# Two-Handed Polygonal Surface Design

*Chris Shaw, Mark Green*

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

**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, and generally acts as a counterpoint to the dominant hand. In this paper, the architecture of the system is outlined, 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 the design task still more difficult because these devices each have one degree of freedom less than the design domain. 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 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 [17], 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. Of course, some operations in a CAD system are naturally 2D or 1D, so the interaction techniques that use a 3D tracker 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[9], 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, outlines 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 [6] 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 was ahead of its time, and 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 [4], 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.

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 [13] 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 [15]. 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 [14] 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 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.

Jiandong Liang at University of Alberta built a system called JDCAD [12, 11] 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 one can create objects in JDCAD is gratifying enough to attempt using the non-immersive interaction style for other design tasks.

## 3 MODEL STRUCTURE

Like JDCAD, THRED is based on an SGI 4D/310 VGX graphics workstation and a pair of Polhemus Isotraks with buttons (Bats) [17], 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 [9]. 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

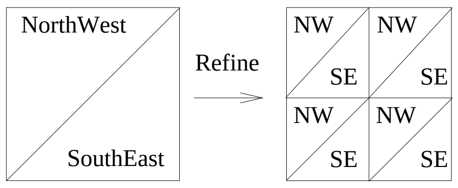

Figure 1: A single square and its refinement into four sub-squares.

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 the 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[9], with the obvious exception that the Dragon Editor manipulates B-Splines.

### 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 descendant geometry. 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 [3], 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 [7, 16], 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 sticks 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 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. All but the last pixel is 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. However, because of geometry errors in the VGX texturing library, the contour texture does not work very well, but the effect is promising.

To get both textures on the surface simultaneously, we would 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. However, unlike the new OpenGL standard, VGX GL does not guarantee that rendering the scene twice will give the correct result, so we just draw with the random texture.

## 3.2 Modeling Interface

When THRED starts up, the user is presented with a window containing two subwindows and the top menu bar. 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.

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, so that cursor separation on the screen is the same as the separation of the bats in real space. 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 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.

Similar to Bier's system [2], control point selection for each cursor is performed by constructing a line from the center of projection through the cursor into the scene. A cone whose axis is this line and which has a 5 degree apex spread is used to initially reject control points. All points that lie within the cone are then sorted by depth, and the closest control point is the current hot spot for the cursor. The cone selection solves the problem of distant items being more difficult to select than closer ones, and although we do not draw it, the cone selection volume projects to a circle of fixed radius on the screen.

The right hot spot is highlighted with a 3D arrow with a cylindrical shaft and a conical arrowhead, with the tail of the arrow at the cursor and the arrowhead at the hot spot on the surface. The left hot spot is simply drawn with a thick line from its hot spot to the cursor. When a cursor has no control points under its selection cone, the previous hot spot is used. Also, if a control point is at a deeper level of refinement than the currently selected refinement level, then that control point will not be added to the hot spot list.

## 3.3 Right Hand Operations

The right hand has three major tasks to perform:

- Control point selection and selection adjustment.
- Reshaping the selected area.
- Refining the selected area. This operation just refines all selected squares at the current level.

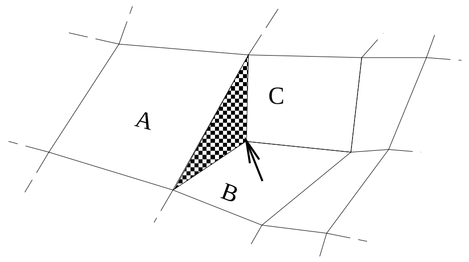

Figure 2: A crack (the shaded triangle) appears between squares A, B and C because the marked vertex joining B and C should be on A's edge. THRED avoids this condition by disallowing edits of these vertices.

*3.3.1 Control Point Selection* For control point selection, the user may select one control point, or a rectangular collection of points. To select one point, button 1 is pressed and released on the desired hot spot 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 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.

*3.3.2 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 three reshaping operations are *move every-*

*thing in tandem*, *cone* reshape, and *pyramid* reshape. 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 square moves in tandem with the right cursor.

The center square 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 reversing 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.

Pyramid reshape is similar to the cone except that instead of calculating the influence factor based on the distance from the cone axis, the influence factor is based on the distance from the X or the Y plane, depending on which face of the pyramid the vertex is on. As with the cone, the peak of the pyramid is centered on a vertex.

## 3.4 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.
- Manipulate the position and orientation of the entire scene. This mode 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 [14], 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.
- Set the current reshape shape. This is a ring menu [12, 11] (figure 3) which has the complete vocabulary of

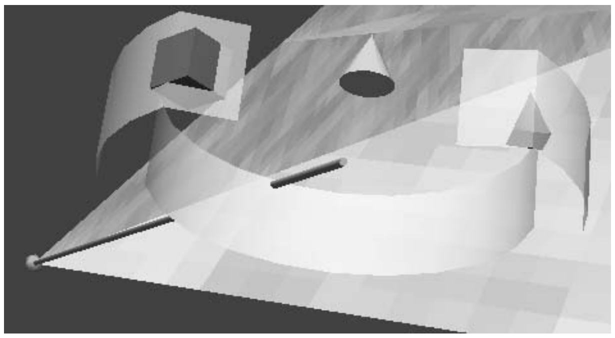
Figure 3: The ring menu. Cone reshaping is currently selected. Tandem move is half behind the translucent band on the left, and pyramid is on the right.

reshape operators on it, and the user twists the ring to make the current selection appear at the selection gap. JDCAD uses an identical mechanism to select primitive shape.

