3D I/O

Michael Cohen, Rik Littlefield,
Stephen Mann, Kenneth Sloan

FR-35, Department of Computer Science
University of Washington
Seattle, WA 98195

Technical Report 88-06-01
June, 1988

Abstract

23 ¿ The 3D design space of GRAIL provides a context for exploring some dimensions of multimedia I/O, including 3D visual output, 3D haptic input with a DataGlove, and musical output with a synthesizer. Three projects are motivated and described: *catch*, a game exercizing 3D visual acuity, *jester*, a 3D mesh editor, and *airDrum*, a virtual musical instrument.

Contents

List of Figures

List of Tables

1 Introduction & Motivation

Early operating systems allowed only textual I/O. Because the user read and wrote vectors of character strings, we call this mode of I/O one dimensional, 1D. As terminal technology improved, users could manipulate graphical objects. Although the I/O was no longer unidimensional, it was still limited to the planar dimensionality of a CRT or touchpad. The latest phase of I/O devices approaches the way that people deal with "the real world"[Bol84]. There already exist 3D spatial pointers and 3D graphics devices. 3D audio (in which the sound has a spatial attribute: originating, virtually or in actuality, from an arbitrary point with respect to the listener) and more exotic spatial I/O modalities are on the technology horizon.

The evolution of I/O devices can be roughly grouped into generations, that also correspond to the number of dimensions[Fol87]. Representative instances of each technology are shown below in Table 1.

dimensions	mode	input	output
1D	textual	keyboard	teletype
2D	planar	trackball, joystick mouse touchpad light pen	graphical display devices
3D	audio	speech recognition	speech and sound synthesis: DECTalk MIDI
	touch/"haptic"	3D trackball, joystick DataGlove (bat)	tactile feedback devices: glove with vibrating fingertips force-feedback joystick Braille devices
	olfactory	??	?
	taste	??	?
	visual	head- and eye-tracking	stereoscopic systems: 3Display vibrating mirrors head-mounted displays projective holography

Table 1: Generations and dimensions of I/O devices

This report describes a particular three dimensional input/output (3D I/O) system, and a few experimental applications.

2 The DataGlove: 3D Haptic Input

2.1 What the Glove Provides

The VPL DataGlove consists of a nylon fabric glove, on which are mounted several sensors, plus a controller box that talks to the sensors and host computer. The combination allows a host computer to access the following information:

1. Position of the wrist (3D rectangular coordinates).
2. Orientation of the wrist (3D angles).
3. Joint angles for the two lower joints of each finger and the thumb.

From the standpoint of the computer, this information is accessed via binary messages on a standard serial I/O line (RS-232 or RS-422). The computer can poll for the information, or the controller can be set up to send it periodically or upon significant change. In addition, messages can be sent to the controller to reset the system, to specify what information is desired, and to setup and query some sensor parameters. The formats of these messages are described (incompletely) in the VPL manuals [VPL, VPL87].

The controller can run at a variety of communication speeds up to 19.2K baud, and can provide data as frequently as 60 times per second. Theoretically, communications are a bottleneck at high sampling rates. In practice, the tradeoff between the sampling rate and the amount of data selected seems to be imposed by host cpu saturation.

To develop software that uses the glove, it is helpful to have some understanding of how a DataGlove really works. The following paragraphs provide a quick overview. To delve deeper, see [VPL, VPL87].

To gather the three types of information noted above, two kinds of sensors are used: *Polhemus* (a brand name[Pol87]) and *flex* (a description). The Polhemus sensor measures coordinates and orientation, and the flex sensors measure joint angles.

The Polhemus sensor consists of two units: transmitter and receiver. The transmitter is fixed relative to the workspace, and contains three coils, orthogonally oriented, sequentially emitting an electromagnetic pulse. The receiver, fastened to the wrist of the glove, also has three orthogonally oriented antennae, which detect the pulses emitted by the transmitter. The combination of the induced sensor currents allows the controller to derive position and orientation. These measurements are relative to the transmitter, but since the transmitter is fixed, the measurements are effectively absolute.

Note that the data reported are for the wrist only. To determine the position or orientation of any finger, the host computer must transform appropriately based on some model of the hand geometry.

The flex sensors consist of optical fibers that have been modified to leak light when bent at certain places. Each fiber is mounted on the glove so as to hold its sensitive point directly over the joint to be measured. The fiber is looped so that both ends are accessible to the controller. The controller sends light into the fiber, and measures how much gets through. The joint angle can then be inferred from the amount of light lost. Unlike the Polhemus sensor, the flex sensors have nonlinear behavior and range limitations that can change for different users and over time. To compensate for these, the host computer can set the light intensity and read the raw data values. This allows the system to be calibrated using whatever procedure is appropriate to the application. After calibration, conversion of sample points can be done by the host computer, or lookup tables can be loaded into the controller, which can then report joint angles directly.

There are 13 flex sensors mounted on our glove, all of which can be read by the host computer. The thumb and each finger has a sensor for each of its bottom two joints (10 sensors total). These sensors correspond to the ones described in the VPL manuals. In addition, the thumb and first two fingers have an extra sensor on the bottom joint.

2.2 Limitations of Glove Data

There are several limitations on the data provided by the glove. These result from noise, nonlinearity, mechanical sloppiness, and degrees of freedom that are not measured by the glove.

