THRED: A Two-Handed Design System

Chris D Shaw, Mark Green

Department of Computing Science,
University of Alberta
Edmonton, Alberta T6G 2H1, Canada
Tel. 403-492-7418
{cdshaw,mark}@cs.ualberta.ca

[1]

Abstract

This paper describes a Computer Aided Design system for sketching free-form polygonal surfaces such as terrains and other natural objects. The user manipulates two 3D position and orientation trackers with three buttons, one for each hand. Each hand has a distinct role to play, with the dominant hand being responsible for picking and manipulation, and the less-dominant hand being responsible for context setting of various kinds. The less-dominant hand holds the workpiece, sets which refinement level that can be picked by the dominant hand, sets the constraint mode and the reshape operator, and generally acts as a counterpoint to the dominant hand. In this paper, the architecture of the system is outlined, the interaction techniques are presented, and a simple surface is shown.

KEYWORDS: User Interface Software, Virtual Reality, Interactive 3D Graphics, Two Handed Interfaces, Free-Form Surfaces, Geometric Modeling.

1 Introduction

The design of free-form surfaces such as terrains and other natural objects is difficult and time-consuming. Part of the reason for this is that the task domain is 3D space, and designing 3D objects is difficult in general. Traditional CAD systems based on 2D I/O devices such as mice, tablets and video displays make

[1] Chris D Shaw's Present Address: Department of Computer Science, University of Regina, Regina, Saskatchewan, S4S 0A2, Canada, Tel. 306-585-4632, cdshaw@cs.uregina.ca

the design task still more difficult because these devices each have one degree of freedom less than the design domain. More recent CAD systems using a single 3D tracker do not take advantage of all the user's capabilities because they ignore the presence of the user's other hand. Broadly speaking, the two areas in which traditional CAD systems fall short are in the perception of 3D shape (output), and in the direct manipulation of 3D data (input).

On the output side, the advent of graphics computers capable of rendering more than 500,000 texture-mapped triangles per second allows the possibility of real-time animation of complex 3D objects. Texture mapping and animation give vital information about the 3D shape of an object, and their intelligent deployment can help the user to better understand the 3D shape of the objects being edited.

On the input side, 3D objects can be more easily manipulated by 3D tracking devices than with a mouse [28], because the user does not have to mentally break down the 3D task into a sequence of 2D operations. Using a 3D device allows the user to directly manipulate the objects of interest without intermediate steps. Using two 3D devices gives the user access to double the spatial bandwidth, since both hands can be employed in parallel to quickly achieve the desired operation. Of course, some operations in a CAD system are naturally 2D or 1D, so the interaction techniques that use 3D trackers should be able to constrain the tracker motion to a line or a plane when necessary.

We have built a Computer Aided Design system called THRED (for Two Handed Refining EDitor) for sketching free-form polygonal surfaces such as terrains and other natural objects. We chose polygonal surfaces because our goal is to design terrains for virtual environments, which must be rendered in real-time. The real-time requirement precludes the use of smooth surfaces such as bicubic patches[12], and since THRED uses real-time display of the surface to help the designer understand its shape, the designer immediately gets a feel for whether this terrain is too complex to display in VR. THRED continually animates the 3D object being sketched, and uses two Polhemus magnetic 3D position and orientation trackers with added buttons to input commands and to manipulate the control points of the 3D surface.

The user sits in front of a graphics console that has a screen, keyboard, mouse, and the two 3D sensors. Each 3D sensor has a distinct role, with the dominant hand being responsible for picking and manipulation, and the less-dominant hand being responsible for context setting of various kinds. For the sake of rhetorical convenience, we will refer to the dominant hand as the right hand and the less-dominant hand as the left, but the system is ambidextrous, because the Polhemus trackers are symmetric and can be handled with equal ease by both hands.

This paper describes the CAD system, introduces the operations and the interaction techniques, shows a simple example surface, and points to directions of future research.

2 Previous Work

The previous non-traditional CAD systems that use 3D input devices can be subdivided into two broad categories – immersive and non-immersive.

2.1 Immersive 3D Design

A Head Mounted Display (HMD) is used in the immersive style to display the scene to the user. In 1976, James Clark [9] build a system which used a HMD and a single 3D *wand* with some buttons to design free-form bicubic patches rendered as line drawings. Control point manipulation was accomplished by moving the wand within the selection radius of the desired control point, pressing one of the wand buttons, and dragging the control point to the desired position. Clark's system required exotic hardware, so neither the system nor the interaction style were successful at the time.

More recent work at the University of North Carolina and Chapel Hill resulted in 3DM [5], an immersive system in which the user manipulates a Polhemus tracker with two buttons mounted inside a billiard ball. The system was loosely based on MacDraw, where the user picks operations and geometric primitives from a rectangular 3D menu in space. The user lays out triangles by clicking a button for each of the control points, and creates boxes, cylinders and the like by dragging out a bounding box. A limited number of operators can be applied to each object, such as scale, rotate, translate and change color.

Stoakley, Conway and Pausch developed a two-handed architectural design system called WIM (Worlds In Miniature) [26]. The user is immersed in the scene at full size, and the user also sees a miniature hand-held copy of the scene (the WIM) attached to a tracker manipulated by the left hand. A lightweight clipboard is attached to the left tracker, and the surface of the clipboard represents the floor of the WIM. The right hand holds a ball containing another Polhemus tracker and two buttons, one for selecting objects, the other for moving them. The WIM provides both an aerial perspective on the entire scene, and allows the user to manipulate objects in the miniature version of the scene.

The nominal benefit of the immersive style is that the HMD gives the user a panoramic view of the scene, which enhances the user's understanding of the scene and allows for much more virtual screen area than is available in non-immersive systems. Putative benefits notwithstanding, the chief drawback of the immersive style is that it cuts the user off from the traditional I/O devices such as the screen, keyboard, mouse, telephone, coffee mug, reference documentation and most importantly, co-workers. See-through HMDs [22] at least offer the possibility of regained access to the real world, but the technology is not yet mature [1].

Another drawback is that to take advantage of the large virtual screen space offered by HMD, the user must stand up. Thus, virtual world interactions are done with the arms outstretched, which soon results in sore arm muscles. By contrast, the non-immersive style does not suffer either of these drawbacks, because

the user sits at a graphics console in the traditional manner. The user can see everything and rest his tired bones on the desk or the arms of the chair.

2.2 Deskside 3D Design

In the non-immersive style, Chris Schmandt built an experimental system at MIT [24]. This system used shutter glasses to provide a binocular stereoscopic display for the user, and a half-silvered mirror which overlaid the graphics display on the real world scene. A Polhemus-based wand was used to trace out curves.

Emanuel Sachs [23] and his colleagues at MIT built a system called 3-Draw, which was a two-handed system for sketching 3D drawings. The left hand held a lightweight clipboard with a Polhemus tracker attached, and the right hand held a Polhemus 1-button stylus. The left hand held the base coordinate frame of the model, so as the left hand moved, the model moved in synchrony on the screen. The right hand directly selected the data items to be manipulated, and selected the operations to be performed using a screen-aligned 2D menu controlled by the 3D tracker. The fundamental data items were polylines and points, which were created by sweeping out the curve to be drawn with the right hand. The stylus was also used to pick and directly manipulate points and polylines.

The simultaneous use of two sensors takes advantage of people's innate proprioceptive knowledge of where the two hands are in space. Guiard [13] gives psychophysical evidence for the idea that the left and right hands quite often act as elements in a kinematic chain. For right-handed people, the left hand acts as the base link of the chain. The right hand's motions are based on this link, and the right hand finds its spatial references in the results of motion of the left hand. Also, the right and left hands are involved in asymmetric temporal-spatial scales of motion (right hand for high frequency, left hand for low frequency). [2] 3-Draw uses this natural division of manual labour by assigning the (low-frequency) setting of spatial context to the left hand, and the (high-frequency) selection and picking operations to the right.

Jiandong Liang at University of Alberta built a system called JDCAD [20, 21] which uses a single Polhemus sensor to create 3D mechanical parts. The user creates items by selecting a primitive from a Polhemus-based ring-shaped menu and dragging out the bounding box. The user can then select objects, reshape them either freely or along a constrained axis, and group them together. JDCAD provides a number of editing and alignment operators such as center-to-center alignment, center-to-edge, and so on. JDCAD also has CSG capability, allowing the user to create complex objects by performing union, intersection or subtraction between two objects.

