Gesture Modelling

Using Video to Capture Freehand Modeling Commands

Mark D. Gross and Ariel J. Kemp
Design Machine Group, Department of Architecture, University of Washington

Key words: three-dimensional interface, video, gesture

Abstract: Desktop video combined with gesture recognition can be used to build powerful and easy to use interfaces for three dimensional modelling. We have built a demonstration prototype of such a system. The paper describes our video capture and gesture recognition scheme and illustrates its use in some simple examples.

1. INTRODUCTION

The Gesture Modelling project aims to provide a design environment for generating, editing, and viewing three dimensional computer graphics models using freehand gestures in space. We wish to enable designers to sketch models in the air, without the restrictions of traditional model-building, traditional sculpture, or conventional CAD modelling, but with the advantages that each of these methods have to offer.

Designers of three-dimensional artefacts (architects, but also mechanical, civil, and industrial engineers) frequently work with models constructed using Computer-Aided Design (CAD) programs. Most CAD programs use a Window-Indicator-Menu-Pointer (WIMP) interface to control the creation, viewing and modification of 3D forms. The designer views and operates on the artefact in orthographic, isometric, and perspective projections. Although designers have become accustomed to this way of working, which derives from traditional paper-based practice, working on a three-dimensional artefact through a two-dimensional interface has some limitations. That is why, in traditional practice, designers often develop a physical working model in addition to two-dimensional projections. The designer can add and remove pieces to the physical model, point to certain elements or identify directions and dimensions in three-space. These operations are clumsier in two dimensions, even with an isometric or perspective projection.

For these reasons we seek to develop interface techniques for interacting with three dimensional design models that transcend the two-dimensional plane. In working with a physical model designers use three dimensional hand gestures to point out features, change the orientation of the model, add and remove model elements. The Gesture Modelling project trades on designers' experience with three-dimensional gesture, aiming to extend the kinds of operations that are possible in reference to a physical model to take advantage of the capabilities of three dimensional computer graphics. In short, we would like to support informal creation, viewing, and manipulation of three dimensional designs using hand gestures.

Using inexpensive desktop video to obtain three dimensional co-ordinates of the designer's hands, Gesture Modelling enables the designer to trace forms in space by directly manipulating three dimensional images. The designer maintains a sense of three dimensionality by interacting spatially with the computer graphics model. The paper describes our first steps in constructing a demonstration prototype Gesture Modelling system, written in C using Microsoft's Direct3D API. We outline our current physical arrangement of cameras and screen, the image processing routines used to identify hand gestures, and the mapping between hand gesture and modelling operations that Gesture Modelling uses.

2. RELATED WORK

Early systems for capturing hand gestures, including "Put That There" (Bolt, 1980), (Foley, 1987) depended on instrumented gloves. For example, VPL's Dataglove (Zimmerman, Lanier, Blanchard et al., 1987) measured finger bending using flex sensors and hand position using Polhemus position sensors.

Interacting with computer-aided design systems through gesture has been an attractive idea for some time. Fifteen years ago, the Maestro project (Nemeth, 1984) proposed a CAD work station that employed gesture as well as natural language to mediate the process of designing. Maestro also proposed a language of gestures for CAD modeling, including gestures for mimicking organic and manufacturing processes, manipulating and shaping forms, and defining space. Since then, several researchers in Computer Aided Architectural Design have experimented with using VR hardware to develop immersive design environments (Donath, 1999; Pratini, 1999; Regenbrecht, 2000)

While sensing gloves are becoming lighter weight, less expensive, and wireless, the advent of inexpensive desktop video and advances in machine vision techniques now make possible a new class of uninstrumented solutions to sensing gestures. A good, if somewhat dated, review of efforts to sense gesture using video can be found in (Huang and Pavlovic, 1995).

The approach we follow is similar to that of Segen and Kumar(Segen, 1993; Segen, 1998) and Bimber (Bimber, 1999).