The Polhemus sensor is vulnerable to noise, nonlinearity, and mechanical sloppiness. Noise and nonlinearity are inherent in the Polhemus sensing technology, particularly when operated in the presence of metal objects and competing magnetic fields, such as CRT deflection yokes. In addition, and probably more serious, the stretchiness of the glove and Velcro attachment enable the sensor to move relative to the wrist. We have not made any careful measurements of these effects. However, a rough guess based on casual observation is that the position data is probably repeatable to within 0.1 inch, while the orientation data may be off by as much as 20 degrees. We have no data on position accuracy (linearity). In addition, we have occasionally seen single data points with spurious values.

The flex sensors are primarily vulnerable to nonlinearity, especially in the form of clamping for extreme angles. The transfer function from joint angle to raw flex value is described by VPL simply as "exponential" [VPL]. (No exact formula is given.) In practice, it appears to be a more complicated function with sharp curvature near cutoff and saturation, and slight curvature in between. Our applications have no need for great accuracy in the joint angles, so we simply pretend that the relationship is linear between 0 and 90 degrees.

A fundamental limitation of the joint sensors is that they do not provide enough information to completely determine hand position. For example, a *pinch* gesture with the thumb and little finger can have exactly the same joint angles as simply curling the fingers. The pinch is accomplished by curling the base of the hand, a movement for which the glove has no sensors.

The VPL documentation describes a third type of sensor, called *Hall sensors*, which are supposed to provide a direct measurement of the distance between the thumb and each finger. This measurement would be useful for detecting the final closing of a pinch gesture. However, these sensors were not commercially available as of February 1988. VPL doesn't sell them, and we don't have any.

3 Stereographics: 3D Visual Output

To achieve the effect of three dimensional images on a flat monitor, stereo pairs must be generated and synthesized in a manner such that a viewer sees a single, three dimensional image. This section describes stereo pairs, the graphics package we used to generate them, the hardware they are displayed on, and the stereo cues we ignored.

3.1 Stereo Graphics

The standard way to display a true three dimensional image on a flat screen monitor is to generate a *stereo pair*, a pair of images, one for the left eye view, and one for the right eye view. These images are then presented so that the left eye sees the left eye view and the right eye sees the right eye view. This section will discuss the generation of stereo pairs.

In classical computer graphics, the scene being rendered is at some point passed through a perspective projection. This projection is what gives the image a feeling of being three dimensional: objects closer to the viewer are larger than those farther away. The human visual system differs from this in that a person actually sees two images, one with each eye. The eyes are separated by a small distance, known as the *inter-ocular distance.* This distance is 65mm for the typical man and 63mm for the typical woman. Because the eyes are in different positions, they see slightly different images. The difference in these images is known as *parallax*, and is what produces the three dimensional effect. A stereo pair is this pair of images that a viewer's eyes would see.

In the human visual system, the eyes converge at a point, and focus on a point. In normal viewing, these two points are the same point. When viewing stereoscopic images, the eyes will converge at points of different depths, but will always be focused on the screen. This presents some problems in trying to generate stereoscopic images that are comfortable to view.

The two images seen can be thought of as the scene projected onto a plane. Each image seen is projected onto a separate projection plane, the human retina. The images are then composed in the brain. In computer graphics, this is approximated by projecting through the computer screen. When both views of a single point are displayed simultaneously on a monitor, the point will appear as two points, separated by a short distance. This distance is known as *disparity.* Depending on the location of the point being viewed, there will be positive, negative, or zero disparity. Points lying in a plane parallel to the viewer (and the viewer's eyes) will have the same disparity.

Since the eyes are always focused on the monitor screen, it is easiest on the viewer to center the object being viewed on the screen. This leaves the eyes in a natural (or close to natural) state. Thus, in talking about disparity, we will assume that the screen is the point/plane of convergence. Points lying on the plane of convergence will have no disparity (figure 1a), points lying between the plane of convergence and the viewer will have negative disparity (figure 1b), and points lying on the far side of the viewing plane will have positive disparity (figure 1c). Thus, scenes which are comfortable to view may have objects that extend farther behind the screen than in front of it.

If the disparity of part of an image is too great, then the human visual system is unable to fuse the stereo pair into a single image. This constrains objects being viewed to be of limited depth. Note that objects on the far side of the viewing plane will never have disparity greater than the ocular distance. Objects on the near side of the viewing plane can mathematically have unbounded disparity, though physically the disparity is limited by the size of the screen.

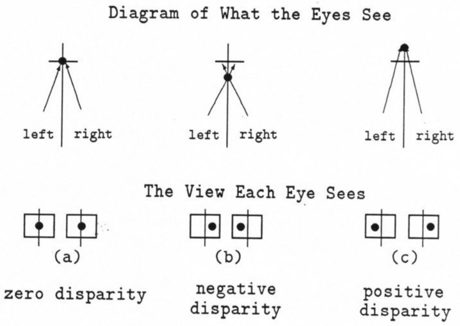

Figure 1: Disparity

Note that the disparity of a point is parallel to the line through the eyes. Our graphics package assumes that the line through the eyes is parallel to the horizontal lines of the screen. Thus, in our stereo pairs, there is never any vertical disparity.