JDCAD is admirable for creating quick sketches of mechanical parts, and for things that look like mechanical parts, but it is not very good for creating natural objects. This is mainly because the geometric vocabulary of canonical solids is not suitable for free-form object creation. However, the ease with which

[2] For left-handed people, the roles of right and left are reversed.

one can create objects in JDCAD is gratifying enough to attempt using the non-immersive interaction style for other design tasks.

2.3 Two-Handed Input

Computer systems which give both hands a spatial input device are gaining popularity, based on the idea that users can use both hands in parallel to quickly achieve a complex task. However, few models exist to structure the design of two-handed interactions, since there are so many ways that two hands can be made to work together.

Buxton and Myers showed that users naturally used both hands simultaneously to position and scale a square to match a target [6]. In this experiment, the right hand used a tablet to position the square, and the left hand used a slider to scale the square symmetrically. Without prompting, and despite encouragement to perform the task sequentially, all but one subject used both hands in parallel. Evidence from a second experiment indicated that one of the benefits of two-handed input is that each hand stays in its "home position". However, as Kabbash, Buxton and Sellen point out [17], there is some danger in designing a two-handed task that imposes a high cognitive load on the user, wiping out the "home position" advantage. They performed an experiment comparing *Toolglass* [2] to two-handed tear-off menus in a task to connect dots with a colored line segment. For each new dot that appears on the screen, the user must choose the new color, and pick the new dot. They showed that Toolglass performed much better than the tear-off menu because the Toolglass presented a unified manipulation in a way that matches Guiard's [13] analysis of bimanual action.

Hinckley et al. [14] show a similar result with their two-handed 3D neurosurgical visualization system. Like 3-Draw, the left hand holds the brain model represented by a ball, and the right hand manipulates a cutting plane represented by a flat plate connected to a second Polhemus tracker. The user moves the cutting plane to interactively generate cutaway views of the brain. New users immediately understand how the manipulation works, and need essentially no training to start doing useful work.

Therefore, THRED is based on the strategy of splitting the effort according to Guiard's observations:

- The right hand operates in the frame of reference set up by the left hand.
- The right hand operates after the left hand.
- The right hand performs finer manipulations than the left hand.

Our strategy can be summarized by saying that the left hand provides context, and the right hand does fine manipulation.

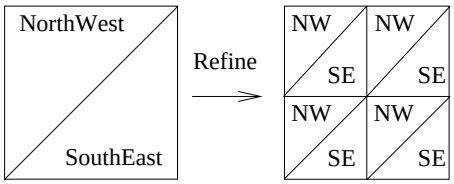

Figure 1: A single square and its refinement into four sub-squares.

3 Model Structure

THRED is based on an SGI Crimson RealityEngine graphics workstation and a pair of Polhemus Isotraks with buttons (Bats) [28], which are the main input devices. Each hand holds one bat, and the user manipulates the graphical scene with them.

THRED allows the user to create hierarchically-refined polygonal surfaces based on quadrilaterals. The hierarchical refinement allows areas of low surface variation to be represented by a few large rectangles, while areas of fine detail have the appropriate concentration of closely-spaced vertices. Also, broad-scale geometric operations can be performed on a few parent rectangles and then propagated appropriately to the children of these rectangles [12]. These broad-scale edits will leave the fine surface details of the child rectangles mostly intact, except with minor distortions induced by surface stretch.

At the root level of refinement, the surface is simply a rectangular grid, with each vertex connected to its North, South, East and West neighbors. To polygonize this grid, each rectangle is subdivided into a NorthWest and a SouthEast triangle, allowing a direct edge connection from a vertex to its NorthEast and SouthWest neighbors. Each rectangle can be subdivided into exactly 4 sub-rectangles, resulting in 8 corresponding triangles.

The data structure of THRED is based on the *square* (figure 1), which contains one vertex and a set of pointers to all its nearest neighbors in the mesh. The square also contains a list of its children, surface normals, plane equations, and graphical display list IDs. Aside from bookkeeping modifications, only the following three operations can modify the mesh data. The *refine* operation creates four child squares for a square, the *move* operation changes the vertex data and recalculates other salient geometric information, and the *add row/column* operation adds a row or a column of squares to the periphery of the root-level mesh.

To create a model, the user starts with a single square at the origin and commences adding rows and columns, refining squares, and moving vertices. The user builds the desired surface by iteratively modifying, refining, and examining the surface given the available interface tools. This view of surface modeling is similar to Forsey's Dragon Editor[12], with the obvious exception that the Dragon Editor manipulates bicubic B-Spline surfaces.

3.1 Surface Display

In order to help the user visually locate the control vertices, the NorthWest triangle of a square is drawn a darker shade than the SouthEast triangle. Each level has its own unique hue, so for example the root level is magenta, level 1 is yellow, and so on. For diffuse and specular reflection, the face normal is used, so that the surface looks strongly faceted. This helps the user find each editable vertex at a glance.

When the user has restricted vertex selection to a maximum level of refinement, descendants of the maximum squares are drawn using the maximum level color, but the geometry of the descendant vertices. That is, when the level is restricted, just the color changes to reflect the restriction and to delineate the selectable points.

One of the problems in working with free-form surfaces is that it is often difficult to immediately understand their shape. This is a 3D perception problem, and the standard depth cues [4], of hidden surface removal, surface lighting, and perspective projection are of course employed. THRED adds continuous motion parallax and surface texture. We do not currently use stereopsis in THRED [10, 27], although this could be added, and we do not use aerial perspective (fog or depth cuing) because fog is most useful for distant objects. The continuous motion parallax is controlled by grabbing the model with the left bat and moving it around with the left hand.

Two types of textures are mapped onto the surface of the triangles, a random texture and a contour texture.

The random texture is 64 by 64 pixels, with just an intensity channel. Each pixel in the texture has a random brightness value between 0.8 and 1.0, which is used to modulate the brightness of the underlying polygon color after all the surface lighting and shading has been performed. The texture mapping process maps the repeating random pattern onto each triangle like wallpaper. We tried using rectangular and hexagonal grid patterns for the surface texture but found that they added visual clutter in the near field of view and created moire patterns at greater distances.

We have found that mapping the random texture to the surface helps the user to immediately understand the surface shape. In situations where it is ambiguous whether a surface feature is a depression or an outcrop, the surface texture combined with the lighting model usually resolves the problem. This is because the user can use natural perceptual knowledge of texture gradient to understand the surface shape. Random texture also gives the interior area of a triangle a visual feature that can be followed as the surface is being reshaped, so texturing also helps editing.

The contour texture produces a series of equal-height contour lines on the surface, which helps the user gauge the height of features and to get a feel for the relative sizes of features. The texture is a one-dimensional line of pixels with just an intensity channel, and this line of pixels repeats to form the contour over the entire surface. This texture presents a ruler pattern of meters and tenths of meters on the surface. The tick

mark at zero meters uses 4 dark pixels (zero intensity), the 0.5 meter mark uses two dark pixels, and the other 1/10 marks use one dark pixel. The remaining pixels are at full intensity, and the mapping function calculates the distance of the surface vertices from the XY plane modulo the length of the pixel row. The planar interpolation of each polygon handles the rest of the texture lookup procedure. This planar distance function is supplied by SGI's Graphics Library, so we just need to specify the equations.

To get both textures on the surface simultaneously, we render the scene twice, first using the random texture with the Z-Buffer hidden surface comparison function set to *draw-if-closer*. Then we change the Z-buffer function to *draw-if-equal*, and draw again using the contour texture and a translucent color.

3.2 Modeling Interface

When THRED starts up, the user is presented with a window containing two subwindows and the top menu bar, as shown in figure 10. There is only one pull-down menu in the menu bar, containing traditional file operations such as load, save, and quit.

Below the menu bar is the refinement level window, which displays all of the levels of refinement that have been made to the surface, and shows which levels are currently selectable. The refinement window is only 100 pixels tall, and is used for quick reference and for quick interactions to change the current level of refinement. This window also shows the current reshape operator on the left.

The majority of the screen space is taken up by the main window, which displays the work area and the left and right cursors. The two cursors are both transformed from a common room coordinate system [25], so that cursor separation on the screen is the same as the separation of the bats in real space, and the cursors are located in approximately the right place on the screen. A re-origining operation moves both cursors by the same amount so that the midpoint of the line segment between them is at the center of the screen space. The re-origining operation helps the user to find the "sweet spot" where manipulations are the most comfortable.