3. DESKTOP CONFIGURATIONS

3.1 The gesture space and the model space

Although the combination desktop VR display and video input allows for an informal and direct interaction with the three dimensional model space, it has some drawbacks. One drawback is that when the input model space (where the designer gestures) is directly in front of the display, then the designer's hands are always in front of the 3D display. If the display is a CRT or flat panel, then the designer's hands obscure parts of the displayed 3D model; if the display is projected on a flat surface, then parts of the model are projected on the designer's hands, which cast shadows on the flat surface.

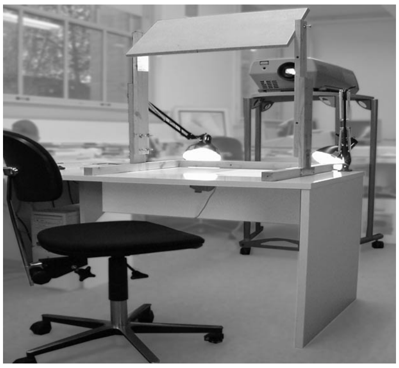

Figure 1. One of our Gesture Modelling configurations: camera and gesture space beneath the desk, with lights, projector and mirror displaying the model image on the desk top.

To address this drawback, in our current system we separate the input space and the model space. In one configuration, (figure 1), a perspective view of the 3D model is projected down onto the desk surface. A video camera is attached to the bottom surface of the desk top looking down, and the designer gestures beneath the table. The display projected on the desktop includes both the image of the designer’s hand(s) and the virtual computer graphic model that the designer is working on (figure 2). In this way, the designer’s hands can be partly obscured by parts of the model.

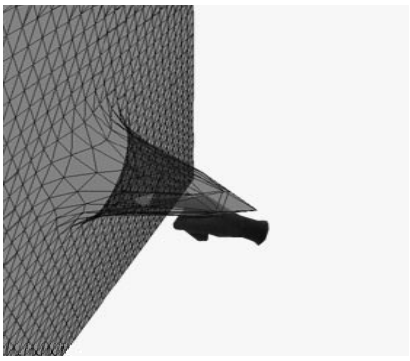

Figure 2. The display of the model space composed with the designer's hand.

3.2 Calculating depth with a single camera

Using a single camera, the system can estimate the z-co-ordinate (position in depth) of the designer’s hand. The farther the hand is from the camera, the smaller the hand appears. By counting pixels in the hand profile, it is possible to obtain acceptable z-co-ordinate data. Figure 3 shows this technique. However, the number of pixels in the hand image depends on the orientation and configuration of the hand, not just its distance from the camera. Refinement of the depth-estimation algorithm would have to take these factors into account. If the frame capture rate is fast enough, we can use knowledge about the hand’s most recent position and configuration to constrain the depth-estimate. Matching the hand profile against stored images of known configurations may also be useful in getting accurate depth estimates.

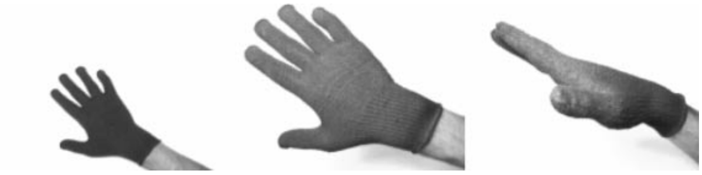

Figure 3. We can estimate the hand's depth position by counting pixels in the profile image

We also explored using a second camera, positioned so that it views the model space orthogonal to the first camera (Figure 4). Both cameras are inexpensive desktop models that communicate with the computer via USB.

A second video stream of the model space would help resolve depth ambiguity, as well as provide additional shape information about the hand configuration. However, accessing two USB cameras through the Microsoft's Direct3D libraries under Windows 98 and Windows 2000 caused unreasonably slow frame rates. Rather than debug this, we chose to postpone exploring the two-camera approach for a later phase of the project.