3.2 Hardware

Special hardware is needed to display stereo pairs. There are various methods for doing this, with the methods breaking down into three classes: those that display both eye images simultaneously and filter out the correct image for each eye; those that rapidly alternate between the two images, blocking the vision of each eye in synchronization with the displayed images so that the viewer sees the correct three dimensional view; and those that use a head mount, showing each eye view separately.

The hardware we are using (3Display[1][Ste85]) was of the second type. A stereo effect is achieved by drawing a left eye/right eye half high pair in a frame buffer. The video signal of this frame buffer is fed into the 3Display box, which alternately displays the right and left eye views on a Conrac monitor. Special goggles are used to block the vision of the left eye while the right eye image is on the monitor (and vice-versa). See figure 2.

A stereo pair is loaded into an Adage Frame Buffer[Iko82]. Only the video signal from the frame buffer is needed; the actual image in the frame buffer need never be seen. However, if it is viewed, the left eye view appears on the top half of the screen, and the right eye view on the bottom half. The Adage is being used in a mode that has a resolution of 512 by 512 pixels, with 8 bits of red, green, and blue, for a total of 24 bits of color. Only 4 bits of each channel are used at a time to allow for a double buffering scheme. Each eye view is about 340 by 220 pixels. If viewed separately on the Adage, each image looks compressed in the vertical direction. In addition to the frame buffer, the Adage has a bit slice microprocessor.

The video signal from the Adage's frame buffer is fed into the stereographics box. This box alternates displaying the left and right eye views on a Conrac monitor. The Conrac 7241[Con86] is a 120 Hz monitor. Each eye view is displayed on this monitor 60 times a second. This higher display rate was chosen (by Stereographics) to eliminate a problem of flickering that occurs when a 60 Hz monitor is used.

The Stereographics box also controls goggles that must be worn to view the image. These goggles have lenses which can be electronically darkened. The Stereographics box synchronizes the goggles with the image displayed on the screen so that the left eye sees the left eye view, and the right eye sees the right eye view [Ste85].

One problem we encountered with this hardware is that "ghost" images of the other eye's view would appear. These ghosts are due to slow phosphor decay time, so when switching images, say from the left eye view to the right eye view, the left eye image

[1]3Display is a trademark of Stereographics Corporation

can be faintly seen by the right eye. These ghosts are extremely detrimental to the stereo effect; if the ghosts are strong enough, the viewer is unable to fuse the image. The ghosts are most apparent in regions of extreme contrast. Unfortunately, as we are doing line drawings, they appear anywhere on the screen where there is something to look at. However, the green phosphor is the only one of the three phosphors whose decay time is too slow. We were able to make our images viewable by eliminating green from the static parts of the images, using it only for highlighting and flashing.

3.3 Graphics Package

The requirements for our stereo line drawing package were for it to be fast enough to support interactive applications; have some color support; and to change the displayed image quickly (in one frame time). The applications we intended to write would not modify many line segments between images; most of the calculations for one image could be used in the next image.

Our graphics package was targeted to be run on a MicroVAX, drawing into an Adage frame buffer. Since the Adage has its own microprocessor, the drawing computation is divided between the two cpu's. The MicroVAX transforms the coordinates of the end-points of the lines from world space to device coordinates. These device coordinates are sent to the Adage, whose processor draws the lines into the frame buffer using Bresenham's line drawing algorithm. Lines have a color associated with them. Each segment may be only one color, though different segments may have different colors.

In order to reduce the communication from the MicroVAX to the Adage, the line segments are buffered after they are transformed to device coordinates. When the buffer fills up, or if an explicit command is given, the line segments are sent to the Adage. A programmer may also provide buffers to use; these buffers may then be downloaded multiple times. This is useful if there is an object that doesn't change from frame to frame (the cost of transforming to device coordinates will only be paid once).

A double buffering scheme is used to allow for smooth transition between images. This was implemented by having two color maps; one color map uses the high order bits, the other uses the low order bits. A write mask is used to allow one image to be drawn while the previous one is displayed.

Jester had somewhat different requirements. While most of one image would remain unchanged in the next image, there weren't well defined objects; any one of the endpoints of the line segments could possibly change, and an arbitrary number of line segments might be affected by this change. Two new calls were added: one to transform a point (giving device coordinates) and another to add a line to a buffer by giving these device coordinates. See section 4.3.6 for a discussion of how *jester* handles its graphics.

3.4 Stereo Cues Ignored

The only three-dimensional cue implemented in the graphics package is one of stationary parallax. There are several other possible cues, such as texture and shading, intensity, and motion parallax.

Our projects employed only wire frame drawings. In separate research, we have generated static three dimensional images with shaded textures. These images were rendered with a ray tracing package. A few images (without texture) were also generated with a zbuffer package. While both the ray-traced images and the z-buffer images look very good, they take too long to generate on our hardware to be of use in an interactive design system. In addition, it is convenient for the designer to be able to see the backside of the object being designed. With a fast, high quality renderer, one idea is to let the designer build the object using line drawings, with the added ability of being able to hit a button and see a medium to high quality rendering of object in a few seconds. The designer would then be able to switch back to the line drawing to make further adjustments.

