

EVALUATION OF AN INTERACTIVE SITE MODELING SYSTEM

by

Frederick W. Hood

ARTHUR LAKES LIBRARY  
COLORADO SCHOOL OF MINES  
GOLDEN, CO 80401

ProQuest Number: 10794276

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10794276

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

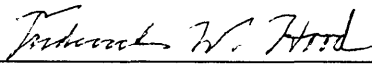
This work is protected against unauthorized copying under Title 17, United States Code  
Microform Edition © ProQuest LLC.


ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 – 1346

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of *Master of Science* (*Mathematical and Computer Sciences*).

Golden, Colorado

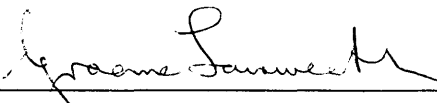
Date 4/4/97

Signed:   
Frederick W. Hood

Approved:   
Dr. William A. Hoff  
Thesis Advisor

Golden, Colorado

Date 4/4/97

  
Dr. Graeme Fairweather  
Professor and Head,  
Department of Mathematical and Computer  
Sciences

## ABSTRACT

Hazardous and/or unstructured environments often require the use of robots. The efficiency with which high level robotic operations are conducted in these environments can be improved with the help of graphical site models. Methods for creating these models have historically fallen into two categories: manual modeling or fully automatic (autonomous) modeling. Manual methods are generally slow but reliable, while typical autonomous systems are fast but lack reliability. However, an interactive system for creating site models (*i.e.*, one that has both manually-driven and autonomous components) can combine the speed of autonomous techniques with the reliability of manually-driven methods. Our **S**ystem for **I**nteractive **M**odeling via **O**ptimization (SIMON) takes 3-D range points from a trinocular stereo vision system as input. SIMON uses an optimization technique known as simulated annealing in conjunction with a supervisory control measure called *traded* control to do the site modeling. The result is a system that fits graphical models to range data quickly and accurately. To demonstrate the performance of SIMON, we have designed and executed experiments using 14 human subjects. The experiments were designed to test the following hypotheses:



- 1) The performance of interactive modeling is not significantly better than manual modeling in terms of total task time and fitting accuracy.
- 2) Operator effort does not decrease significantly when utilizing interactive modeling.
- 3) User expertise has a significant effect on interactive modeling task time.
- 4) Initial placement of the primitive object model by the human supervisor will not improve performance significantly when interactively modeling a “difficult” scene, nor will minimal human interaction (no initial placement) significantly increase performance on “easy” scenes.

Our results refuted all of the above with the exception of the first part of Hypothesis 4.

## TABLE OF CONTENTS

<b>1. INTRODUCTION .....</b>	<b>ix</b>
<b>2. REVIEW OF BACKGROUND LITERATURE.....</b>	<b>8</b>
2.1. MANUAL SOLUTIONS.....	8
2.2. AUTONOMOUS SOLUTIONS.....	10
2.3. INTERACTIVE SOLUTIONS.....	13
2.4. OPTIMIZATION.....	15
<b>3. SYSTEM DESCRIPTION .....</b>	<b>18</b>
3.1. STEREO VISION SENSOR .....	20
3.2. AUTOMATED OPTIMIZATION/MODEL FITTING .....	24
3.2.1. Optimization by simulated annealing .....	25
3.2.2. The Model Fitting GUI .....	30
3.3. MANUAL COMPONENT (TRADED CONTROL) .....	36
<b>4. EXPERIMENTS.....</b>	<b>38</b>
4.1. PRELIMINARY EXPERIMENTS.....	39
4.2. FORMAL EXPERIMENTS .....	42
4.2.1. Discussion of hypotheses .....	43
4.2.2. Experimental design .....	46
4.2.3. Procedure/Execution .....	48
4.2.4. Results.....	54

**5. CONCLUSIONS AND FUTURE WORK ..... 60**

## LIST OF FIGURES

FIGURE 1: THE MAJOR COMPONENTS OF OUR SITE MODELING SYSTEM.....	19
FIGURE 2: TRINOCULAR STEREO CAMERA ARRAY MOUNTED ON GANTRY ROBOT END EFFECTOR.....	21
FIGURE 3: THE STEREO VISION "SCENARIO" (ADAPTED FROM <i>MACHINE VISION</i> <sup>18</sup> ).....	22
FIGURE 4: IMAGES (LEFT, CENTER, RIGHT) FROM STEREO VISION SYSTEM.....	23
FIGURE 5: FINAL RANGE POINTS GATHERED.....	23
FIGURE 6: VARYING ORIENTATIONS FOR CUBE (ONE, TWO, AND THREE VISIBLE FACES) AND CYLINDER (SIDE, SIDE AND CAP, AND CAP ONLY VISIBLE) MODELS .....	29
FIGURE 7: THE APPLICATION GUI .....	31
FIGURE 8: AN EXAMPLE OF A SCENE GRAPH (ADAPTED FROM <i>THE INVENTOR MENTOR</i> <sup>26</sup> ) .....	32
FIGURE 9: INITIAL POSITION OF THE MODEL.....	33
FIGURE 10: FINAL POSITION OF MODEL AFTER FITTING.....	33
FIGURE 11: THE MODEL-FITTING APPLICATION CONTROL PANELS.....	34
FIGURE 12: SCENES USED FOR PRELIMINARY EXPERIMENT (SEE TABLE 2 FOR ORDER) .....	40
FIGURE 13: SCENES UTILIZED IN THE FORMAL EXPERIMENTS L TO R FROM TOP: SYN1, SYN11, SYN2, AND SYN10 (EASY); SYN3, SYN9, SYN4, AND SYN12 (DIFFICULT) .....	50
FIGURE 14: INDIVIDUAL SUBJECT EVALUATION FORM.....	52
FIGURE 15: SHEET CONTAINING FORMAL EVALUATION GUIDELINES.....	53
FIGURE 16: COMPARISON OF MEAN TASK TIMES FOR EACH SCENE (ERROR BARS REPRESENT STANDARD DEVIATIONS) .....	58
FIGURE 17: A COMPLETED SITE MODEL.....	61
FIGURE 18: AN EXAMPLE OF HOW INCOMPLETE DATA CAN SKEW A MODEL'S FIT .....	63