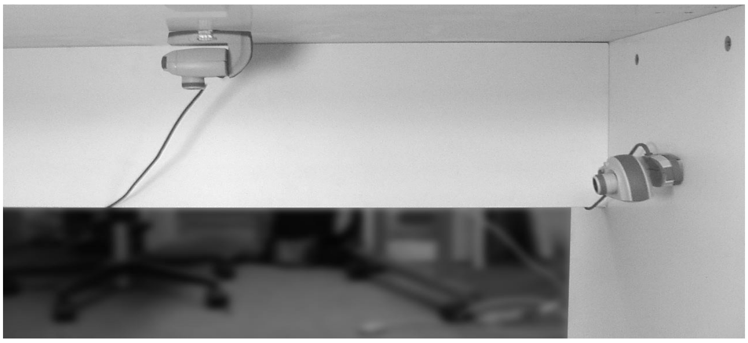

Figure 4. A second camera positioned orthogonal to the first

A problem with the configuration in figure 1 is that the designer must make hand gestures lower than is comfortable (around knee-height in a sitting position) to get the hand far enough away from the camera. To address this problem we experimented with using a mirror or two to extend the optical path, allowing the camera to be placed in a different location, for example, looking up at a mirror mounted on the bottom surface of the desktop. However, it is difficult to find positions for the camera and mirror such that the camera sees only the reflected image of the hand; that is, the hand does not come between the camera and the mirror, nor does the camera see both the hand and its mirror image. We tried mounting the camera on the floor, looking up at the gesture space. Although this resolves the distance-from-the-camera problem, it introduces two new difficulties because the camera sees the hands from the opposite side than the designer. Mounting the camera on the floor inverts the relation between the hand's z-co-ordinate and the distance-from-camera; the pixel-count method of depth estimating must be modified to account for this inversion. A more serious problem is that the camera sees the back of the designer's gesture while the designer would expect to see it from the front. This might be overcome by recognising the gesture profile and replacing the image inserted into the model with a stored view from above, or perhaps simply by flipping the image.

4. GESTURE RECOGNITION

Our approach uses pattern recognition techniques to identify the designer’s hand configuration. The number of hand configurations is limited, so it may not be difficult to distinguish them even from a relatively low-resolution image. Also, the hum1an hand can only move through possible gestures in certain ways; which constrains the recognition problem.

Wearing a black glove the designer gestures between the camera and a white background. The physical scene is lit by a diffuse light source (a luxo lamp shaded with white paper) located behind the camera, as well as a diffuse room light (a standard fluorescent light) located above the white background. This diffuse light source reduces shadows cast by the first light, which make the vision task more difficult.

The vision task consists of extracting from the image the following information:

a) the gesture of the hand, matching an item of a set of known gestures, or a ‘none’ output indicating no match is satisfactory
b) spatial information that will serve to generate forms, such as fingertip positions.

As the system is presently implemented the user’s palm must remain oriented within a plane normal to the optical axis of the camera. Extended fingers must also lie along this plane.

4.1 Computation on input images

Input from the camera consists of a 160 by 120 RGB bitmap image (figure 5a). This is thresholded at a given intensity level, currently hard-coded in the system. (In the next phase of the project, we will compute the threshold intensity level from the first flat region after the primary peak of the intensity histogram.) The primary peak corresponds to the dark, ‘glove’ pixels. The histogram and the threshold level is computed only for the first frame of the sequence. The same threshold level is used for the remainder of the frames in the session. This eliminates unnecessary per-frame computation that would slow down the system. We may add to this a strategy to eliminate darker skin tones, which can become part of this primary histogram peak. Alternatively, the black glove can be worn with a bright long sleeve.

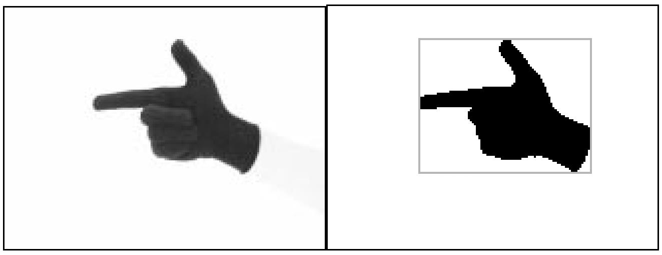

Figure 5 (a) raw camera image; (b) thresholded image with bounding box.

