \cdot P_B$$

Take careful note of the ordering of the subscripts in transformation T_{A_B}. This is the transformation *from* coordinate system B *to* coordinate system A and not vice-versa. Though possibly counter-intuitive (since the subscripts are read from left to right), this notation has the desirable property that the subscripts cancel out when composing transformations. For example, given a transformation from coordinate system B to A (T_{A_B}) and a transformation from coordinate system C to B (T_{B_C}), the transformation from coordinate system C to A can be found by composing the two transformations.

$$T_{A\_C} = T_{A\_B} \cdot T_{B\_C}$$

For a more detailed treatment of transformations see reference [Foley 1990].

A.2 Coordinate System Graphs

A tool that helps in the understanding of coordinate systems and their relationships in a virtual world is the coordinate system diagram. The coordinate system diagram is a graphical representation of the various coordinate systems in a virtual world and their relationships. In a coordinate system diagram, nodes represent coordinate systems and

edges represent transformations between coordinate systems. A coordinate system diagram for a typical virtual environments application at the University of North Carolina (UNC) is presented in figure 5.

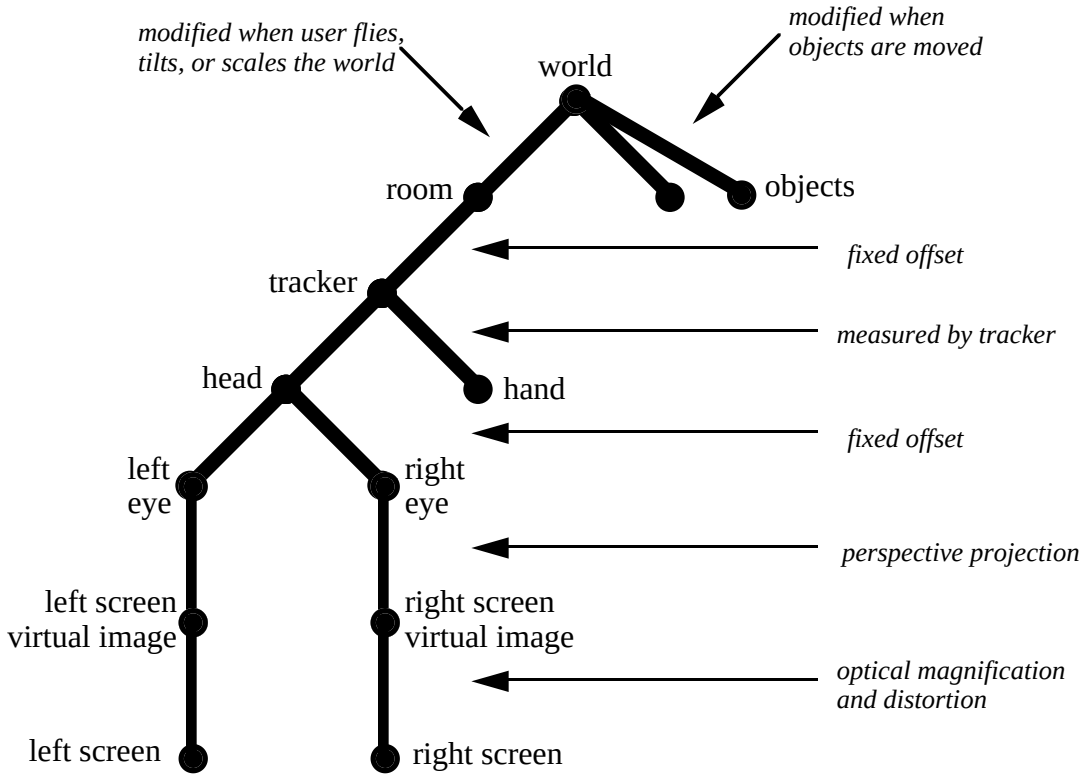

Figure 5. Coordinate system diagram for single-user head-mounted display system.

A.3 Specifying Actions With Invariants

Coordinate system diagrams can be used in conjunction with what are called *frame-to-frame invariants* to systematically determine the transformations involved in the computation of any virtual world interaction. An invariant is a relation between a set of transformations in the current frame and a (not necessarily the same) set of transformations in the previous frame.

For example, as will be shown below, one way to "grab" an object in the virtual world is to specify that the transformation between the object coordinate system and the hand coordinate system remains unchanged from frame to frame.

$$T_{Object\_Hand}' = T_{Object\_Hand}$$

(where the apostrophe in T_{Object_Hand}' indicates a transform in the current frame).

By using the section of the coordinate system diagram presented in figure 6 (next page), we can see the T_{Object_Hand} in fact represents a sequence of transformations between coordinate systems:

$$T_{Object\_Hand} = T_{Object\_World} \cdot T_{World\_Room} \cdot T_{Room\_Tracker} \cdot T_{Tracker\_Hand}$$

These transformations can be substituted into the equation above to yield:

$$T_{Object\_World}' \cdot T_{World\_Room}' \cdot T_{Room\_Tracker}' \cdot T_{Tracker\_Hand}' = T_{Object\_World} \cdot T_{World\_Room} \cdot T_{Room\_Tracker} \cdot T_{Tracker\_Hand}$$

This equation can then can be solved to determine T_{Object_World}' the inverse of the transformation T_{World_Object}' which defines the objects new position in the world after the grab operation.

$$T_{Object\_World}' = T_{Object\_World} \cdot T_{World\_Room} \cdot T_{Room\_Tracker} \cdot T_{Tracker\_Hand} \cdot T_{Hand\_Tracker}' \cdot T_{Tracker\_Room}' \cdot T_{Room\_World}'$$

The algebraic manipulations used to derive this equation make use of the fact that the inverse of a transform such as $T_{Tracker\_Hand}'$ is the transform $T_{Hand\_Tracker}'$.