## LIST OF TABLES

TABLE 1: SAMPLE STEREO MATCHING STATISTICS (FROM PRELIMINARY EXPERIMENTS) .....	24
TABLE 2: ORDER IN WHICH THE SCENES WERE INTRODUCED TO THE SUBJECTS (ORDER REPEATED DURING SECOND HALF OF TRIAL WITH OPPOSITE FITTING METHODS) .....	41
TABLE 3: SUMMARY OF PRELIMINARY EVALUATION RESULTS.....	42
TABLE 4: INDEPENDENT EXPERIMENTAL VARIABLES.....	46
TABLE 5: SCHEDULE OF SCENES ENCOUNTERED DURING EXPERIMENTS.....	48
TABLE 6: RESULTS OF ANOVA .....	56
TABLE 7: MEAN EXPERIMENTAL RESULTS FOR THE INDIVIDUAL TEST SUBJECTS.....	56
TABLE 8: MEAN EXPERIMENTAL RESULTS FOR ALL SCENES.....	57
TABLE 9: INTERACTIVE TASK TIMES FOR VARYING USER EXPERTISE.....	57
TABLE 10: INTERACTIVE TASK TIMES USING VARIOUS FITTING STRATEGIES.....	57

## ACKNOWLEDGMENTS

I would like to thank Dr. William A. Hoff for serving as my advisor, for hiring me, and for offering guidance and support throughout. I would also like to thank the following people for their contributions to my work:

Thesis committee members: Dr. Robin Murphy and Dr. Steve Pruess.

Project team members: Dr. William Hoff, Dr. Robert King, Tory Lyon, Khoi Nguyen, Lin Xia, Doug Swartzendruber, and Rex Rideout.

Volunteer test subjects: Scott Walker, Dave Hershberger, Hilda Layne, Myron Smith, Charles Farris, Joe Dvorak, Dan Morganto, Matt D'Amore, John Markus, Eric Northcut, Glenn Blauvelt, Jake Sprouse, Jodi Noone, Lin Xia, Khoi Nguyen, Chris Colborn, and Aaron Gage.

## 1. INTRODUCTION

The use of robots to do a variety of tasks is becoming more common in industrial settings. In particular, the use of robots for operations in hazardous environments is, in many cases, a necessity because humans are unfit to perform such operations due to health risks or physical limitations.

The efficiency of robot operations can be improved by making common operations such as the avoidance or manipulation of objects more autonomous. In order to enable autonomous operations, robots require 3-D data. Specifically, the location, orientation, shape, size, and other vital information pertaining to objects in an environment or *scene* are needed to autonomously avoid or manipulate objects. The acquisition of this 3-D data is the subject of this thesis.

Determination and representation of 3-D object information is what can be referred to as the *site modeling* problem<sup>1</sup>. Site models contain the vital object information necessary for a particular robotic application. Site modeling is the creation of a graphical representation of 3-D objects in a scene (a *site model*). As part of this process, *pose estimation* must be performed. Pose estimation refers to the process of determining the

location and orientation of objects. Our research to this point has addressed the site modeling problem in order to arrive at an effective solution.

However, site modeling and pose estimation are not trivial problems for machines. Humans, on the other hand, can view images and separate objects from the background much more efficiently than machines.

Many techniques have been developed in an attempt to solve the site modeling problem. In the past, site models were constructed from explicit measurements or blueprints using CAD applications. Recently, the emergence of machine vision techniques for gathering range, distance, and image data has provided an accurate alternative to manual measurement. These techniques include stereo vision and structured light systems and have resulted in more efficient site modeling methods.

Historically speaking, site modeling methods have fallen into one of two categories: manually-driven modeling techniques and autonomous methods. Manual site modeling methods are, in general, very reliable since the user can employ specific knowledge about the scene when placing or building object models. These systems usually consist of a graphical user interface that allows the user to fit models of various objects to range data or background images<sup>2</sup>. Although reliable, these manually-driven methods suffer from a lack of speed and too much effort on the user's part<sup>1</sup>.

Autonomous modeling methods have been designed to overcome the problems with manually-driven methods. Since they are implemented using computerized algorithms,



these methods tend to find solutions quickly and require little or no human intervention. Examples of autonomous modeling systems include: a system that fit an MRI model to laser range data<sup>3</sup>; a technique that used dense range images to form a discrete occupancy map of an environment<sup>4</sup>; and a system that created 3-D surface meshes from range points and estimated the corresponding cylindrical models that most closely fit the surfaces<sup>5</sup>. However, autonomous methods suffer from unreliability (*i.e.*, do not perform correctly when completely unsupervised), particularly when operating in unstructured environments<sup>1</sup>. This unreliability helps to support the notion that site modeling is a difficult problem in computer vision.

To minimize the problems with speed and/or reliability facing existing site modeling systems, researchers have moved toward developing *interactive* solutions. Interactive site modeling systems consist of both an autonomous component and manual supervision. This combination of manual and autonomous components (if done correctly) results in systems that operate efficiently and accurately. An example of the growing presence of interactive technologies is ARPA's ongoing Research and Development for Image Understanding Systems (RADIUS)<sup>6</sup> project in which aerial images of buildings are used in automated and semi-automated systems to create site models.