Pixels that are less than the threshold value are replaced by white (maximum intensity) pixels; those that are greater are replaced by black (minimum intensity) pixels. The result of this thresholding is the black silhouette of the gloved hand against a white background. A bounding box is obtained inscribing all the black pixels in the image (figure 5b). We aim to recognise the largest single continuous black object (blob) in the image. Only the region inside the bounding box is used for the remainder of computation.

Next, we compare the image inside the bounding box with a set of pre-stored low-resolution images, or templates. The problem of matching a template to an image is well-researched; however, a novel, computationally light approach is introduced here since the task is quite specific—recognising a *configuration* of a single known object that is guaranteed to reside within a given bounding box.

This approach consists of producing templates that are spatial maps of image features that can be compared pixel by pixel with the box-bounded camera image for fast per-frame performance. This contrasts with algorithms that extract features such as edges, segments, or lines from the unknown (camera) image, processes that would slow down the per-frame rate substantially. The templates are currently produced by a human operator. However, future work will focus on generating these templates automatically during a 'training' or 'calibration' period. This is important because the templates are produced from a series of real camera images of the a single user's hand; the system currently performs best when operated by that user. As the calibration need not perform in real time, we may use more sophisticated, computationally intensive methods to pre-compute the templates.

4.2 Template images

The template images and the matching strategy are next described. A good match with one of the template images constitutes a recognised gesture; the template has attached additional spatial information used in form synthesis.

Each template image stores information that serves to differentiate one gesture from another. An example of such a template is shown in figure 6.

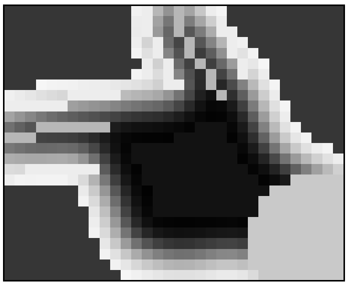

Figure 6. A template image for a 'gun' gesture. The image shown here is magnified

This template image was created by hand from a camera image manipulated with the Photoshop program as follows: The raw camera image is thresholded and cropped at a box bounding the gloved hand pixels. The resolution is decreased using bicubic resampling (presently to a size in which the greatest dimension is 32 pixels) and subsequently a Gaussian blur is applied with a mask of radius 1.5 pixels. Areas that remain white are replaced by red pixels. Areas that remain black within the palm are replaced by blue pixels. A region of grey pixels remains unmodified. Next, lines are drawn that indicate finger positions. As fingers are stick-like, they appear as black linear regions surrounded by the white background. Pixels in the centre of these black linear regions are indicated in green; the white regions around them are indicated in yellow. A composite of some templates stored by the system is shown in Figure 7. Notice that multiple rotations of the same gesture are represented. This eliminates unnecessary per-frame computation (and loss of accuracy due to re- or sub-sampling) that would be required for run-time rotation.