Another depth cue which we haven't implemented is intensity. The idea is to brighten objects based on their closeness to the viewer. Objects farther from the viewer would be dimmer ones nearer to the viewer. Color calculations for the end points of the lines could be done on the MicroVAX, and the colors interpolated in the micro-code drawing routines. This probably would not significantly increase the computation time.

A third stereo cue we have ignored is motion parallax. Motion parallax occurs when a person moves his or her head. In the real world, there are slight changes in what is seen when the viewer turns his head. Closer objects seem to shift more than farther ones. These changes provide a depth cue that can be seen even with a dynamic monocular view.

The stereo equipment we have provides no information about the position of the head; one method of gathering such information would be to place a polhemus on the viewers head. Thus, motion parallax is an effect we are unable to implement. In addition, our computer hardware is not fast enough to render the constantly changing scene as would be required to implement motion parallax. And so, thus far, we have been constrained to rendering stationary parallax.

While stationary parallax seems to be adequate, there is one effect that is disturbing to the viewer. When the head is moved from sided to side, one sees slight changes in the objects viewed. However, the image on our monitor doesn't change when the viewers head moves. This causes the viewer to perceive the object as if the back of the object is moving the same direction as the viewer's head, while it should appear stationary.

4 Projects

The projects described in this section represent applications of the technology described earlier in this paper. They were built in the 3D Design Space of GRAIL (the Graphics and AI Lab) of the Computer Science Department at the University of Washington. A brief history of the architecture of our system is given, followed by a discussion of *catch*, *jester*, and *airDrum*.

4.1 Architecture

Our projects are implemented in a distributed fashion, load-sharing the cycle-intensive DataGlove polling and interpretation, graphical display, and musical output routines. (See Figure 2 below.) *Catch* is localized to a MicroVAX. For *jester* and *airDrum*, though, a program (running on a Sun 3) is responsible for polling the DataGlove and sending the parameters across an ethernet socket to processes (running on a Sun 3 or MicroVAX) that interpret the gesture, and update a graphics server or spawn musical events.

This distributed architecture allows and even encourages extension of our systems. For example, the networked I/O devices may be used in applications that require specialized execution environments available only on particular hosts. Also, our system is extensible, in that, for instance, it could easily be enhanced by the addition of extra DataGloves.

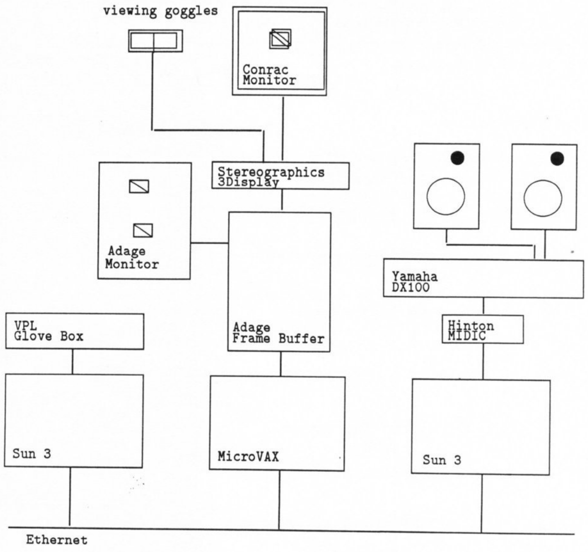

Figure 2: 3D I/O Architecture

4.2 *catch* — a 3D Visual Acuity Game

It was clear from the start that our hardware was not fast enough to support shaded surfaces for the applications we wished to develop; line drawings were the best we could hope to render quickly enough. Two questions immediately arose: was the hardware fast enough to support stereo line drawings? Could people accurately position in 3D based on line drawings? It was with these two questions in mind that the simple game *catch* was developed.

4.2.1 Description

A user playing *catch* sees a box, with a pad on the floor of the box. The pad is tied to a tether, with the other end of the tether fixed to the center of the box (the box and tether are depth cues that will be discussed shortly). By moving the mouse, the user is able to position the pad anywhere on the floor of the box. See figure 3

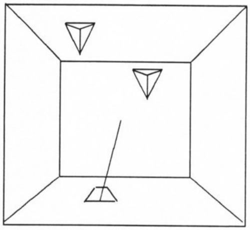

Figure 3: Monoscopic view of *catch*

At random times, a tetrahedron will appear at a random location on the ceiling of the box. After a short delay, these tetrahedra drop to the floor of the box. The object of the game is to catch these falling tetrahedra with the pad. When a tetrahedron is caught, the pad briefly changes color.

The mouse buttons affect the playing field in the following fashion: the left button toggles the box on and off, the middle bottom toggles the tether, and the right button creates more tetrahedra for the impatient.

4.2.2 Implementation

Catch was implemented on a Micro-VAX workstation, using the graphics package described in the previous section. Three display buffers were used: one for the box, one for the pad, and one for the tetrahedra. A large X window covering the work station screen was used to read mouse input.

4.2.3 Depth Cues

The primary depth cue appears to be the parallax from the perspective viewpoints. Tetrahedra of different sizes are no harder to catch than those of uniform sizes. The box and tether have a more interesting affect. While the user is still able to catch the tetrahedra without one or both of these cues, most users noted a loss of a sense of absolute position of the tetrahedra when the box and tether were not present. No other depth cues (such as intensity, motion parallax, etc) were tested.