The right cursor is drawn as a 3D jack, and the left is a jack with a flat plate attached, reminiscent of a clipboard. Once the user understands which cursor belongs to which hand, cursor confusion rarely arises because the user relies on proprioception to find his/her hands and therefore their surrogates in the design space.

4 Right Hand Operations

The right hand has three major tasks to perform:

- Control point selection and selection adjustment.

- Reshaping the selected area.
- Reshaping the selected area with orientation.

4.1 What Control Points Can Be Chosen?

To reshape the surface, the user must first select which control points or squares are to be changed. This section describes the syntax of control point selection, and the next section describes the geometry of control point and object selection using the right bat.

The user may select one control point, or a rectangular collection of points. To select one point, right button 1 is pressed and released on the desired control point. To select a rectangle of squares, the user presses and holds button 1 at any control point, drags the right cursor to the opposite corner, and releases button 1. The rectangle may be degenerate: a single row or column of points is also valid. The currently-selected rectangle of squares is outlined by yellow piping along its periphery. The selection rectangle can be adjusted by pressing button 1 while the hot spot is on the yellow piping and dragging the edge or corner to another control point in the usual manner of resizing 2D rectangles.

Selection depends on the current selection context, so for example if the user has restricted selections to level 3 or less, then level 4 and deeper levels will not be selectable. Also, once the user selects a control point of a given level, the selection rectangle cannot include higher or lower level control points. This is accomplished by following the periphery of the current selection rectangle from the starting point to the current point, and halting progress along the X or Y direction when a different level is encountered.

The selected area may be refined or reshaped. In cases where the selected area is on the border with a square of shallower refinement, then this edge of the selection area is only refinable, not reshapable. Without this restriction, the user could grab a vertex that lies on the midpoint of a less-refined edge. When this vertex is moved, a crack in the surface could appear as shown in figure 2 because one edge between two vertices is no longer coincident with the more deeply refined edge of three or more vertices.

In most cases, a smaller rectangle inside the selected rectangle is reshapable, because the edge-to-edge vertex matching restriction is met. This smaller region is outlined in red piping. If there is no red piping then the entire yellow-framed region is both resizable and reshapable.

The presence of uneditable vertices is the obvious disadvantage of this hierarchy of rectangles. The advantage of this scheme is that there is a naturally-preserved hierarchy of fine geometry that can ride on the back of coarse squares, so that global edits can be performed by moving a few coarsely-refined vertices.

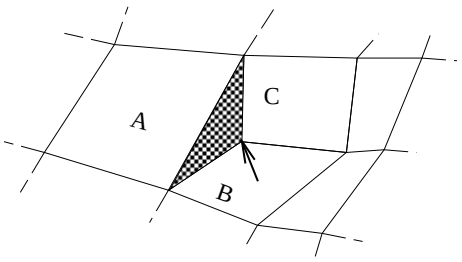

Figure 2: A crack (the shaded triangle) appears between squares A, B and C because the marked vertex joining B and C should be on A's edge. THRED avoids this condition by disallowing edits of these vertices.

4.2 Geometry of Control Point Selection

Given that the user desires a control point, this section describes how the user geometrically controls the right bat to pick the desired point.

We use a probe selection technique similar to Liang's [20, 21] *spotlight*. A probe, represented by a narrow cylindrical shaft, is attached to the right cursor, and the user controls the position and orientation of the probe with the right bat. The distance from this probe is computed for each control point using a specialized distance metric called the *probe metric*, and the point that generates the smallest value is the current *hot spot*.

The probe has two visual presentations; one for selecting objects, and the other for selecting a single point on the surface of an object. For surface selection, we need an unambiguous and easily-understood cursor point. To achieve this, the probe metric is evaluated for all vertices, and those vertices with large metric values are eliminated from further consideration. Next, all surfaces adjacent to the remaining small-metric vertices are intersected with the probe axis. The intersection that is at the minimum Euclidean distance from the right cursor is the chosen surface point. This intersection point is highlighted by drawing a small sphere, and we draw the probe axis from the right cursor to the intersection point.

The hot control point is the vertex closest (in Euclidean distance) to the intersection point on the intersection triangle. The hot spot is highlighted by drawing an arrow from the intersection point to the hot spot. This highlight arrow lies in the surface of the triangle where the intersection occurs. The right image of figure 4 shows the probe and the hot spot arrow.

If no objects intersect the probe axis, the probe visually switches to object selection mode. To select objects, we draw the probe axis to some arbitrary length, and we draw a translucent cone which has a radial spread angle of 10 degrees. The probe is the cone axis, as shown in the left image of figure 4. We attach to the cursor a spotlight that has identical visual geometry to the translucent cone. As the user sweeps the probe about the scene, the objects that fall within the cone intersect visually with the cone wall, and are

highlighted by the spotlight. Since we are interested in selecting vertices, we draw an arrow from the right cursor to the nearest (using the probe metric) control point. When there is an intersection between the probe axis and a surface, we turn off the translucent cone and the spotlight, because both tend to obscure surface detail.

The non-Euclidean probe metric is designed to closely mimic the shape and behavior of the probe axis. That is, the closer a point is to the axis, the more likely the point is to be picked. The probe metric also allows the selection of objects that are small and/or far away. Simply tracing a mathematical ray along the probe axis and picking the first intersecting object is not effective, because a high degree of cursor accuracy is required to pick small objects.

Given a point $P = (x, y, z)$, and given that α (measured in degrees) is the angle between the probe axis and P, then the probe metric from P to the origin is given by

$$ProbeMetric = |P|(1 + \alpha)$$

where $|P| = \sqrt{x^2 + y^2 + z^2}$ is the usual Euclidean distance.

To compute the probe metric from a cursor located at point C, with the probe axis directed along the normalized vector R, we need to generate V, the vector from the cursor to P:

$$V = P - C \qquad V_n = \frac{V}{|V|}$$

Since R and V_n are unit length,

$$\alpha = \cos^{-1}(R \cdot V_n)$$

and

$$ProbeMetric = |V|(1 + \frac{180 \cos^{-1}(R \cdot V_n)}{\pi})$$

The conversion from radians to degrees serves to sharpen the outline of the isodistance curve, as shown in figure 3. Figure 3 also shows the isodistance curve for the standard Euclidean metric $|V|$, and for Liang's ellipsoidal metric[3]:

$$K = \sin(cone\ angle)$$

$$EllipsoidalMetric = |V|\frac{\sqrt{1 + (K^2 - 1) * (R \cdot V_n)^2}}{K}$$

There are some important differences between Liang's spotlight technique and our probe technique. The spotlight is only useful for selecting objects. It cannot be used without modification to select a point on the surface of an object, because of the ellipsoidal distance metric. This metric is rather difficult to control because when the probe axis is moderately close to parallel to the surface (less than 30 degrees), the roundness of the ellipsoid near the probe axis evaluates off-axis points on the surface to be closer than on-axis points.

[3] There is an error on page 502 of Liang's paper [20]. The correct formula is given here.

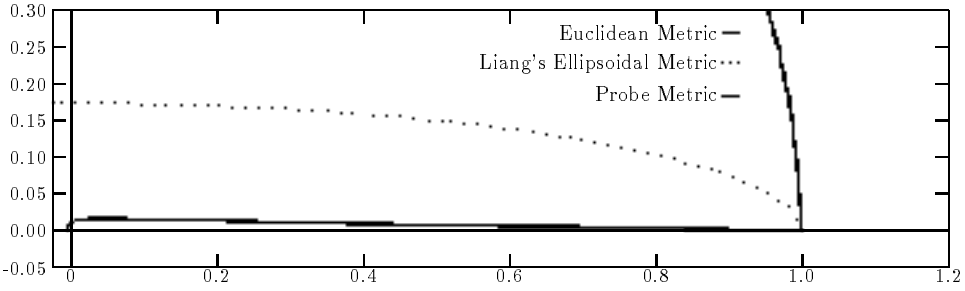

Figure 3: This plot shows the isodistance curves for the Euclidean distance (a circle), Liang's Ellipsoidal metric, and the Probe metric. The cursor is at the origin and extends along the horizontal axis. This plot is to scale.