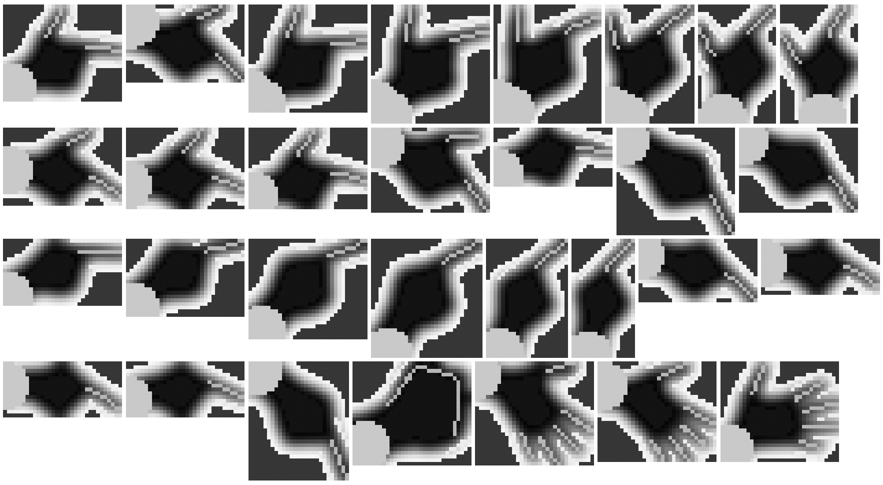

Figure 7. A composite of some templates stored by the system

4.3 Matching the templates:

The running program now compares each template image (**T**) to the unknown image. The unknown image (**U**) consists of the pixels contained within the bounding box. A scoring system determines the best match. First, the aspect ratios of **U** and **T** are compared. If they differ by more than a constant amount, **T** is 'discarded' and cannot be considered a match. (The next template image is considered, starting over the comparison process). Next, **U'** is obtained by subsampling **U** to match the dimensions of **T**. (This is not done explicitly; rather, pixels in **U** are accessed as if **U'** were subsampled). Red pixels in **T** are compared with the corresponding pixels in **U'**. If too few (a fixed fraction) of the corresponding pixels are white in **U'**, **T** is not a match. Then the matching score is incremented by an amount proportional to the percentage of pixels that do match. A similar process compares blue pixels in **T** with black pixels in **U'**. Then, green pixels in **T** are checked against the corresponding *un-subsampled* pixel in **U** and its eight neighbors. The score is incremented by an amount proportional to the number of pixels out of these sets of 9 that are black. A similar process is done for the green pixels in **T** and white 9-neighbors. Grey pixels in **T** are compared to pixels in **U**, and the score is incremented by how close the intensity values are. The grey area thus permits variations in the two images that can still produce a good match. Turquoise pixels in **T** aren't matched against pixels in **U** or **U'** because the system shouldn't compare wrist angles. Finally, the final score is thresholded to eliminate weak matches, and the top-ranked 3 templates are noted. From this list, the system selects the gesture that matches the largest number of the last several frames' top-ranked gestures. This avoids gesture 'flicker' in case of a wrong match. An equivalent strategy would be to change the reported gesture only after it has been top- or near-top- ranked for *n* sequential frames.

4.4 Recognition performance

Shown below are images of the recogniser's 'vision' window. The coloured pixels represent the spatially corresponding, with respect to bounding boxes, matched template image. The program is run using Windows 2000 and a USB personal desktop camera:

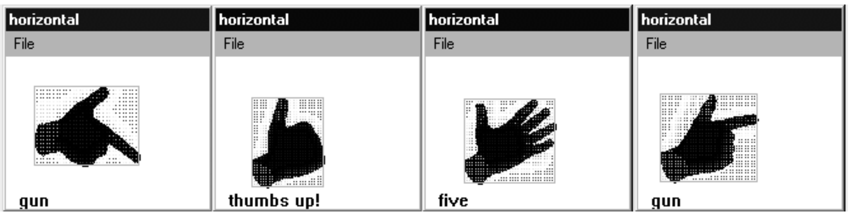

Figure 8 from left: 'gun' gesture, thumbs up!; 'five' and 'gun' with imperfect match

We have not subjected the system to rigorous quantitative testing, but the demonstration applications we have built perform well in real time. The system has been trained to recognise seven gestures: point, pinch, gun, thumbs up, fist, five (open palm), two (v-symbol). The system is fairly robust, as can be seen from figures 8-c and -d, where the template image and the camera image do not match as directly as they could, yet the proper gesture is reported. A benefit of matching the template image to the bounding-box of the gloved hand is that spatial information (fingertip position, etc) used to create forms is continuous, even though template images are not (for example, template images differ by rotations of 10 degrees).

Most gestures are recognised correctly, with the correct orientation and few non-gestures are reported as matches. In a test of the terrain modeling application (see below), for example, five users who had no experience with the system were immediately able to make recognized gestures. However, this system only employs three distinct gestures: (five, point, and fist). The biggest problem using the system is extraneous pixels from the user's sleeve or other parts of the body entering the camera's field of view, which confuses the recognizer.

Recognition templates are prepared in Photoshop and loaded into the system. The Gesture Modeling environment cannot be trained or customized for a different user, nor can new gestures be added without editing and recompiling code. It could be extended to train the system interactively to recognize new gestures. This would require building the image processing functions described in section 4.2 (Gaussian blur, bicubic resampling, clipping). Though not conceptually difficult, this remains a project for future work, as does the interactive binding of gestures to specific application commands.