4.2.4 Results

Catch answered the two questions asked: our hardware is fast enough to render stereo line drawings in real-time, and users are able to use stereo line drawings to locate objects in three dimensions. Clearly, cues such as a bounding box and a tether enhance a user's capability to locate objects.

4.3 *jester* – a Mesh Editor

4.3.1 Overview

The *jester* program was written primarily to demonstrate some key concepts of 3D input and display, as they might be applied in a 3D design environment. *Jester* is a very simple interactive graphical editor for *meshes* of 3D points. Prior to *jester*, the only means available at GRAIL for generating such meshes was to manually specify coordinates and topology, or to write a program to compute them. Both approaches are tedious and error prone, thus providing a backdrop against which *jester* looks both friendly and useful.

Jester uses the DataGlove as the main input device, and the Stereographics system for display. The mesh being edited is continually displayed in stereo, along with a 3D cursor. Cursor position is bound directly to DataGlove position, so that the user "flies" the cursor using intuitive arm movements. Edges or points in the mesh can be picked simply by positioning the cursor near them. As usual, picked items are highlighted. Once an item is picked, the user can operate on it by making some simple gesture. For example, closing the thumb and all fingers will "grab" a picked point, so that it moves with the cursor. The strategy is very similar to using a 2D display and a 2D mouse with buttons, except that *jester* runs in 3D and uses hand position. Describing the glove as a *bat* (a flying mouse, of course) seems both amusing and helpful.

4.3.2 Definition of Mesh

As implemented by *jester*, a *mesh* consists of a collection of planar triangular faces, such that each edge is shared by at most two faces. At present, the entire mesh must be connected by faces sharing edges. This definition produces a distinction between *internal* edges, which are shared by two faces, and *boundary* edges, which belong to only one face. A mesh may have no boundary if it closes back on itself, such as four faces arranged in a tetrahedron.

4.3.3 Editing Gestures and Keyboard Input

At present, four gestures are recognized:

- none (open hand) — No editing is being done, and items can be picked by positioning the cursor near them.

- grab (thumb and all fingers flexed) — If a vertex is currently selected, then it is bound to the cursor and moves with it. If something besides a vertex is selected, or if nothing is selected, nothing happens. When the bound vertex is properly positioned, it can be released by opening the hand.
- pinch (thumb and index finger flexed, other fingers straight) — If a boundary edge is currently selected, then a new face is "pulled out" of the edge by creating a vertex at the cursor position. The new vertex is then bound to the cursor so that it can be moved to the desired position.
- hold (thumb and all fingers: lowest joint flexed, other joints straight) — This gesture is reserved for use in changing the viewing orientation, i.e., "holding the mesh and turning it around". However, this function is not currently implemented.

In addition to recognizing DataGlove gestures, *jester* also checks continually for keyboard input. At present, only one keyboard command is recognized: "q" or "Q" for quit.

4.3.4 Display Conventions

The *jester* display of the mesh comprises just points and edges in 3D stereo. There is no explicit representation of faces. However, color coding is used to distinguish boundary and interior edges. In our limited experience, users have easily been able to infer where the faces are.

As customary in mouse-based systems, the cursor changes form depending on its use. In the present *jester*, there are three forms, corresponding to the three implemented operations:

- open hand – a 3D axis-aligned cross (3 vectors).
- grab – a 3D axis-aligned cube (12 vectors).
- pinch – a 3D asterisk consisting of the diagonals of a cube (4 vectors).

The cursor changes form to match the gesture, even if no edges or vertices have been selected. This has turned out to be quite useful as a training tool, giving new users immediate and obvious feedback of whether their gestures are being recognized.

4.3.5 DataGlove Calibration

Jester recalibrates the glove at the beginning of every execution. This was the easiest approach to handling a variety of users and avoiding potential problems with the glove drifting. The procedure takes about 10 seconds with a little practice:

1. The user straightens her fingers and hits a carriage return (CR).
2. The user curls her fingers to 90 degrees and hits CR.
3. The user curls her thumb to 90 degrees (outer joint) and 45 degrees (base joint), and hits CR.
4. The user positions her hand at one corner of the volume she wants to move her hand in, and hits CR.
5. The user moves to the diagonally opposite corner, and hits CR.

Gesture recognition is done by hardcoded rules based on certain joints being flexed in certain ways. The current rules are:

```
if (JointAngle[1] > 20 &&     /* outer thumb */
    JointAngle[3] > 20 &&     /* outer index */
    JointAngle[9] < 20    )   /* outer pinkie */
  gesture = PinchGesture;

else
if (JointAngle[1] > 30 &&     /* outer thumb */
    JointAngle[3] > 30 &&     /* outer index */
    JointAngle[9] > 40    )   /* outer pinkie */
  gesture = GrabGesture;

else
if (JointAngle[0] < 10 &&     /* inner thumb */
    JointAngle[2] > 20 &&     /* inner index */
    JointAngle[4] > 20    )   /* inner middle */
  gesture = HoldGesture;

else
  gesture = NoGesture;
```

This strategy was chosen mainly because it was quick to implement. It also turns out to be easy to explain. Although the rules are quite rigid, users seem to adapt to them quickly when told what they are, and given the immediate feedback from the form of the cursor. Actually, the major problem has been a misleading prompt: when told to straighten their fingers, most users seem to hyperextend them. This leads the calibration process to require an "open hand" position that is inconvenient if not actually uncomfortable. Probably it would be better if the calibration process used something like a "relaxed but open" hand as one endpoint.