Also, the ellipsoidal metric must be cut off by the surface of the cone. If it is not cut off, the ellipsoidal metric selects surfaces that are (Euclidean) close to the cursor but which lie outside the cone. In sum, the ellipsoidal metric does not usefully approximate the shape of the probe axis unless the probe axis is close to perpendicular to the surface.

Our probe metric does approximate the imagined needle-like shape of the probe axis, and it has the useful property of being equal to the Euclidean metric when evaluated on-axis. In fact, when the surface intersection test is turned off, the probe metric usually picks the vertex closest to the surface intersection point unless the probe axis is almost parallel to the surface (within 2 degrees).

A disadvantage with the probe metric is that in situations where the probe is perpendicular to the middle of a triangle, and is close to its surface, any vertices behind it that are close to the probe axis will give small probe metric values. Unless every surface is intersected with the probe axis, or an intersection-tracking scheme is used, a vertex that is close to the probe axis will end up being chosen as the hot spot. In this case, the user must point the probe at the desired vertex.

4.2.1 Volume Cursors

Kabbash and Buxton [16] make the salient point that picking a point with an area or volume cursor is easier than with a point cursor. An example of this is Zhai's [29] *silk cursor*, in which a translucent box is used to surround objects in the scene. However, because the user cannot conveniently control its size, the silk cursor is only good for selecting widely separated points or objects.

Liang's spotlight cone does not have this problem because the cone has a linearly increasing cross-section, and the user controls its position and orientation. To pick one of two small adjacent objects, the user can move the cone close to the desired object (reducing its cross-section), and reorient the cone so that the ellipsoidal metric unambiguously picks one object. However, as previously discussed, the ellipsoidal metric must be cut off by the cone, or it will give false positives at nearby points outside the cone.

The probe metric solves the false positive problem by using an isodistance shape that closely approximates a line segment. Unlike the Ellipsoidal metric, the probe metric is unidirectional, extending only slightly "behind" the cursor. Thus, while the probe may be thought of as a volume cursor, the shape of the probe metric allows us to dispense with an explicit point-in-volume test. The metric allows us to generate a partial order of all the vertices in the scene. Vertices below some threshold are used as the most likely to generate a surface intersection.

We had previously used a scheme similar to Bier's [3], in which a line is constructed from the eyepoint through the cursor into the scene. A cone whose axis is this line and which has a 10 degree apex spread was used to initially reject control points. All points that lie within the cone were then sorted by depth, and the closest control point is the current hot spot for the cursor.

This scheme is functional, but it has the drawback that the user can only do selection by translating the right cursor. To select widely-separated points, the user's arm must therefore translate a fairly large distance. With the probe, the user can simply reorient the right bat with the fingertips to select widely-separated points. The probe also allows the user to select objects and surface features that are hidden behind other surfaces, which is impossible with the old scheme.

4.3 Probe Orientation

All of the previous section assumes that the probe axis is aligned to some arbitrary axis of the right bat. Mathematically, it makes no difference what vector relative to the right bat's coordinate system is used as the probe axis, but from a human factors standpoint, it matters a great deal. Liang used the X axis of the bat (see figure 8) as the spotlight cone axis because this is the axis along which the wire connects to the Polhemus sensor.

When we attached buttons to the Polhemus sensor, we found that it was difficult to comfortably point the probe at the target while pressing a button. In particular, vertices on the lower half of the screen were hard to pick because the natural posture [4] for the right bat is with the X axis pointing straight up (figure 8). For a while, we experimented with using the Z axis because in the usual posture, the Z axis of the right bat points towards the screen. The Z axis gives the right hand more equal access to all of the vertices because it is in a more central orientation.

The Z axis is still not ideal, so we tried to determine a vector that would closely match the line that a pen would take in the standard grip. The pen concept was based on the idea that since people are used to drawing with a pen, the usual orientation of the pen in hand would be easily remembered. Unfortunately, this is not the case, and the salient issue here isn't spatial memory but spatial accessibility.

The ideal probe orientation is perpendicular to the screen when the hand is in a relaxed pose. This

[4] See section 6 for a discussion of this issue.

orientation can be generated by simply tracing a ray from the virtual eyepoint through the cursor when the user is in a relaxed pose. The line is transformed into the coordinate system of the right bat, and is used as the probe axis until the next re-orient command is issued by the user. We have found that this reduces the amount of fatigue that users experience, since users do not have to twist their right arms into funny shapes in order to reach moderately distant vertices.

Another benefit of this is that in comfortable orientations, the user can successfully draw on the surface, even on curved surfaces. With a little practice, the user can trace along a contour line on the surface. We have not studied this, but we think it may even be a little easier than tracing with a mouse.

4.4 Reshaping

When the user presses right button 2, the squares of the reshapable selected area can be reshaped according to the given reshaping context. Currently the five reshaping operations are *move everything in tandem*, *cone* reshape, *scale* reshape, and *scale to median plane*, and *adopt parent*. Reshaping uses only the right cursor's position data.

The tandem move operation is useful for grabbing a block of squares and moving them all at once. It has limited expressive power all by itself, however, because this reshape mode simply devolves to moving one vertex at a time.

When moving perpendicular to the plane, the cone reshape operator acts as if a conical tent were being erected centered at the middle vertex. For a vertex in the selected area, this is implemented by multiplying the cursor motion by a linear factor of the distance from the center vertex. The linear factor is determined such that the most distant selected vertex from the center does not change, and such that the center vertex moves in tandem with the right cursor.

The center vertex is determined by taking the middle square within the selected rectangular array, and if there is a tie, taking the North, East, or NorthEast square as appropriate. Taking a vertex as the center of manipulation is necessary because it yields a clear peak when reshape is performed, and because the operation is undoable by moving the center of manipulation back to its original position. Using the arithmetic mean of all points or just the center of the axis-aligned bounding box is not reversible, because the center point changes relative to the vertices.

Movements in the plane have an analogous cone-like effect to movements perpendicular to the plane, and the result is to change the spacing of vertices on the plane.

The scale operation scales the distance of the non-center vertices from the center vertex. This operator can be used to exaggerate or minimize the spikiness of the selected features.

A similar operation is scale to median plane, in which the median plane of all the selected vertices is calculated, and the reshape operator scales the distance of each vertex from this plane. This is essentially a

1 dimensional operator, and this is used to scale in a space that represents a major feature of the data.

The adopt parent operation causes the selected squares to adopt the geometry of their parent squares. The selected squares realign into groups of four on their parent squares as if the parents had just been refined. The purpose of this operator is to reduce the amount of high-frequency surface detail, and is used as a "start over" operator. This is not under continuous control, and takes place as soon as the user presses right button 2.

4.5 Reshaping with Orientation

When the user presses right button 3, only the tandem reshape mode is active, and the user can directly manipulate both the position and orientation of the selected vertices in tandem. This operation is useful if the user wishes to maintain the shape of a region, but wants to make it steeper or shallower with respect to the rest of the surface.

5 Left Hand Operations

The left hand has four tasks to perform, all related to context setting in some way:

- Set the current constraint mode.
- Set the current selectable refinement level.
- Set the current reshape shape.
- Manipulate the position and orientation of the entire scene.

Scene orientation is a toggle controlled by left button 3. Pressing button 3 attaches the workpiece to the left cursor, and pressing it again leaves the workpiece in place. This clutching mechanism does not use continuous button depression because the user spends a large amount of time moving the workpiece around, and continuous button pressure would be a significant burden.

This mode is similar to 3-Draw [23] and WIM and related systems [14, 26], with the exception that in THRED, it can be turned off. We have found that moving model while doing control point editing adds a certain amount of inaccuracy, so users generally turn it off while refining the design.

5.1 Constraint Mode

When manipulating the control points, there are usually two phases: gross approximation followed by many steps of refinement. Initially the user wants to quickly place control points in approximately the right place, and then at some later point the user will refine these points to greater and greater precision.