The research question facing us was, "Can we utilize interactive techniques to improve site modeling performance and maintain reliability?" This question, in turn, raised several technical issues concerning our approach to site modeling. The first issue is the

type of data that we use to either fit models to or build models from. For example, the data can be sparse or dense, 2-D or 3-D. Differing data types necessitate the use of different algorithms. A second issue is the choice of an automated method for use in site modeling. Automated techniques for building and fitting models can improve the efficiency of such techniques compared to manually-driven methods. Another issue is the means by which we would implement interaction. Human interaction is necessary in an interactive system and can improve reliability compared to purely automated systems. Object model representation comprises yet another issue. Several alternatives exist and can influence the difficulty of the problem.

The potential advantages of interactive systems and the research issues we raised led us to create a **S**ystem for **I**nteractive **M**odeling via **O**ptimization (**SIMON**). Our system consists of three major subsystems: a stereo vision sensor, an automated optimization routine, and a supervisory control measure. The stereo vision subsystem addresses the need for data. Stereo vision is the source of sparse (from 50-150 “points” per image) 3-D range data and was selected primarily because it is inexpensive. However, we have tested SIMON successfully using data from a laser rangefinder. The issue of which autonomous method to employ is handled by our choice of *simulated annealing*. Optimization by simulated annealing was selected for its ability to escape local minima when searching an error space (a crucial aspect in our problem environment). The choice to use simulated annealing over methods such as least squares and genetic algorithms also revolved around

its availability, our familiarity with the algorithm, and because it was relatively easy to code. Finally, a supervisory control measure called *traded control*<sup>7</sup> addresses the issue of how to enable human interaction. In particular, traded control was incorporated to allow human intervention when the automated portion fails. The optimization and supervisory control portions co-exist in a software application written for Silicon Graphics workstations which takes range data as input and allows the interactive creation of site models.

By designing SIMON in this way, we, in essence, made several simplifying assumptions which facilitated SIMON's creation of site models. First of all, we assumed that objects in a scene were rigid and were comprised of "primitives", or simple objects such as cubes, cylinders, spheres, and cones, that were of known size. This choice addresses the issue of object representation -- an important point in computer vision problems since objects can also assume parametric, superquadric, and depth map representations (among others). The choice of simple geometric primitives also allowed the user to choose the model and associated dimensions from a database of such objects, thus eliminating the problems of segmentation and sizing of objects. Primitive object models also simplified distance calculations from range data to the object surface. For each primitive, an associated "closest point" formula was easy to produce because of the regular geometry of a primitive. Occluded objects could also be modeled using this approach. Another assumption was that differing densities of range data would be

handled in the same way. Therefore, any set of range points expressed in terms of their (x, y, z) coordinates in meters from a camera-centered origin could be utilized. This convention makes the use of any range source (e.g. stereo vision, structured light, etc.) viable.

In addition to development of our interactive site modeling system, we had the following objectives:

- 1) Do performance analyses comparing SIMON to identical but purely manual and purely automated systems. The analyses should use identical scenes and data and should use human subjects where necessary. Data sets should contain outliers and scenes with occluded objects. Also, devise performance metrics (functions of time, fitting accuracy, or both) to adequately represent the results.
- 2) Determine the best fitting strategies for scenes of varying difficulties.
- 3) Determine if the benefits of SIMON are greater for novice/skilled operators and for simple/difficult scenes.

Through experiments conducted with 14 human subjects, we have demonstrated that our system completes tasks quickly and accurately for the “cube” and “cylinder” primitive model types. We compared our interactive system to an identical but purely manually-operated system by having the subjects fit models in identical scenes using both methods. The hypotheses we intended to refute were as follows:

- 1) Interactive modeling should not significantly outperform purely manual modeling in total task time and fitting accuracy.
- 2) Operator effort does not decrease significantly when utilizing interactive modeling.
- 3) User expertise has a significant effect on interactive modeling task time.
- 4) Initial placement of the primitive object model by the human supervisor will not significantly improve performance when interactively modeling a “difficult” scene, while minimal human interaction (no initial placement) will not increase performance significantly on “easy” scenes.

A preliminary evaluation also addressed the lack of reliability inherent in an identical but purely autonomous site modeling system.

The remainder of this thesis discusses literature relating to our methodology and area of application, gives a detailed description of our system, summarizes the experiments and their results, and submits conclusions and directions for future research.

## 2. REVIEW OF BACKGROUND LITERATURE

Technologies in the areas of computer vision, robotics, and optimization are relevant when discussing the site modeling problem. This review addresses a selection of works in these areas that were consulted throughout the course of this project. In particular, this review will address manually-driven solutions to the site modeling problem, autonomous solutions, interactive solutions, and optimization techniques used in such problems. The autonomous solutions are further decomposed into surface and/or map-based methods versus model-based approaches. The section on interactive solutions also discusses the supervisory control literature. Several optimization techniques will be addressed, but the concentration will be on simulated annealing.

### **2.1. Manual Solutions**

Manual solutions to the site modeling problem, as with many other systems, are reliable but time-consuming<sup>7</sup>. By *manual*, we mean that the computer provides a graphical user interface (GUI) and contributes little else, while the user performs most or all of the modeling tasks. Systems in this category typically facilitate intuitive piloting of the application through the presence of a sophisticated GUI.