There are several alternate strategies that might be considered: user-modifiable rules; calibration of gestures (as opposed to joint angles) for each user; or even adaptive pattern recognition (to handle long sessions in which one's hand might get tired or lazy). None of these possibilities have been explored.

4.3.6 Graphics

A *jester* scene can logically be broken down into two parts: the cursor and the mesh. The cursor has its own buffer for graphics (as described in an earlier section). The mesh is different, and requires further explanation.

Between consecutive frames, the mesh is mostly a static image. At most one vertex will change (though an arbitrary number of lines and faces may change). Faces are not drawn in *jester*, and are not part of the graphics data structure.

In order to reduce the amount of computation to a minimum, two lists are kept: one for the vertices, and one for the line segments. The lists are stored as arrays; the index of a point/line is a handle that the rest of the *jester* package uses reference it. The vertex array contains device coordinates for each point. In the array of lines, each line is stored as a pair of indices into the vertex list, and a color. When a vertex is changed, all of the lines sharing that vertex are automatically updated. The mesh is drawn by running through the array of lines and adding the pair of vertices indexed by each line to the display buffer.

This scheme reduces the amount of computation needed for transforming vertices of the mesh. This reduced cost in computing transformations is gained by performing more copying, which is a substantially less expensive operation.

4.3.7 Extensions to *jester*

To be blunt, *jester* was developed as a demonstration, not as a test or testbed for 3D interaction techniques. We have not made any structured effort to develop or compare a range of techniques, nor do we make any claims regarding the ability of *jester* to support such work.

There are several directions in which *jester* obviously could be extended, however. To be useful, many more editing operations are required, such as deleting faces, constructing faces from existing vertices, merging vertices, aligning coordinates, etc. Being able to change the viewpoint, or to zoom in on a part of the mesh, would be useful. In short, any feature appropriate to a 2D drafting tool seems likely to be useful in 3D as well.

It seems clear that recognition of single gestures (i.e., hand positions) will not be adequate to cover the number of commands that will be required, just as single keystrokes and mouse clicks are not adequate in 2D applications. Again though, any interaction technique that works in 2D should also work in 3D. Obvious suggestions include pull-down and pop-up menus (selected by DataGlove action, of course), or switching to the keyboard for really complicated commands.

In addition, the DataGlove opens some new possibilities. For example, the glove has enough sensors to allow a fairly sophisticated "sign language" consisting of temporal sequences of hand positions and movements. This area is completely unexplored.

4.4 *airDrum* — a Virtual Musical Instrument

4.4.1 Description: Throwing Down the Gauntlet Audibly

The *airDrum* project explores ways of using 3D input to produce and control music. Spanning human factors and computer music, it implements a (programmably arbitrary) style of haptic input. Using the DataGlove as a MIDI (musical instrument digital interface) controller, the *airDrum* software translates mechanical haptic input into MIDI events (messages). The semantics of the *airDrum* interpretation suggests a multi-timbred musical instrument, both in sound and style of play.

4.4.2 Implementation: Glove at Fist Site

The prototype *airDrum* was very coarse. Local vertical minima in the sweeping of the DataGlove'd hand were recognized using Polhemus data. The *ictae*, or beats, were audibly displayed by a "control G", the ascii bell.

Current versions feature a more interpretive range, with respect to both the sophistication of the audio output as well as the types of analysis done on the user's arm movements. The first step was to interface our interpretive module to a MIDI synthesizer, a Yamaha DX100[Yam], so that "middle G" could be played instead of "control G". The data from the DataGlove polls are analyzed to yield MIDI events. These are converted from RS-422 to MIDI format by a Hinton MIDIC module[Hin], which drives the synthesizer directly. (See Figure 2.) We also extended the significance of the user's wrist position, as parametized by the Polhemus data:

The lateral axis ("x") controls the pitch of the struck note, quantized and truncated into a 4 octave chromatic scale, the range of the synthesizer. This same axis might also control (linearly) the "location" of the sound, by varying the balance with a virtual "pan pot", or panoramic potentiometer.[2]

The proximal/distal axis ("y") selects the voice, or instrument, used to generate the music, by choosing from among several with different harmonic characteristics. The timbre of the virtual instrument is selected in realtime, just before striking the note. The current implementation of the *airDrum* resembles something in between a drum and a piano, with respect to both style of play and type of sound produced. Because realistic drum sounds are produced only by sampling drum machines, the percussion

[2]Unfortunately, our synthesizer didn't provide a parametezation of this variable, as more expensive systems do.

sounds most closely resemble chimes or a "hyperxylophone" instead of drums.[3]

The vertical axis ("z") is the one to which the most attention is paid: the main characteristic analyzed is the occurrence of the vertical cusps in the trajectory of the DataGlove'd hand. These local minima mark the ictae which trigger the timing of the notes. Additionally, the range and velocity of the swinging hand determine the dynamics (or volume[4]) and could also specify the duration[5] of the note.