5. APPLICATIONS

5.1 Form Generation

The first version of the Gesture modelling system generates a series of points in the model space as the designer traces a line in the gesture space. The points are represented in the model space by tiny tetrahedra, so that they are visible from any viewpoint (Figure 9). In this first version, the first dark pixel found in the video frame (scanning from left to right and top to bottom) was used to identify the (x y) co-ordinates for the new point. If the designer uses his/her index finger as a “brush” and is careful not to rotate his or her hand, this simple method of getting the co-ordinates works well. However, the designer must orient his or her hand so that the index finger is always the first thing the system sees in this scan pattern. We find this constraint unacceptable.

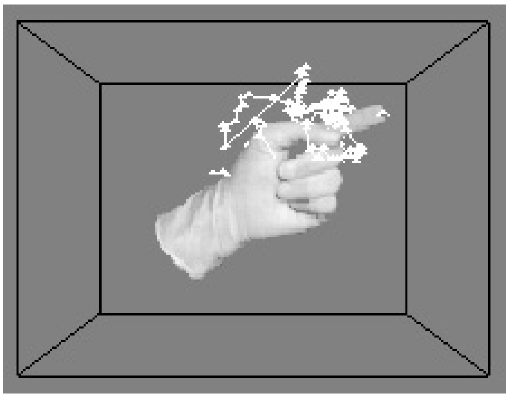

Figure 9. The system leaves small tetrahedron at the fingertip (first pixel scanning from left, top) using hand size to determine a Z co-ordinate

We considered painting the tip of the designer’s index finger (or a latex glove) to find the red pixels in the image, and using this to specify the (x y) co-ordinates of the insertion point. We did not complete implementing this approach. Instead, we began to work on recognising the designer’s hand gestures, which would offer a more general and less constrained interaction.

5.2 Mesh Distortion

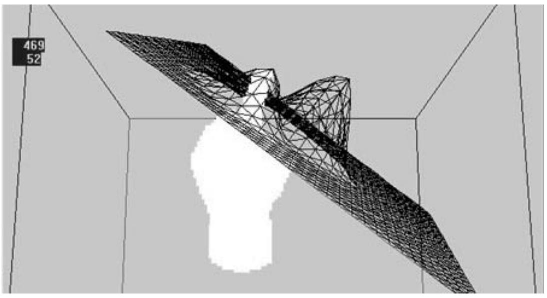

Figure 10. Creating a landform by deforming a mesh

Figure 10 shows how, using the ‘point’ hand gesture, the designer can deform a mesh to create a land form. The numbers at the left of the screen indicate the co-ordinates of the hand’s position.

5.3 Selecting and moving objects

Figure 11 shows how, once a land form mesh has been created, the user can place objects on the mesh surface and on top of one another. Two rectangular solids have been placed on top of a hill in the land form. The designer is using the ‘fist’ gesture to grasp and carry one of the objects. In the next version of the software, we plan to run a simple hydrology model to simulate water runoff on this landform.

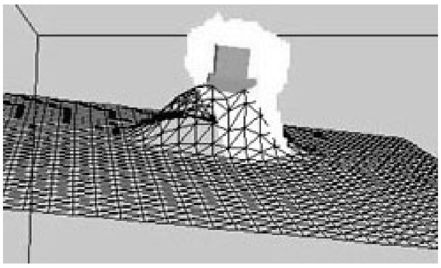

Figure 11. Selecting and moving objects

Figure 12 shows how, using the ‘open palm’ gesture, the designer selects one among a set of blocks. As the designer’s hand passes over each block, it highlights in red to indicate that it is selected. If the designer then forms a fist, the selected object is grasped and can then be moved in three dimensions.