For example, Mechanical Technologies Incorporated (MTI) developed a Topographical Mapping System (TMS) that used a GUI for manipulating and fitting object models to structured light range data<sup>2</sup>. SIMON was developed with a similar thought in mind: the fitting of pre-built object models to range data displayed in a GUI viewing area. Structured light data, however, is very dense (*i.e.*, comparable in order of magnitude with the number of pixels in an image) and gathering this data exacts significant time and expense relative to our stereo vision system. MTI's TMS also exacted a cost in time and effort from the user, who was ultimately responsible for choosing and placing or fitting the graphical models using the range data as a guide.

Other manual site modeling techniques have been developed in conjunction with actual robotic systems. Researchers at Sandia National Laboratories have created several Graphical Programming systems<sup>8</sup>. These systems used the IGRIP software package to display graphical models of robots and objects in scene. IGRIP also gave object and robot models the functionality necessary to perform full-fledged simulations (joint movements, gripping, collision simulation, etc.). The choice not to use IGRIP for SIMON was one of availability and lack of an immediate need for functional object models. In the Graphical Programming system, the models were predefined and placed manually within the scene, much like with SIMON. The authors recognized that such a system could benefit from a computer vision-based approach to modeling and updating like SIMON's.

The concept of *pose querying* is also important to manual modeling techniques. Pose querying refers to the measurement of range via touching an object with a sensor. Tracking this sensor's location will then return the pose. Pose querying is typically accomplished via teleoperation of a robot. The developers of the Graphical Programming system mentioned pose querying as a method of improving their system. Measuring pose through virtual manipulation of objects<sup>9</sup> and by manually placing a pattern of targets in a scene from which depth can be estimated<sup>10</sup> were variations on the idea of pose querying. Pose querying offers reliable measurements, but at the expense of time and operator effort. Manual placement of targets, too, is time-consuming as well as an unwanted health risk to humans in hazardous environments. Stereo vision, as was used in SIMON, provides a much safer and less laborious means of measuring range.

Overall, it is evident from the relative lack of recent literature on manually-driven systems that more modern research literature addresses partially and fully automated systems. This observation is indicative of the current trend in the area of site modeling systems, as well.

## **2.2. Autonomous Solutions**

Autonomous solutions to the site modeling problem attempt to overcome the speed deficiencies inherent in manually-driven methods via the utilization of computerized



algorithms. These algorithms may be designed to build graphical models from range data or features extracted from an image.

Methods which construct depth maps or surface-based representations of objects from range or image data comprise one category of autonomous modeling methods. Depth maps produced by methods such as the environment modeling done at the University of Michigan<sup>4</sup> are volumetric occupancy grids created from dense range data. Depth mapping methods are useful because they are not limited to any particular types of objects. However, methods like this are time-consuming due to the seamless set of dense range data and multiple views required to produce the maps. In contrast, SIMON produces and uses a fairly sparse set of range data, which takes little time to produce. In addition, depth mapping methods do not perform *segmentation*. Segmentation is the separation of objects in an image from the background. Segmentation is critical for efficient operations via the use of site models because it facilitates the manipulation of individual objects. SIMON does segmentation interactively by fitting simple primitive object models to the data.

Surface-based representations do more segmentation than do depth mapping methods. The point in the modeling process at which segmentation occurs, however, can vary amongst methods. For instance, one approach “grew” geometric models from seed regions in an image that were termed most likely to contain data belonging to only one object<sup>11</sup>. Unlike SIMON, which uses simulated annealing and primitive models, the

selection and placement of the geometric parametric models was accomplished using a “winner takes all” algorithm. Other approaches addressed segmentation in the latter stages of modeling only. An example was the “Artisan” system at Carnegie Mellon<sup>5</sup> which used dense scanning laser rangefinder data to produce a surface mesh, followed by construction of planar and quadric surface representations, and finally the placement of more meaningful models such as cylinders. Artisan suffered from modeling inaccuracies due to sensor error and also failed to fit occluded data effectively due to a lack of multiple views. SIMON was designed to handle occluded objects, with less data, and data from a single view. Another example was a method which employed range images and the 3-D Hough transform to extract planes from object surfaces and later determine the location and orientation of objects from these planes<sup>12</sup>. The Hough transform method only worked effectively for objects composed of planar surfaces and had difficulty with objects in which only one surface was visible. SIMON has been tested successfully with objects having curved and/or planar surface composition.

Model-based autonomous solutions to the site modeling problem more closely resemble SIMON’s approach in that pre-built models are fit to data or extracted image features. The desire for neurosurgeons to operate without a stereotactic frame attached to the patient prompted the creation of an automatic registration method<sup>3</sup>. This method fit MRI models to 3-D laser range data using a variation of least squares fitting. Another method used the iterative closest point (ICP) algorithm to fit a hierarchy of models

(including point sets, parametric curves, 3-D surfaces, and complex models built from primitives) to 3-D point data<sup>13</sup>. Both of the above methods resemble SIMON in their use of optimization and 3-D point data as well as pre-built models. However, they demonstrate little tolerance for multiple objects, occlusions, and faulty data, particularly in the form of statistical outliers.

Thus, purely autonomous methods, most of which require dense range data, tend to lack the capabilities to deal with occlusions, statistical outliers, and faulty data. It follows that such methods, by themselves, would be unreliable in unstructured environments and adverse conditions.

### **2.3. Interactive Solutions**

Interactive solutions to the site modeling problem combine the speed of autonomous methods with the reliability of manual techniques. Typically, the automated portion of an interactive system extracts image features and either builds models from these features, or fits models to these features (or to range data). The manual portion of the system allows the user to monitor and interact with the system as needed to guide its operation and correct errors.