These axes of interpretation are summarized in Table 2 below.

axis	direction	pitch	timing	volume	timbre	location	duration
x	left→right	√				(√)	
y	back→forth				√		
z	down→up		√	√			(√)

Table 2: Mapping dimensions of gesture to dimensions of sound

4.4.3 Performance Performance

Because of throughput constraints and network latency, the *airDrum* cannot play an arbitrarily fast drum roll. The DataGlove is polled at 30 Hz, which is too slow to get fast beats. It will be some time, for instance, before we can do justice to the drum solo in "Wipeout"[Sur63]. Currently, the *airDrum* can accurately recognize ictae at a frequency of about 2.5 beats/sec.

Due to noise in the Polhemus data (perhaps a product of our lab environment), we found it necessary to introduce a hysterisis to inhibit spurious ictae identification. These thresholds also prevent vibratory excitation— triggering the *airDrum* with small oscillations.

4.4.4 Future Work: *airBaton*

By connecting the *airDrum* to a sequencer, we change the *airDrum* to an *airBaton*, and the domain from realtime performance to realtime conducting. There are many

[3]so our virtual percussion is truly cymbolic

[4]Volume adjustment is done by globally changing the synthesizer output, rather then by employing the "key velocity" feature implemented on more expensive MIDI systems, but not by the DX100.

[5]Earlier versions of the *airDrum* experimented with adjusting the articulation (legato → staccato) of the played notes, based on the sharpness of the downward beat, but for performance reasons this feature was taken out.

similarities, of course, since a conductor coaxes sound from an orchestra in some of the same ways a musician plays an instrument. Certainly gesture interpretation of the type implemented by the *airDrum* blurs this distinction.

The goals of an *airBaton*[CDM88] are different from an *airDrum*, however. An *air-Baton* decouples the tempo from the rhythm, adaptively adjusting the tempo of the music in realtime, like a dynamic metronome, and leaving the rhythm to the sequencer. Instead of driving the music like an *airDrum*, whose back-end is reactive, an *airBaton* system must be proactive, anticipating the way music is meant to be shaped; an orchestral player *prepares* to play her note *on* a beat in a way characterized by the conductor's gesture *before* the beat.. Like a real musician familiar with the idiosyncrasies of an instrument, an *airBaton* system must match the conductor's direction to the music.

In some ways, it is easier with an *airBaton* to achieve satisfactory performance, since (a coarse approximation of) its implementation is less sample-intensive. But in other ways, the conceptual obstacles seem overwhelming. To exploit the (admittedly arguable) power of music to express all known human feeling, conducting — the interpretation of music's outline — seems full of arbitrary nuance.

Even though this generalized real-time 3D gesture recognition is difficult, we hope that a restricted subset of conducting will yield to the techniques explored by the *airDrum*. In practical terms, a useful instrument could be built controlling just three parameters— tempo, dynamics and articulation— which could significantly enhance computer music as a performance medium, especially in ensemble with live musicians. The limited and well-defined set of gross gestures (visible to the back row of an orchestra, perhaps 12–15 meters [40–50 feet] away) and the strong context provided by the sequencer make our approach feasible, yet interesting and subtle enough to illuminate more general gesture recognition problems.

The mapping of one medium into another (in this case, gesture $\rightarrow$ sound) challenges our intuitions about the essence of the media. By short-circuiting the musician's visual translation (of gesture $\rightarrow$ "musical thought" $\rightarrow$ gesture?), have we distilled or distorted?

While thinking about these issues, researchers might be tempted to exploit the expressive power of systems like a combined *airDrum/airBaton*. Like a player/manager, an artist/scientist might explore expressive media that blends elements of both playing and conducting.

Like conductors' conventions, people will convene (electronically[Coh88, Coh87] or otherwise) to communicate in mixtures of musical performance, dance, mime, and sign language, driving sound as well as visual effects like those suggested by *jester*.

For instance, it is amusing to juggle while wearing our *airDrum*. With notions of space as an acknowledged medium of communication, both as a "first class" input device (via devices like the DataGlove or DataSuit) and as a "first class" output device (via fully 3D visual [like that suggested by *catch* and *jester*], audio, and tactile systems), performance artists, perhaps working in ensemble, would enjoy an adaptive, interpretive, predictable, expressive medium.

Programmably mechanized and transmittable gesture interpretation— on a fine scale (like that employed by *catch* and *jester*) as well as on a coarse scale (like that employed by an *airInstrument* or *airBaton*)— expressed sonically, spatially, and every other way[Coh88], could evolve into a communication mode that transcends even a combination of cinema, theatre, dance and music and other conventional artforms.

5 Future Directions

Extensions include (orthogonally):

- combining the audio, tactile, and visual stimuli into a more complete "artificial reality"[Coh88]
- using multiple DataGloves for multiple hand and/or user input
- using a DataSuit (being developed by VPL) to sense whole bodies instead of just hands

6 Acknowledgments

James Painter ported the DataGlove software, writing optimized I/O routines in the process. Rob Duisberg and John Maloney provided help with the musical aspects of *airBaton* and *airDrum*. Ron Blanford and Robert Ling made architectural suggestions. Charles Loop wrote most of *jester*'s mesh-manipulating routines and contributed much to its user-interaction style. Without the help of all of these people, our projects would not have been succesful.