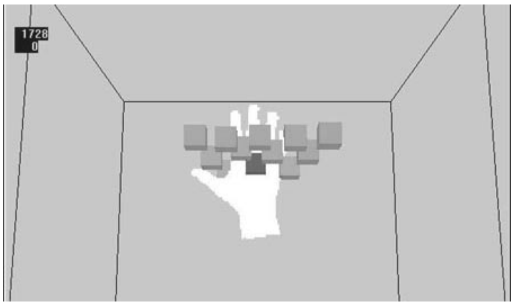

Figure 12. Selecting blocks to arrange them in a massing model

Figure 13 shows how the designer can make an arrangement of blocks, by grasping them as above and then placing them in three-space.

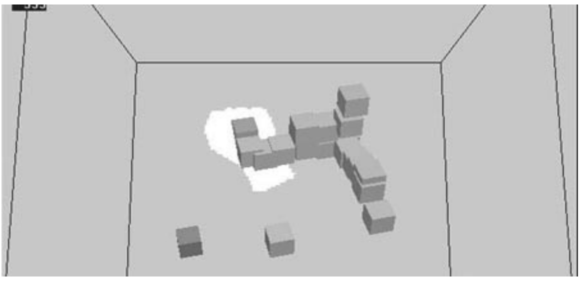

Figure 13. Creating a massing model by assembling blocks

6. FUTURE WORK

The present Gesture Modelling program is little more than a proof of concept demonstration and a platform for experimentation. We have used it to create several small demonstration systems, in which particular gestures are bound to particular operations. For example, in the terrain modeler, ‘point’ deforms the mesh; ‘five’ selects a block, and ‘fist’ moves a selected block for placement. However, these are specific examples, and we have not used the Gesture Modelling program to build a larger system, nor have we tested the system’s recognition more than a handful of distinct gestures. We have also not explored the use of dynamic gestures. These are immediate directions for further work.

7. ACKNOWLEDGEMENTS

We would like to thank the anonymous CAAD Futures reviewers for their helpful suggestions. This research was supported in part by the US National Science Foundation under Grant No. IIS-96-19856/IIS-00-96138. The views contained in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

Bimber, O., 1999, "Rudiments for a 3D freehand sketch based human-computer interface for immersive virtual environments", In *VRST 99*, ACM, London, p. 182-183.

Bolt, R., 1980, "`Put That There': Voice and Gesture at the Graphics Interface." *Computer Graphics* 14(3), p. 262-270.

Donath, D., 1999, "Using Immersive Virtual Reality Systems for Spatial Design in Architecture", In *AVOCAAD Second International Conference [AVOCAAD Conference Proceedings*, Brussels (Belgium), p. 307-318.

Foley, J. D., 1987, "Interfaces for Advanced Computing"." *Scientific American* 257(4 October 1987.), p. 126-135.

Huang, T. and Pavlovic, 1995, "Hand Gesture Modeling, Analysis and Synthesis", In *Proc. International Conference on Automatic Face and Gesture Recognition*, p. 73-79.

Nemeth, C., 1984, *Maestro: A Gesture Recognition System.* Institute of Design, Chicago, Illinois Institute of Technology.

Pratini, E., 1999, "Esboçando com Gestos: O Projeto 3D SketchMaker", In *III Congreso Iberoamericano de Grafico Digital [SIGRADI Conference Proceedings]*, Montevideo (Uruguay), p. 141-144.

Regenbrecht, H., Kruijff, E., Donath, D., Seichter, H. and Beetz, J., 2000, "VRAM - A Virtual Reality Aided Modeller", In *Promise and Reality: State of the Art versus State of Practice in Computing for the Design and Planning Process [18th eCAADe Conference Proceedings*, Weimar (Germany), p. 235-237.

Segen, J., 1993, "Controlling Computers with Gloveless Gestures", In *Proceedings Virtual Reality Systems*, New York City, p. 1993.

Segen, J., 1998, "Gesture VR: gesture interface to spatial reality", In *International Conference on Computer Graphics and Interactive Techniques; abstracts and applications*, ACM SIGGRAPH, p. 130.

Zimmerman, T. G., J. Lanier, C. Blanchard, et al., 1987, "A hand gesture interface", In *CHI+GI*, ACM, Toronto, p. 189-194.