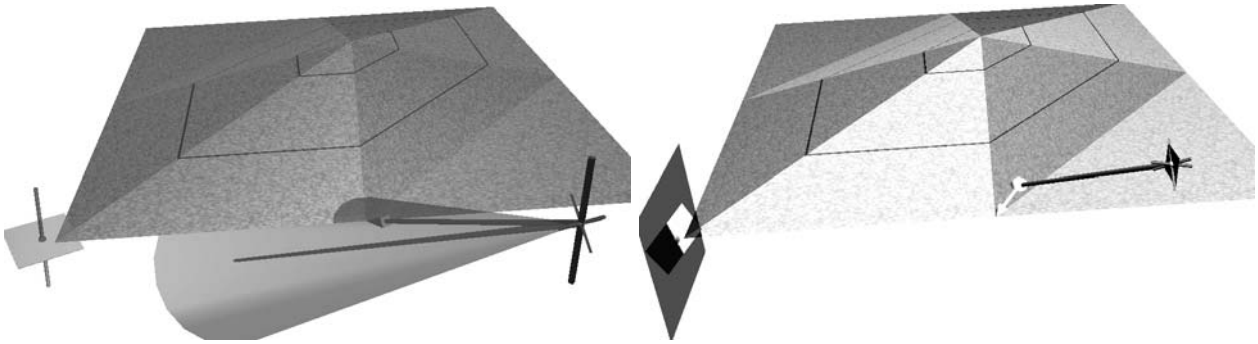

Figure 4: Left: Line constraint mode is active. Selected vertices can only move vertically. The right cursor shows the selection cone with its central probe axis, and an arrow pointing from the cursor to the nearest vertex.
Right: Plane constraint mode is active. Selected vertices may move in the horizontal plane shown by the left cursor. The probe on the right cursor intersects the surface, and the hot spot is indicated by a small arrow from the intersection point.

Users also generally find precise placement of objects much easier to control if they are controlling only one degree of freedom at a time. Users also do not want to wreck previous refinements by accidental movements caused by refinements in another degree of freedom. Liang [20, 21] points out that a substantial fraction of the operations that users perform in JDCAD are axis-constrained operations, and our experience in using JDCAD bears this out.

Therefore, both constrained and unconstrained operations are available in THRED, with the left hand providing both the type of constraint and the geometry of the constraint. Clicking left button 2 moves from the current constraint mode to the next in a three-step cycle of *no* constraints, *line* constraints, and *plane* constraints.

Initially, the right cursor can manipulate vertices with all 3 positional degrees of freedom. When left button 2 is clicked, the system moves into line-constraint mode, in which right-handed reshaping operations are constrained along one of the three major axes. Both cursors change their shape to reflect line constraint mode, with the left cursor changing to a short pole through a square, and the right cursor adding a short pole that snaps to the current constraint axis (figure 4). The left cursor does not snap but remains tied to the left bat, because the canonical axis closest to the bat's Z vector (the pole axis in the left cursor) determines the constraint axis.

Thus while the left bat is geometrically stating the constraint axis, the right axis can be making a selection and starting the reshape operation. In order to avoid any inadvertent changes in snap axis, the constraint axis remains fixed while the reshape operation takes place, so the left hand does not need to hold a steady

orientation during reshaping.

Similarly, in plane constraint mode the canonical plane of motion is stated continuously and directly by the orientation of the left hand, and remains constant while the right hand is doing a reshape. In this case, the right cursor shows a small square that is snapped to the constraint plane, and the left cursor shows a much larger snapped plane as shown in figure 4. The left cursor also has a small square rotating freely with the left bat, to help the user get to the desired orientation.

5.2 Selecting Refinement Level

All of the operations discussed above take place when the operating cursor is within the main window. However, the left cursor can select the current refinement level when it is moved above the top edge of the main window into the refinement level window.

The geometric space of these two windows is contiguous, and is kept consistent between window resizes so that when the left cursor moves above the main window's top edge, it appears in the same place in the refinement level window.

Each level is represented by a square triangulated in the same NorthWest-SouthEast pattern as the edited surface, and is drawn using the same level-based colors. The squares are screen-aligned and are arranged from left to right joined edge to edge. The number of levels evenly subdivide the window horizontally, so if there are 3 levels, the level 0 rectangle takes the left third, the level 1 rectangle takes the middle third, and the level 2 rectangle takes the right third. The currently selected levels are outlined with yellow piping.

When the left cursor is moved into the refinement level window, it is projected onto the surface of the refinement squares, and converted into a 1D slider. To find which level is being intersected by the left cursor, just the horizontal position of the cursor is used and range checks are performed for the boundary between each square. Thus, precise location of the left cursor within the small window is not necessary, because the user just has to move into the refinement window. This low-precision selection mechanism helps in enlarging the targets so that they can be hit quickly [11].

The cursor is an arrowhead pointing over the current location located just in front of the squares, and to select the level under the cursor, the user just clicks the left button 2 once. To select more than one level, the user holds down button 2 at one end of the selected set and drags the cursor to the other end. Double clicking the mouse button toggles between the current selection and selecting everything, so the user can quickly step in and out of select-everything mode with a double click in the level selection window.

5.3 Selecting Reshape Operators: The Sundial Menu

Since the user holds a bat in each hand, it is inconvenient to drop the right bat in order to perform menu selection with the mouse. We therefore need a means of selecting among a group of choices using the bat.

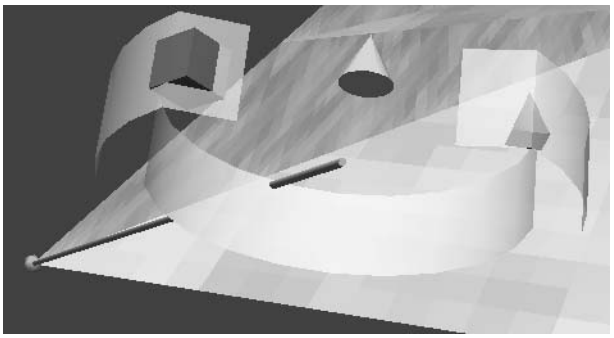

Figure 5: The ring menu. Cone reshaping is currently selected. Tandem move is half behind the translucent band on the left, and scale, scale to median plane and adopt parent are on the right around the circumference.

Our first attempt at this was to use Liang's Ring menu, (figure 5), which is a popup menu controlled by the orientation of the left bat. The menu items are placed on the circumference of a circle at a fixed orientation with respect to the bat. The translucent belts serve to visually disambiguate the items in the front half of the ring from the back items, and the gap in the front belt supplies a selection zone. The belts are maintained on the Y-Z plane of the bat with the selection gap always pointed at the user, so as the user rotates the bat about the X axis, the item to be selected appears within the gap and is highlighted.

There are five main difficulties with the ring menu:

- Using a bat button to pop up the menu limits the range of motion to about 3/4 of a revolution. The ring menu was designed to work without buttons, so that the user could twirl the Polhemus sensor about the X axis using its wire (see figure 8). When a bat button is used, the user must rigidly grip the bat through the entire menu selection process, which means that the user cannot extend the bat's orientation envelope by unclutching and clutching it. While the button is pressed, the bat has the same orientation envelope as the wrist.

- The translucent belts tend to obscure the items in the menu. Users typically reorient the Y-Z plane of the ring so that they can examine what choices are available.

- Users must sequentially pass through items in order to reach the desired item. Standard rectangular menus have this difficulty also.

- The ring menu is not conveniently hierarchical. A child menu cannot simply pop up when its parent pointer is highlighted, because the user may be on the way to another item. Some spatial means must be chosen to indicate that a child is desired, but none is readily at hand. Users typically use the two remaining degrees of orientational freedom to examine what choices are available, and they reposition

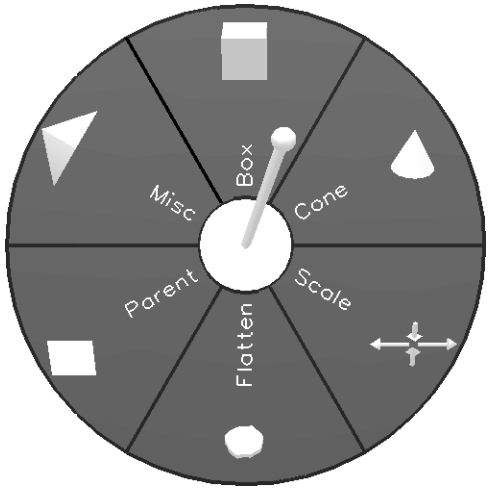

Figure 6: The sundial menu. Tandem reshaping is currently selected. Clockwise from tandem is cone, scale, scale to median plane, adopt parent, and a child menu.

the menu so that they can see the items more clearly. For this reason, the ring menu is limited to about 20 items.