ARPA's RADIUS project has prompted the development of many such systems, primarily using data gathered from aerial images of buildings and the surrounding terrain. One example was the SITECITY system at Carnegie Mellon<sup>14</sup>. The image understanding

techniques utilized in SITECITY included edge detection as well as the incorporation of geometric constraints. The extracted data was then used to build graphical models. The paper that communicated this work was especially relevant to our work with SIMON in that it:

- 1) demonstrated the performance and usefulness of an interactive system
- 2) evaluated the choice of automated process(es)
- 3) supported the use of an identical but purely manually-driven system for performance evaluation
- 4) identified a performance measure referred to as *user cost*.

Another RADIUS-related work discussed the addition of interaction to an automated modeling system<sup>1</sup>. Minimal human interaction was the goal of this work as well and, much like SIMON's use of traded control, was used to fine-tune the fit of a model to extracted image cues such as edges or shadows. This fine-tuning was only part of a final editing measure and did not approach full traded control.

Interactive systems, in general, have recently become more widely used in research, but the idea of supervisory control is decades old<sup>7</sup>. Methods like the above use a form of *traded* control, meaning that human or automated control of the system at any one time is exclusive. The use of traded control in SIMON has been extensively as a means of interaction, primarily for initially placing or fine-tuning the fit of an object model. *Shared* control, on the other hand, allows the human operator to control several degrees

of freedom while the automated portion controls the remaining degrees. Shared control has been implemented for SIMON, as well, but has not been tested extensively. An example of SIMON using shared control would be to allow the user to manipulate the translation of an object while automated simulated annealing attempted to fit the orientation.

With the added reliability of human interaction, interactive systems have the potential to outperform manual systems in speed and autonomous systems in reliability. The RADIUS project examples, most of which used aerial views of sites to be modeled, still seem to lack the capability to model occluded objects. However, a model-based interactive approach such as SIMON's relies on the user for segmentation. This approach eliminates the possibility that occluded objects might be perceived as being smaller than they really are.

## **2.4. Optimization**

Optimization is often used in computer vision for fine-tuning of the estimated positions and orientations of objects. The methods covered here represent but a few of the possible optimization schemes that can be adapted for vision applications.

These optimization methods can be categorized according to their formulation -- some utilize derivatives of their respective *objective functions* (i.e., the function to be minimized), while the others do not utilize the derivatives. Methods which use the

derivatives include Random Sample Consensus (RANSAC), which has been applied to the Location Determination Problem (LDP) in computer vision<sup>15</sup>, neural networks (and associated training methods such as gradient descent), which have been used in conjunction with the Hough transform to perform simple object recognition<sup>16</sup>, least squares methods such as the Levenberg-Marquardt variant<sup>17</sup>, and least-median-squares<sup>18</sup>. One major drawback in using these methods is that coding the calculation of these derivatives is difficult, especially for a free form representation of objects (nearly impossible, in fact).

In addition to being relatively intuitive to code, a method used in an environment such as SIMON's must be tolerant of outliers in the data set. Derivative-based methods have an upper bound on the number of outliers that can be tolerated. Least-median-squares, for instance, can not handle a situation where more than 50 percent of the data points are outliers.

Several other optimization methods that we have encountered do not utilize derivatives in their formulation. SIMON utilizes simulated annealing optimization. Simulated annealing has been used for such problems as the Traveling Salesman<sup>19</sup> and circuit performance design optimization<sup>20</sup>, but not for building site models until the advent of SIMON (to the best of our knowledge). Genetic algorithms also fall under this category and have been used for adaptive image segmentation<sup>21</sup>. Both algorithms were promising for use in our research because of their tolerance of outliers (simulated

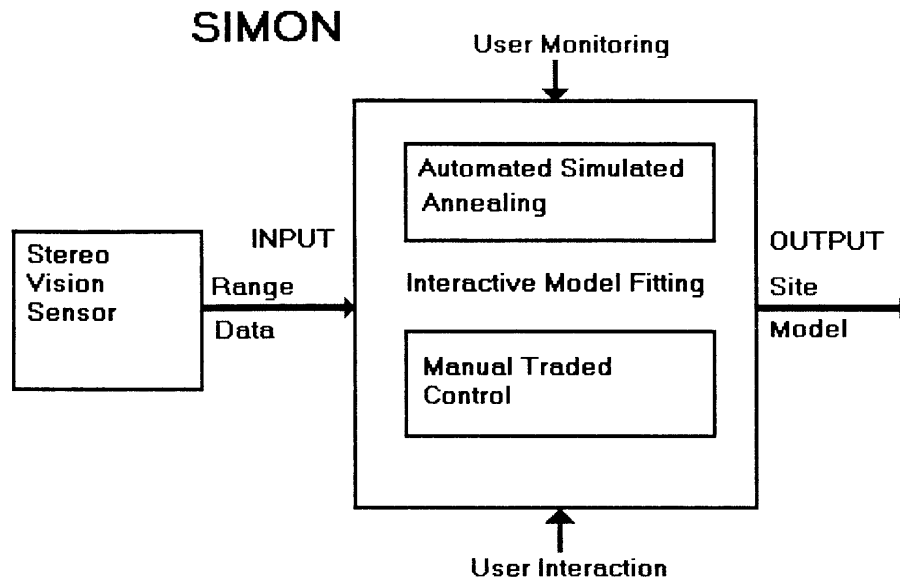
annealing has exceeded 70% when used in SIMON's experimental testing) and ease of implementation. We chose simulated annealing over genetic algorithms due to our greater familiarity with the algorithm as well as the availability of usable code.