Examples of the use of these frame-to-frame invariants (and the resulting transformation equations) are presented below to demonstrate the implementation of the various modes of interaction discussed in this paper.

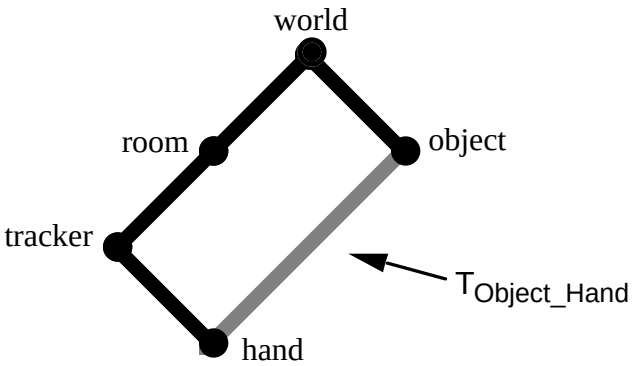

Figure 6. Definition of T_{Object_Hand}.

A.4 Examples

A.4.1 Pointing Mode Flying

As was discussed above, translation of the user's room-space (see the coordinate system diagram in figure 5) through the virtual world is one way to implement flying. To do this we must update the transformation T_{World_Room} to some new value T_{World_Room}'. This will translate the origin of room-space relative to the world coordinate system. This translation can be specified by a transformation $T_{Room<translate>Room}$ such that:

$$T_{World\_Room}' = T_{World\text{-}Room} \cdot T_{Room<translate>Room}$$

As described in section 2.1.1, one way to specify the direction of user motion is *pointing* mode, where you base your flying direction on the current orientation of the user's hand. The user's speed of motion can be determined using any one of the methods discussed in section 2.1.2. Together, these two pieces of information define a transformation $T_{Hand<translate>Hand}$ which specifies a change of position in hand-space. To convert this change of position in hand-space to a change of position in room-space we use the transformation T_{Room_Hand} and its inverse T_{Hand_Room}. This yields:

$$T_{Room<translate>Room} = T_{Room\_Hand} \cdot T_{Hand<translate>Hand} \cdot T_{Hand\text{-}Room}$$

Note the cancellation of subscripts. This equation can then be substituted into the one for T_{World_Room}' to yield:

$$T_{World\_Room}' = T_{World\text{-}Room} \cdot T_{Room\_Hand} \cdot T_{Hand<translate>Hand} \cdot T_{Hand\text{-}Room}$$

A.4.2 Selection

Implementation of selection is relatively simple in terms of the transformations that must be used, but may be expensive computationally (depending on the number of objects to be tested for selection). To select an object using *local* mode (by moving your hand within the objects bounding box) you must simply compare the relative location of the hand and the object within world-space:

$$\text{near}(T_{World\_Hand}, T_{World\_Object})$$

T_{World_Hand} and T_{World_Object} are the locations of the hand and the object being tested in world space, and near() is some function which implements the selection criteria test (checking to see if the hand is in the object's bounding box, for example). By examining the coordinate system diagram (figure 5) we see that T_{World_Hand} can be determined by the composition of the following transformations:

$$T_{World\_Hand} = T_{World\_Room} \cdot T_{Room\_Tracker} \cdot T_{Tracker\_Hand}$$

T_{World_Room} is the current location of the user in world space, $T_{Room\_Tracker}$ is the location of the tracker origin in room space, and $T_{Tracker\_Hand}$ is the current location of the hand in tracker space (as detected by the tracking system).

A.4.3 Hand Centered Grabbing

Hand-centered grabbing can be implemented by using the invariant:

$$T_{Object\_Hand}' = T_{Object\_Hand}$$

which can broken down into its component transformation and then solved to determine T_{Object_World}' the inverse of the transformation T_{World_Object}' which defines the objects new position in the world after the grab operation.

$$T_{Object\_World}' = T_{Object\_World} \cdot T_{World\_Room} \cdot T_{Room\_Tracker} \cdot T_{Tracker\_Hand} \cdot T_{Hand\_Tracker}' \cdot T_{Tracker\_Room}' \cdot T_{Room\_World}'$$

A.4.4 Hand Centered Scaling

To scale an object about the user's hand, we define $T_{Hand<scale>Hand}$, a transformation in hand-space which specifies the desired scaling factor. In a manner analogous to that used for flying, we can determine the new T_{World_Object}' transformation (which includes the desired scaling of the object) as:

$$T_{World\_Object}' = T_{World\_Hand} \cdot T_{Hand<scale>Hand} \cdot T_{Hand\_World} \cdot T_{World\_Object}$$

$T_{Hand<scale>Hand}$ is simply an identity transformation (no translation or rotation component) with the specified scale factor applied. Recall that T_{World_Hand} is defined as:

$$T_{World\_Hand} = T_{World\_Room} \cdot T_{Room\_Tracker} \cdot T_{Tracker\_Hand}$$

T_{Hand_World} is simply the inverse of this.