- There is no way to cancel menu selection. Once the menu has popped up, the user must choose one of the options. This could be fixed by allocating a blank area on the ring as a no-op item, but this isn't really satisfactory.

To avoid these problems, we designed the sundial menu (figure 6), which is a popup menu controlled by the orientation of the left bat. The menu choices are arrayed on the circular plate of the sundial, each on its own pie-shaped sector. The desired item is picked by pivoting the *shadow stick* about its base so that the stick's endpoint lies visually in front of the sector. The base of the shadow stick is located at the center of the plate, and when the menu first pops up, the stick is aligned with the line of sight, pointing directly at the user. Around the base of the stick is an inner circle which the user can point the stick at to indicate *no selection*. The position of the sundial is fixed rigidly to the left bat, and the dial lies parallel to the projection plane.

This is a 3D version of the pie menu [7] which encodes the position of the 2D cursor by projecting the 3D position of the endpoint of the shadow stick onto the 2D sundial plate. To implement this, we trace a ray from the eyepoint through the endpoint of the stick, and intersect it with the sundial plate. Under parallel projection, this can be simplified by simply taking the X and Y components of the endpoint's offset from the dial center, but this results in visual inaccuracy under perspective projection. Similarly, the realignment of the shadow stick along the line of sight at popup time is needed so that the stick starts at a neutral orientation. Realignment also has the advantage of allowing the user to have the bat at any orientation

when the menu pops up.

We set the length of the shadow stick equal to 1.2 times the radius of the dial, which allows the user to cast a shadow on the outer circumference of the dial with a deflection from center of 56 degrees. A deflection between 56 and 90 degrees is of course still valid, since the nearest sector is highlighted. A longer stick allows smaller deflection angles to reach the circumference, but makes the central *no selection* circle slightly harder to hit. The advantage of a longer shadow stick is that it feels more responsive when ballistic rotations are made, and it reduces the amount of rotation necessary to move the endpoint into the desired item. Currently, the inner circle is 0.2 times the radius of the dial, which means that a stick of length 1.2 must be deflected within 9.6 degrees of center to select nothing. We feel that this is a good trade-off between the fine control needed to move to a small target (the center) and the speed needed to move quickly to a large target. Controlling shadow stick size is not really a good means of manipulating C/D ratio, however. More direct means should be used, such as multiplying the rotational offset from center by some factor.

5.3.1 Hierarchical Sundial Menus

Hierarchical menus are supported using the sundial menu, as shown in figure 7. The purpose of the hierarchy is of course to allow for a larger collection of choices than can be conveniently and legibly placed on a single menu. As figure 6 shows, a child menu is represented by an outward-pointing arrowhead.

The user moves the shadow stick into the parent sector to pop up the child. As a visual reminder of where the parent is located, the child pops up with its center located over the arrowhead of the parent sector. The child is popped up only if the rotational velocity of the bat drops below a certain threshold while in the parent sector. This ensures that the shadow stick is not in motion when the child appears.

To pick a top-level item, users typically make a ballistic rotation towards the desired item and then release the button. Users expect similar behavior when a child menu is to pop up, and they also expect the shadow stick to appear stable in the child's center when the child appears. Without the velocity check, the child would pop up immediately on parent entry, and the shadow stick would continue to move as the user completes the parent-selection rotation. Because the shadow stick is realigned to the line of sight at child popup, continued shadow stick motion causes the user to inadvertently select one of the child items. The time it takes for the user to recognize that the child has popped up and to accordingly stop rotating the bat is enough to allow the shadow stick to turn another 10 to 20 degrees. This is exacerbated by the lag in the tracking system, which increases the time it takes to stop the shadow stick. The maximum velocity check also allows the user to quickly pass over a parent menu item without popping up its child.

To return to the parent when a child has been incorrectly entered, a user may simply move the shadow stick to the no-selection circle and release the button. We experimented with using an inner *backup zone* inside the no-selection circle, which pops down the child menu and recenters the shadow stick on its parent

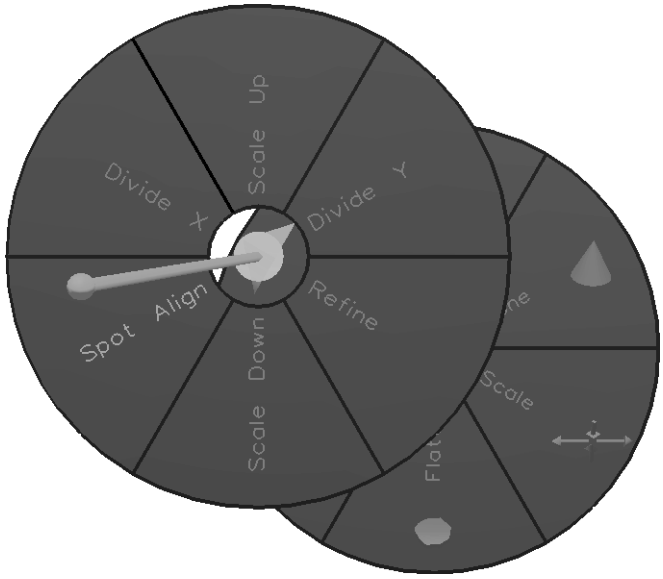

Figure 7: A hierarchical sundial menu. The child menu of the previous figure has been selected, and the probe orientation command described in section 4.3 is currently selected in the child menu.

when entered. The purpose of this zone is to satisfy the user's desire to avoid "starting from scratch" when they have accidentally selected the wrong child menu. The inner backup zone is not terribly effective, since it is hard to hit, and in conditions where there are three or more levels of menus, entering the backup zone of level 3 will cause a backup to level 2, and the recentering will cause a backup to level 1. Without recentering, the shadow stick reappears once again on the parent sector, and since the bat has low rotational velocity, the same child once again pops up.

More consistent with the idea of backing up to the parent menu is to use the volume of space behind the child as the backup zone. To get to this zone, the user turns the shadow stick so that it is behind the child menu (90 degrees of deflection or more), as if trying to touch the parent menu. Entering the zone immediately causes the child to pop down. We do not recenter the shadow stick because the user's hand is usually in an awkward orientation, so the user recenters by moving back to a comfortable orientation. This scheme has a similar problem of cascading backups if the user has turned the shadow stick too far behind, but this does not always happen. The advantage of this scheme is that backing up can be accomplished with a flick of the wrist, since the backup zone is large (the full hemisphere) and therefore easy to hit.

5.3.2 Sundial Menu Discussion

The sundial menu shares some of the advantages of pie menus [7], such as the ability to pick any item with a short cursor movement, and being structurally easy to remember. The short strokes are possible because

the sundial uses two degrees of freedom to select a menu item instead of just one. Also, the targets are comparatively large, and so are easy to hit [11]. As a result, pie menus give shorter selection times and lower error rates than linear menus [7].

The disadvantages of ordinary 2D pie menus are as follows:

- When horizontal text is used, pie menus get rather large because the text for each item must fit within a pie sector. The effect is most severe for the top and bottom pie slices [18]. This large size tends to obscure a large part of the screen.
- Diagonal text may be used to economize on menu size, but it is harder to read and hard to generate.
- Pie menus popped up near any edge of the screen must be warped to a more central location so that the user can see all the items. Hierarchical pie menus run into this problem at every child popup. Hierarchical linear menus only have edge collisions near the bottom or the right edges of the screen.
- There is an upper limit to the number of elements that can be put in a single pie menu. Linear menus with scroll arrows have a much higher limit to the number of elements available.
- Circular objects are not easy to generate in most window systems.

The sundial menu shares the pie menus' disadvantages of limited item count and harder-to-read diagonal text. Because we are using the vector-based Hershey fonts, text at any orientation is easy to generate. Diagonal text helps to reduce the size of the sundial menu, so that it does not obscure too much of the screen. However, obscuration is less of a problem in THRED than in 2D systems or in one-cursor systems for two reasons:

- The menu's position can be changed during the interaction by simply translating the left bat.
- The user is usually concentrating on what the *right* cursor is doing. The left cursor is usually on the left side of the screen, and does not perform tasks that require the user to move it into the central area.

If the menu is obscuring something important, the user can move it out of the way without affecting the menu selection. This operation is entirely natural, and does not require the user to issue a command. The menu does not occupy a fixed amount of screen space, because as the user moves the bat farther from the eyepoint, the menu appears smaller due to perspective. For the same reason, popping up the sundial near the screen edge is not a problem. The user can move the sundial away from the edge to expose the obscured items if necessary. This natural translation of the menu allows for the user to continuously customize the location and size of the menu without any cognitive effort, and without having to restart the menu selection at a more convenient location.