### 3. SYSTEM DESCRIPTION

We have designed and implemented an *interactive* modeling system (*i.e.*, one that combines an automated fitting mechanism with human supervision) to solve the pose estimation and/or site modeling problem. The system is called SIMON and consists of three major components (see Figure 1): a stereo vision sensor, an automated optimization algorithm called *simulated annealing*, and a human supervisory control method known as *traded* control. Stereo vision is a source of sparse range data to which 3-D graphical models can be fit. The latter components (simulated annealing with traded control) were implemented together in our model-fitting application using RapidApp (a visual C++ rapid application and GUI development package) and the Open Inventor graphics libraries on an SGI Indigo2 workstation. Simulated annealing fits a primitive model to data by minimizing an objective function based on the distances from the points to the corresponding closest points on the surface of the model. Presently, a primitive model refers to a cylinder or cube, although cones, spheres, and other primitives (should they become available) could be added easily. If the model gets stuck in a local minimum (since simulated annealing is an iterative process, local minima are a possibility) a human supervisor can assume control of the fitting procedure via traded control. This supervisor



can then encourage the model in the direction of lowest error using a 6 DOF manipulation device and, thus, the best fit as the global minimum value of the objective function was designed to return the correct pose of the object.



**Figure 1: The major components of our site modeling system**

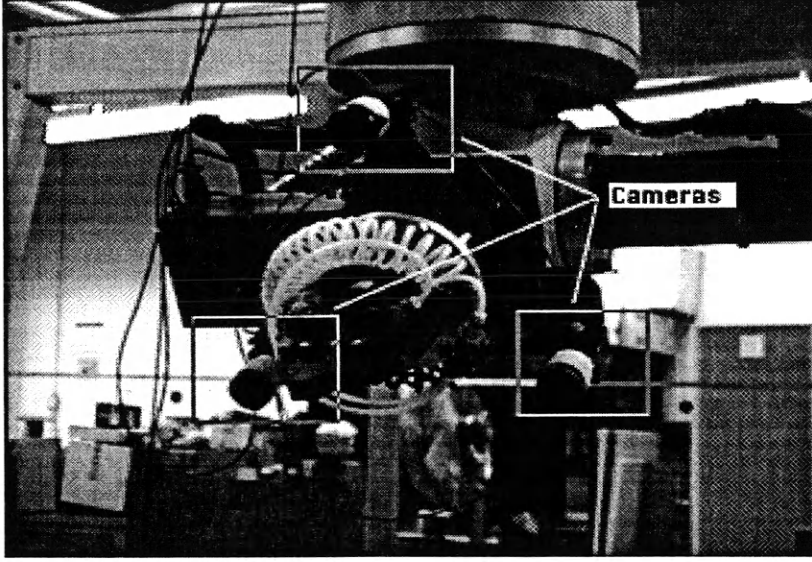
The remainder of this section discusses the major components of our system in more detail. In particular, each component's composition, implementation, and function will be discussed as well as the interaction between components. The description of the stereo vision sensor is necessary due to its importance to the system, but will be less detailed than the remainder of the system since its development was beyond the scope of this thesis.

### **3.1. Stereo Vision Sensor**

Stereo vision is a technique used in a wide variety of computer vision applications for gathering 3-D and depth information from images. Stereo was chosen as SIMON's source of range data for a number of reasons. First of all, stereo vision is a relatively inexpensive source of range data when compared to alternatives such as structured light or laser radar. Also, stereo vision data can be collected quickly. Images from the stereo vision system interface smoothly with the Galileo video capture board on the SGI workstation. This fact does not apply to the alternatives which require special hardware and software interfaces to transmit their data to a computer.

The range points from stereo are sparsely distributed as a result of the difficulty in matching points between images in areas where the contrast is low. This makes the pose estimation problem more challenging since the edges of an object are invariably less well-defined than with the use of dense range data. Thus, the challenging nature of stereo range data is a good test for SIMON -- *i.e.*, if SIMON can work well with stereo data, it should work well using just about any source.

The word "stereo" implies that a pair of images of the same object or scene are used, but the use of three or more views of the same scene is possible<sup>22</sup>. Our sensor consists of a trinocular array of cameras placed in an equilateral triangular configuration (Figure 2).



**Figure 2: Trinocular stereo camera array mounted on gantry robot end effector**

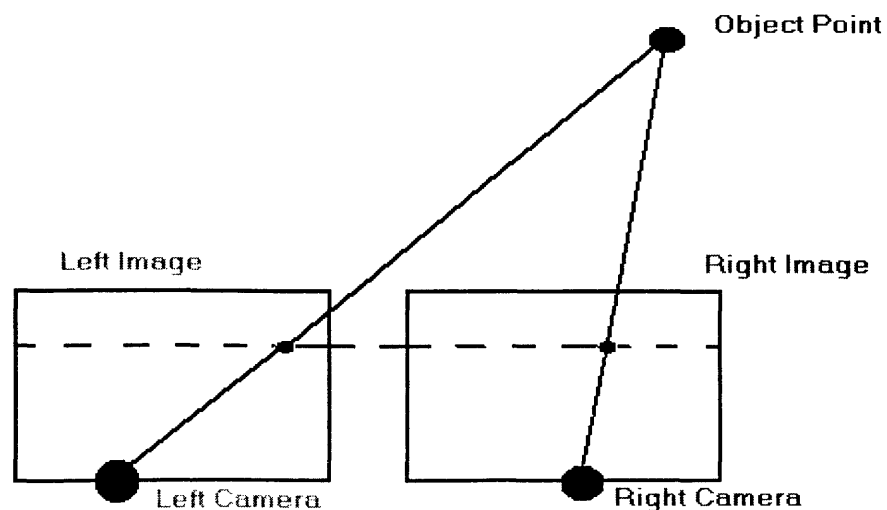
The two cameras on the bottom are the left and right components of a stereo pair. In stereo vision, depth information from two cameras is gathered by first matching corresponding pixels between images and then calculating depth using triangulation. Pixels (points) in the high contrast regions of one image are matched to pixels in the other using the following, cross-correlation-based score<sup>23</sup>:

$$r(d_x, d_y) = \frac{\sum_{(x,y) \in S} [f_1(x,y) - \bar{f}_1][f_2(x+d_x, y+d_y) - \bar{f}_2]}{\left\{ \sum_{(x,y) \in S} [f_1(x,y) - \bar{f}_1]^2 \sum_{(x,y) \in S} [f_2(x+d_x, y+d_y) - \bar{f}_2]^2 \right\}^{1/2}}$$

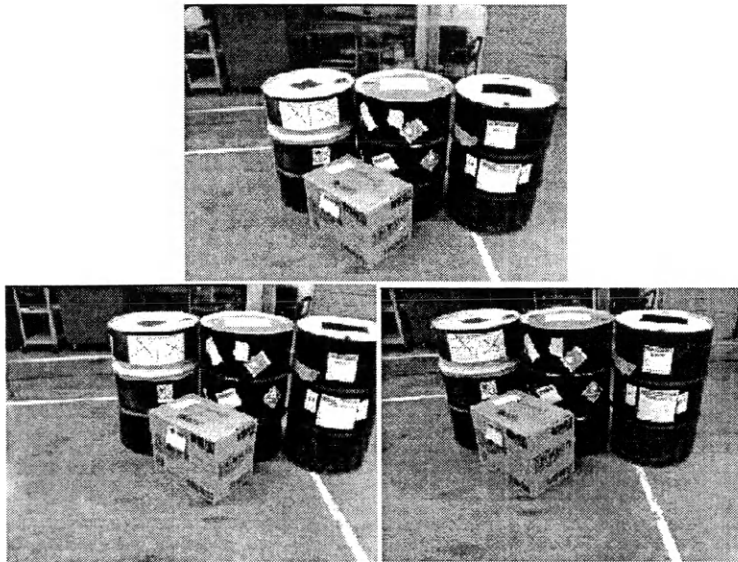
If this score exceeds a pre-determined threshold, the points are tentatively considered a match. The depth of each range data point is then calculated from the locations of a pair of matched points using triangulation. Spacing between the left and right cameras gives

the length of the base of a triangle with vertices at the left focus, right focus, and the point in question as seen in Figure 3. Assuming proper calibration, the left and right images are then used to calculate the point's angle of deviation from each image center. With these measurements available, the remainder is simple geometry and use of the Pythagorean theorem.

Further verification via the third or "center" camera is done following the depth calculations. This camera is placed halfway between the left and right cameras and slightly above them. To perform verification, the derived 3-D point is projected into the third camera image (see Figure 4 and Figure 5 for images and resulting data). If no feature was found at the predicted location, the point is eliminated. Table 1 displays tabulated stereo matching statistics.



**Figure 3: The stereo vision "scenario" (adapted from *Machine Vision*<sup>18</sup>)**



**Figure 4: Images (left, center, right) from stereo vision system**



**Figure 5: Final range points gathered**

**Table 1: Sample stereo matching statistics (from preliminary experiments)**

Scene	Left Interest Points	Right Interest Points	Matched Left-Right Points	Final Points After Verification	Final Remaining Mismatches
box1	167	216	99	68	3
box2	105	120	65	49	4
box3	103	140	65	47	3
drum1	340	369	161	129	3
drum2	133	204	63	55	3
drum3	186	194	91	79	3
gantry1	235	317	148	120	6
gantry2	163	231	118	102	2
gantry3	163	262	107	93	1
gantry4	192	255	110	91	2
gantry5	180	265	142	89	1
gantry6	218	282	164	106	9
gantry7	203	264	154	121	8
Average	184	240	114	88	4

### **3.2. Automated Optimization/Model Fitting**

Data acquired from the stereo vision sensor is fed to our model fitting system. This system consists of two major components, namely an automated component and a manual component. The automated portion is a numerical representation of an optimization algorithm called *simulated annealing*. The routine was integrated with the Open Inventor and RapidApp-generated code so that the results of the algorithm (*i.e.*, the pose of the model) could be viewed continuously and monitored by a human supervisor. The incorporation of automation into SIMON was made to speed execution of site modeling. The remainder of this section will address the simulated annealing algorithm itself

followed by specifics of the user interface portion of SIMON and the computer vision techniques that were utilized in the solution process.

### 3.2.1. Optimization by simulated annealing

As discussed in the literature survey and above, simulated annealing performs optimization on an iterative basis. The version we used can be found in *Numerical Recipes in C*<sup>17</sup> and is based on the *downhill simplex* method of optimization. The vertices of the simplex consist of n-dimensional vectors which represent possible states of the system. In this case, the state of the system was the pose of the primitive model -- specifically, a 6-D vector with 3 translation parameters (x, y, z in meters) and 3 orientation parameters (ax, ay, az -- the Euler angle representation in radians). There are n+1 of these vectors (corresponding to vertices), each representing slightly different states of the system. Downhill simplex will then choose the vertex that evaluates to the maximum value of the objective function and “reflect” or “expand” it in the geometric sense to make it achieve a lower objective function value. Smaller, “downhill” movements toward a minimum cost occur as the simplex contracts. Downhill simplex could thus be termed a greedy algorithm.