### 3.4.1 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.

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 [12, 11] 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 long 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

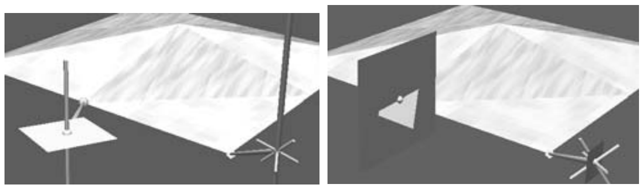

Figure 4: Left: Line constraint mode is active. Selected vertices can only move vertically.
Right: Plane constraint mode is active. Selected vertices may move in vertical plane shown by the left cursor.

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 plane rotating freely with the left bat, as shown in figure 4. The left cursor also has a small snapped square to help the user get to the desired orientation.

*3.4.2 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

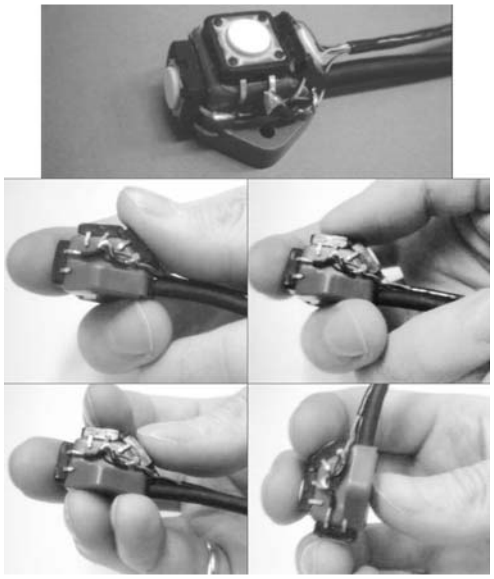

Figure 5: Top: The button-enhanced Bat.
Middle and bottom: Four postures for holding the bat

selection mechanism helps in enlarging the targets so that they can be hit quickly [8].

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 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.

## 4 DEVICE ERGONOMICS

The top of figure 5 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 5 middle left). Shifting between button 3 and button 2 is a simple matter of rocking the thumb over the pivot of the button casings (fig 5 middle right). 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.

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 5 bottom left).

This bat is an improvement on commercial products such as the Logitech 2D/6D mouse [10] 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 [5]. 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 5 bottom right). Also, the lighter weight of the bat limits the amount of finger fatigue users experience.

### 4.1 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, enter numbers for vertex coordinates, and to enter infrequent commands.

As might be expected, the 6 bat buttons are prime real estate, so commands which are executed infrequently such as *scale scene* are one-character keypresses on the keyboard. 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.

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 [14] 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 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 the presence or absence of velcro tells immediately which bat is which.

## 5 SAMPLE SURFACE

Figure 6 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 centre and the centre right. Experts ski down the face in the middle left. This model contains about 300 polygons, and was created in about half an hour.

The left cursor is about dead center and the right cursor is to the 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.

## 6 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.

The system described here is an early prototype, but 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, allowing for constrained manipulation along arbitrary axes, 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 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.

3. M.L. Braunstein. *Depth Perception Through Motion.* Academic Press, New York, 1976.

4. 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.

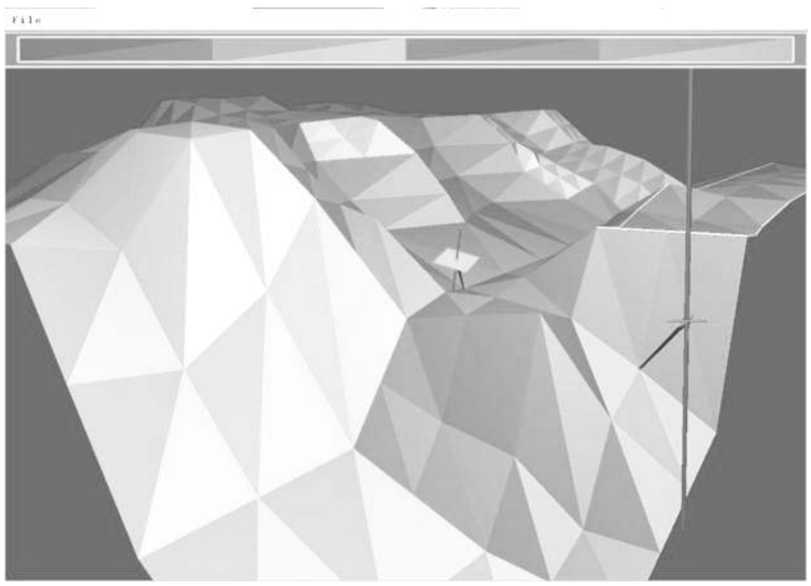

Figure 6: Sketch of the Standish Face of the Sunshine Village Ski resort at Banff, Alberta.

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

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

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

8. 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.

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

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

11. Jiandong Liang and Mark Green. Geometric Modeling Using Six Degrees of Freedom Input Devices. *Computers and Graphics*, 18(4), 1994.

12. 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.

13. 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.

14. 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.

15. 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.

16. 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.

17. 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.