A unique feature of the sundial and ring menus is that the size of the menu target zones remain constant, regardless of their size on the display. That is, because they are 3D artifacts using 3D input, the manual effort required to hit the target is constant, while the visual target size depends on how close the menu is to the eyepoint. As users learn the menu layout, they can take advantage of this fact, and pop the menu up at locations that result in smaller visual sizes. Experienced users can further optimize sundial menu selection by performing the ballistic rotations without actually looking at the menu.

A similar no-look strategy is used with marking menus [18, 19], in which the experienced user just makes the 2D strokes to select a menu item. The stroke path is drawn on the screen, so as the user descends the menu hierarchy, a zigzag path appears. A pie menu pops up after a small delay if the user holds the mouse still, so beginners must wait a little to find the next menu. Although sundial menus pop up immediately, ballistic motion is still available. With marking menus, the user can "back up" the ink trail to undo an erroneous selection. This is similar to using the central region of the sundial as the backup region, which has the problems outlined above.

The sundial menu has none of the difficulties of the ring menu:

- The range of orientation change is small, typically 30 degrees.
- All items are visible at once.
- Any item can be reached directly, without passing through other sectors.
- The sundial is hierarchical.
- Menu selection can be canceled by selecting the inner no-pick region.

Although we have not tested this, we believe that the sundial offers selection speeds and error rates competitive with pie menus.

6 Device Ergonomics

The top of figure 8 shows one of our bats, which is a standard Polhemus sensor to which we have attached three Omron JP3101 mechanical keyswitches, which are momentary pushbuttons that have an audible and palpable click.

In the usual posture, the thumb can press button 3 (nearest the wire), and button 2 (the middle button) (fig 8 bottom far left). Shifting between button 3 and button 2 is a simple matter of rocking the thumb over the pivot of the button casings (fig 8 bottom middle left). The index finger presses button 1, which is therefore used for *primary* tasks. The sensor is supported in the fingers by the middle finger and/or the third finger, and the small finger is sometimes used to grip the cable.

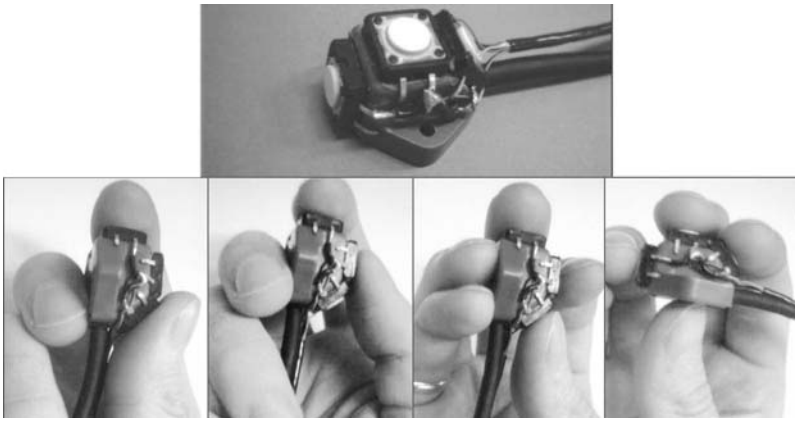

Figure 8: Top: The button-enhanced Bat. Bottom: Four postures for holding the bat. The X axis points up and the Z axis points to the left in the bottom left three pictures.

Two-button chords using button 1 and either button 2 or button 3 are easy to perform, since the thumb and the index finger each control one button. The middle finger supports the force applied by the forefinger and thumb, and these fingers both move along their prime directions of movement. A chord using buttons 2 and 3 is more difficult because either the thumb must press both buttons, or the index finger must press button 2 using abduction, or scissor-like movement. Three-button chords require a different posture, with the middle finger on button 1 and the third and small finger supporting (fig 8 bottom middle right).

This bat is an improvement on commercial products such as the Logitech 2D/6D mouse [15] because of its smaller size and weight. Because our bat is smaller, it can be manipulated by fingertips alone, while the Logitech mouse must be held aloft by the thumb and little finger, with the three middle fingers pressing the buttons. Thus, button presses pivot the mouse so that it is pressing on the palm of the hand, so in effect the entire hand is responsible for positioning and orienting the mouse. As a result, there is no precise coordination between the thumb and the index and middle fingers, which is vital if fine manipulation is to take place [8]. Also, the need to grip the mouse with the whole hand limits the range of orientations to that of the hand, while the fingertip grip allows the user to rotate the bat to a wide range of orientations (fig 8 bottom far right). Also, the lighter weight of the bat limits the amount of finger fatigue users experience.

As mentioned in section 4.3, the usual posture for the THRED user is to point the X axis of the bat up, as shown on the three bottom left images of figure 8. The bat's Z axis points to the left, towards the screen. This pose arises because the user's elbows are resting on the arms of a chair or on the desk, and the weight of the forearm is balanced on the elbow in a minimum-energy posture. Also, the weight of the cable is most easily managed when the cable is hanging straight down. The posture shown in the bottom right image of figure 8 is rare, because the weight of the wire makes it more difficult to hold the bat in this position.

Using a stylus instead of one of our bats for the right hand is an attractive possibility, since Polhemus

sells one, and the stylus might be used as a pen. We did not use a stylus for three reasons:

- The Polhemus stylus has only one button.
- The cable hangs off the end of the stylus, and therefore there is a tendency for the stylus to point straight up, to minimize the work needed to hold it.
- Because the stylus shape is mimiced by the probe, the probe therefore cannot be reoriented relative to the stylus as explained in section 4.3.

Thus, the stylus and the probe must always be parallel, or the whole point of visual mimicry is defeated. The stylus must therefore be perpendicular to the screen to put the probe in a centered orientation. A simple experiment with taping one of our bats to a pencil indicates that this orientation generates a certain amount of wrist fatigue, because the most relaxed orientation for the stylus is pointing up 45 degrees. Selecting items on the bottom half of the screen is a chore that requires either a large amount of wrist flexion, or that the user lift the right elbow off the arm of the chair. The ergonomic consequences of using a stylus reduce its appeal somewhat.

6.1 Device Fusion

Most commands are entered using the bats, but when the need arises, the user can quickly put down one or both bats and move the mouse or type at the keyboard. The mouse is only used to activate the pull down menu, and to move windows around and access other applications. The keyboard is used to type filenames, and enter numbers for vertex coordinates. Also, single keystrokes are sometimes used to invoke commands that are in the prototype stage. However, unlike text entry, pressing a single key does not require that the user actually drop a bat. Instead, the user can press a single key with an outstretched finger while still holding on to the bat. Typically the hand that is not performing a spatial operation at the time of the command is used to hit the key.

As might be expected, the 6 bat buttons are prime real estate, so commands which are executed infrequently such as *scale scene* are put in the left sundial menu, as shown in figure 7.

The bats are small, so putting them down does not require that the user find large amount of desk space to put them. By contrast, the left clipboard in 3-Draw [23] requires about the area of a steno pad to lay on the table, which means that this amount of area must be free when the left hand needs a rest. Similarly, 3D mice require a mouse-size area to put them when they are not in use, and in the THRED prototype, there are would be three mice beside the console, which is confusing.

There is some potential for confusion between which is the left bat and which is the right, and while visual labels are effective, we also find that a tactile difference between the two bats can be quite good for

User Photograph Goes Here

Figure 9: One of the authors using THRED. The user is in a relaxed position with the weight of each forearm balanced on each elbow.