A Running the Demonstrations

All of these should be run in GRAIL (the Graphics and AI Lab), room 420 of Sieg Hall— the Computer Science Department at the University of Washington.

A.1 *catch*

(Use the default key bindings.[6]) Open an X window on bezier. Then run

```
bezier> /usr/graphics/demo/catch
```

Instructions, summarized below, appear upon startup.

- Left button — create a new tetrahedron.
- Middle button — toggle tether.
- Right button — toggle room frame.
- Use the mouse to position the pad on the floor underneath the falling tetrahedra.

[6]To reset the key bindings, type "`mv 7.uwmrc 7.uwmrc.sav`" and reinitialize the window bindings.

A.2 *jester*

A.2.1 *Canned visual simulation*

```
bezier> /usr/graphics/demo/jesterDemo \&> /dev/null
```

A.2.2 *jester*

Turn on the VPL GloveBox. Carefully put the DataGlove on your right hand. On bezier and marr, run the following in the indicated order:

```
marr> /usr/graphics/demo/GBD_vax -s
```

```
18              ColdReset
2               Repeat 30
y                   Change
y                     Flex
y                        Separate bright/dim
y                     Polhemus
n                     Hall
```

```
bezier> /usr/graphics/demo/jester -mi /usr/graphics/demo/init.m
```

Now follow calibration described in Section 4.3.5.

The cursor's shape will change according to the interpreted gesture:

- open hand – a 3D axis-aligned cross (3 vectors).
- grab – a 3D axis-aligned cube (12 vectors).
- pinch – a 3D asterisk consisting of the diagonals of a cube (4 vectors).

A line segment will change from red to green when it's selected. Internal and ungrabable elements remain red.

Type q to quit.

Be sure to shut off the VPL GloveBox when finished.

A.3 *airDrum*

Turn on the

- VPL GloveBox
- Yamaha DX100 synthesizer
- external speakers (or plug in the headphones)

Plug in the power cord to the Hinton MIDIC module.

```
marr> /usr/graphics/demo/MIDIC-init
```

(This should play an ascending, decrescendoing scale after about 4 seconds.)

Now enable the airDrum process:

```
marr> /usr/graphics/demo/airDrum
```

In another window on marr, invoke the DataGlove process with the appropriate commands:

```
marr> /usr/graphics/demo/GBD_sun -s
```

```
18              ColdReset
2               Repeat 30
y                   Change
n                     Flex
y                     Polhemus
n                     Hall
```

Now the *airDrum* is enabled. Carefully put on the dataglove and play. The playing area corresponds roughly to the table to the left of the marr terminal.

When finished, be sure to turn off the

- VPL GloveBox

- Yamaha DX100 synthesizer
- external speakers (if used)

and unplug the power cord on the Hinton MIDIC.

B Bibliography

References

[Bol84] Richard A. Bolt. *The Human Interface.* Lifetime Learning Publications, 1984.

[CDM88] Michael Cohen, Rob Duisberg, and John Maloney. *airBaton.* submitted to aaai, '88, Department of Computer Science, University of Washington, FR-35, Department of Computer Science; University of Washington; Seattle, WA 98195, 1988.

[Coh87] Michael Cohen. Stereotelephonics. Internal Memorandum IM-000-21460-87-04, Bell Communications Research, 1987.

[Coh88] Michael Cohen. Holoistics. Winner, Honeywell Futurist Competition, April 1988.

[Con86] Conrac Corporation. *User's Guide, Model 7241, 19" RBG Color Monitor*, IB-106658-999U, Y8605, Issue 1 edition, August 1986.

[Fol87] James D. Foley. Interfaces for advanced computing. *Scientific American*, 257(4):126–135, October 1987.

[Hin] Hinton Instruments, 168 Abingdon Road; Oxford, OX1 4RA; England. *MIDIC*, 1.1 edition.

[Iko82] Ikonas Graphics System, Inc.; a subsidiary of Adage, Inc., One Fortune Dr; Billerica, MA 01821. *RDS 3000 User's Guide*, 10-301-095-10a edition, 1982.

[Pol87] Polhemus Navigation Science Division, McDonnell Douglas Electronic Company, P.O. Box 560; Hercules Dr; Colchester, VT 05446. *3SPACE ISOTRAK User's Manual*, May 1987.

[Ste85] Stereographics Corporation, P.O. Box 2309; San Rafael, CA 94912. *3Display, Handbook for Stereoscopic Computer Graphics*, 1985.

[Sur63] The Surfaris. Wipeout. Dot Records, April 1963.

[VPL] VPL (Visual Programming Language) Research, Inc., 656 Bair Island Rd.; Suite 304; Redwood City, CA 94063. *DataGlove Model 2 Operating Manual.*

[VPL87] VPL (Visual Programming Language) Research, Inc., 656 Bair Island Rd.; Suite 304; Redwood City, CA 94063. *DataGlove Model 2 Test and Calibration Software Operating Manual*, September 1987.

[Yam] Yamaha, Nippon Gakki Co., Ltd.; Hamamatsu, Japan. *DX100 Owner's Manual*, OMD-148M-1, 86 06 6.5 edition.