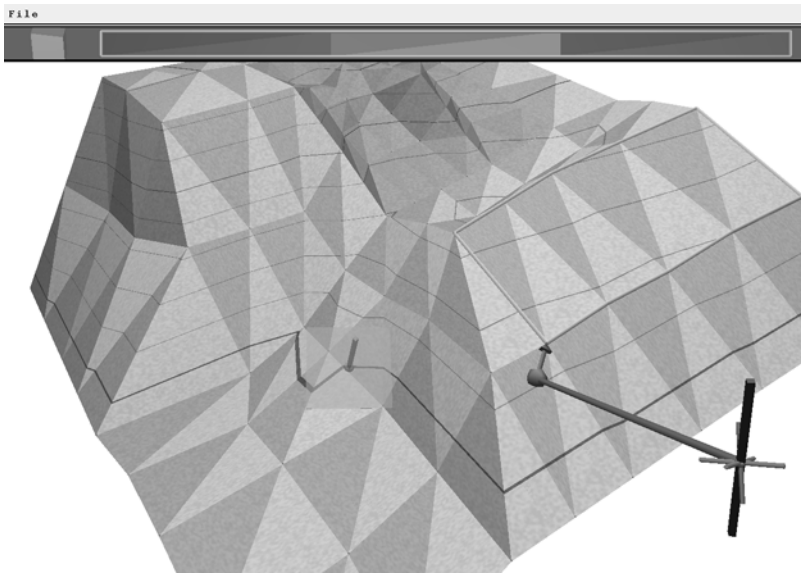

Figure 10: Sketch of the Standish Face of the Sunshine Village Ski resort at Banff, Alberta.

telling the user which bats is being manipulated. Our left bat is also used on a DataGlove, and has a patch of velcro on it, while the right bat does not. When the user picks up both bats, the presence or absence of velcro tells immediately which bat is which. Figure 9 shows one of the authors using THRED.

7 Sample Surface

Figure 10 contains a screen shot of a surface designed using THRED. It is a rough approximation of the Standish Face of the Sunshine Village Ski resort at Banff, Alberta. The lift line goes from the bottom left to the top left, and skiers with moderate skill (i.e. the authors) typically ski into one of the two bowls to the top center and the top center right. Experts ski down the face in the top left. This model contains 428 polygons, and was created in 18 minutes from personal memories of the first author.

The left cursor is about dead center and the right cursor is to the bottom right. THRED is in line constraint mode, and the right cursor is snapped to the vertical axis. Some squares to the right are currently selected, awaiting modification. The level refinement window shows that all levels are selectable, and indicates that THRED is in tandem reshape mode.

8 Conclusions and Future Work

We have described a computer-aided design system for creating hierarchical quadrilateral based surfaces. The interface to the system uses two hands to interact with the surface, with the left hand setting geometric and other context, and the right hand manipulating the surface geometry. In constrained manipulation, the left hand provides both the constraint mode and the constraint geometry for the right hand. This conforms to Guiard's [13] principles of bimanual action:

- The left hand provides the base frame of the right hand's manipulation. The left hand positions and orients the object, and sets the constraint geometry.
- The left hand moves first, setting the object position and constraint geometry before the right hand moves.
- The left hand moves at coarser resolution – constraint geometry snaps to one of the three canonical axes.

As a result, the two-handed interaction is fluid. The user typically does not spend much time stating the constraint mode or the constraint axis, and the direct manipulation of the scene to examine the surface is easy to use.

Future work on this system includes enriching the vocabulary of geometric operations, and evaluating this system with real users.

References

[1] Michael Bajura, Henry Fuchs, and Ryutarou Ohbuchi. Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient. *Computer Graphics (Proceedings of SIGGRAPH '92)*, 26(2):203–210, July 26-31 1992.

[2] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony DeRose. Toolglass and Magic Lenses: The See-through Interface. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 73–80, August 1993.

[3] Eric Allan Bier. Skitters and Jacks: Interactive 3D Positioning Tools. In *Proceedings of ACM 1986 Workshop on Interactive 3D Graphics*, pages 183–196, Chapel Hill, North Carolina, 1986. ACM SIGGRAPH.

[4] M.L. Braunstein. *Depth Perception Through Motion.* Academic Press, New York, 1976.

[5] J. Butterworth, A. Davidson, S. Hench, and T.M. Olano. 3DM: A Three Dimensional Modeler Using a Head-Mounted Display. In *Proceedings of 1992 Symposium on Interactive 3D Graphics*, pages 135–138, Cambridge, Massachusetts, March 29 - April 1 1992. ACM SIGGRAPH.

[6] William Buxton and Brad A. Myers. A Study in Two-Handed Input. In *Proceedings of ACM CHI'86 Conference on Human Factors in Computing Systems,*, pages 321–326, Boston, Massachusetts, April 13-17 1986.

[7] Jack Callahan, Don Hopkins, Mark Weiser, and Ben Shneiderman. An Empirical Comparison of Pie vs. Linear Menus. In *Proceedings of ACM CHI'88 Conference on Human Factors in Computing Systems*, pages 95–100. ACM SIGCHI, Washington, DC, 1988.

[8] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.

[9] James H. Clark. Designing Surfaces in 3-D. *Communications of the ACM*, 19(8):454–460, August 1976.

[10] Michael Deering. High Resolution Virtual Reality. *Computer Graphics (Proceedings of SIGGRAPH '92)*, 26(2):195–202, July 26-31 1992.

[11] P. M. Fitts. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, 47:381–391, 1954.

[12] David Forsey and Richard Bartels. Hierarchical B-Spline Refinement. *Computer Graphics (Proceedings of SIGGRAPH '88)*, 22(4):205–212, August 1-5 1988.

[13] Yves Guiard. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *The Journal of Motor Behavior*, 19(4):486–517, 1987.

[14] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassel. Passive Real-World Interface Props for Neurosurgical Visualization. In *Human Factors in Computing Systems CHI'94 Conference Proceedings*, pages 452–458, Boston, Massachusetts, April 24-28 1994. ACM SIGCHI.

[15] Logitech Inc. *2D/6D Mouse Technical Reference Manual.* Fremont, California, 1991.

[16] Paul Kabbash and William Buxton. The "Prince" Technique: Fitts' Law and Selection Using Area Cursors. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1, pages 273–279, Denver, Colorado, May 7-11 1995.

[17] Paul Kabbash, William Buxton, and Abigail J Sellen. Two-Handed Input in a Compound Task. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1, pages 417–423, 1994.

[18] Gordon Kurtenbach and William Buxton. User Learning and Performance with Marking Menus. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1, pages 258–264, 1994.

[19] Gordon P Kurtenbach, Abigail J Sellen, and William A S Buxton. An Empirical Evaluation of Some Articulatory and Cognitive Aspects of Marking Menus. *Human-Computer Interaction*, 8(1):1–23, 1993.

[20] Jiandong Liang and Mark Green. JDCAD: A Highly Interactive 3D Modeling System. *Computers and Graphics*, 18(4):499–506, 1994.

[21] Jiandong Liang and Mark Green. Geometric Modeling Using Six Degrees of Freedom Input Devices. In *3rd International Conference on CAD and Computer Graphics*, pages 217–222, Beijing, China, August 23-26, 1993.

[22] Warren Robinett and Richard Holloway. Virtual-Worlds Research at the University of North Carolina at Chapel Hill. In *Proceedings of 1992 Symposium on Interactive 3D Graphics*, Cambridge, Massachusetts, March 29 - April 1 1992. ACM SIGGRAPH.

[23] Emanuel Sachs, Andrew Robers, and David Stoops. 3-Draw: A Tool for Designing 3D Shapes. *IEEE Computer Graphics and Applications*, 11(6):18–24, November 1991.

[24] Chris Schmandt. Spatial Input/Output Correspondence in a Stereoscopic Computer Graphics Work Station. *Computer Graphics (Proceedings of SIGGRAPH '83)*, 17(3):253–261, July 1983.

[25] Chris Shaw, Mark Green, Jiandong Liang, and Yunqi Sun. Decoupled Simulation in Virtual Reality with The MR Toolkit. *ACM Transactions on Information Systems*, 11(3):287–317, July 1993.

[26] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual Reality on a WIM: Interactive Worlds in Miniature. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1, pages 265–272, Denver, Colorado, May 7-11 1995.

[27] Colin Ware, Kevin Arthur, and Kellogg S. Booth. Fishtank Virtual Reality. In *Human Factors in Computing Systems INTERCHI'93 Conference Proceedings*, pages 37–42, Amsterdam, April 24-29 1993. ACM SIGCHI.

[28] Colin Ware and Danny R. Jessome. Using the Bat: A Six Dimensional Mouse for Object Placement. In *Proceedings of Graphics Interface '88*, pages 119–124, Edmonton, Alberta, June 6-10, 1988.

[29] Shumin Zhai, William Buxton, and Paul Milgram. The "Silk Cursor": Investigating Transparency for 3D Target Acquisition. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1, pages 459–464, 1994.