However, our needs required a method that was not greedy as the error spaces in our domain of problems are often littered with local minima -- the kinds of problems that cause simple greedy algorithms to fail miserably. These local minima can be caused by

faulty data, objects that may or may not be similar to the one we are trying to fit, background noise, and many other such anomalies. Simulated annealing builds upon the downhill simplex algorithm through the use of a “temperature” parameter. The temperature is actually a statistical measure which reflects the probability that state vectors having greater error will be retained in the simplex -- a higher temperature means a higher probability. Specifically, the reflections, expansions, and contractions of the simplex are tested via evaluation of the objective function to determine whether the move will be kept. If the change in the value of the objective function is less than or equal to zero ( $\Delta E \leq 0$ ), then the move will be kept. However, if the change in the value of the objective function is greater than zero, the probability that the move will be kept is a function of this change and the temperature (specifically,  $e^{-\Delta E/k_B T}$ ). This temperature is reduced iteratively in our system, retaining only 99 percent of its original value after every 100 iterations. This allows the state of the system (in this case, the pose of the current model being fit) to settle into a minimum much as the metal cooling process for which simulated annealing was named settles into a minimum energy state. Other temperature reduction schedules are possible and can improve results depending on the problem<sup>24</sup>.

The objective or cost function that simulated annealing minimizes uses the locations and orientations of the model and the locations of the range points. Our goal is to



minimize the distances from all data points corresponding to the object to the visible surface of the model. The objective function we use is:

$$errorscore = \sum_{i=1}^{\#pts} -G/r_i$$

Here,  $r_i$  is the distance from a point to the model's surface and  $G$  is constant. Thus, the contribution of the  $i^{\text{th}}$  point to the score varies as the inverse of its current distance from the surface of the model to be fit. Note also that this equation is the analogue of the potential energy in an electrostatic or gravitational environment. This measure minimizes the contribution of outlying points which can present a problem for least squares and other such estimation methods. Closer points exert a large force since  $1/r$  is large for relatively small values of  $r$ . For  $r$  less than 1 centimeter, which is the assumed stereo error, the contribution to the score was taken to be 1 centimeter to avoid infinitely large contributions. Likewise, for large  $r$  (*i.e.*, outlying points) the small value of  $1/r$  indicates a much smaller contribution.

To calculate the distance from a point to the closest visible surface of the model, we employ several steps. Since the pose of the model and the camera are known, transformations from a camera-centered coordinate system to an object-centered coordinate system are possible and greatly simplify distance calculations.

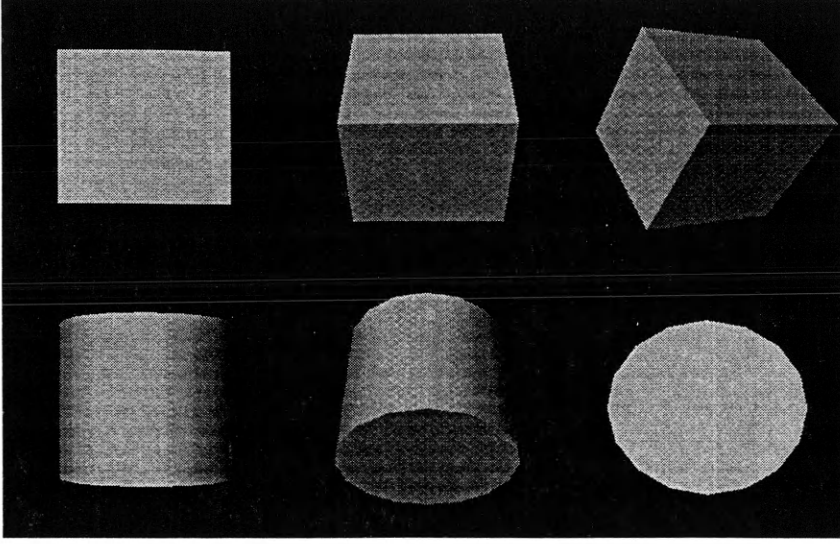
The pose of a primitive object in Inventor is represented using a 7-D vector. This vector consists of three translation components ( $x$ ,  $y$ ,  $z$ ) and four rotation components in

an axis-angle representation (ax, ay, az, and theta). From the 7-D vector, we arrive at a rotation and translation of axes which places the origin at the object's center and aligns the y-axis with the primitive's major axis.

The next step is to determine the visible faces of the primitive (Figure 6). Knowing the visible faces allowed us to calculate the distance from each range point to its associated closest visible surface on the primitive. For instance, the visible surfaces of a cylinder or cube model can be found using the orientation and location of the model (see Appendix for sample code). Then, classification of the closest region on the primitive (rim, cap, or lateral surface for a cylinder; edge and lateral surface for a cube) makes calculation of the distance from a range point to that region trivial. The fact that we are using objects that can be described with few parameters (radius and height for a cylinder; height, width, and depth for a cube) further simplifies our calculations. Given a cylinder and a coordinate system with origin at the center of this cylinder, a point whose y-coordinate is greater than  $h/2$  ( $h$  is the height of the cylinder) and whose distance from the y-axis is greater than  $r$  ( $r$  is the radius of the cylinder) is closest to the upper rim of the cylinder. The distance from the point to the rim is then easily calculated via:

$$d = \sqrt{(y - h/2)^2 + \left(\sqrt{x^2 + y^2} - r\right)^2}$$

This is done for all points in the scene at each iteration for the current primitive model.



**Figure 6: Varying orientations for cube (one, two, and three visible faces) and cylinder (side, side and cap, and cap only visible) models**

The result of the numerical error score is taken to be the value of the objective function and the performance for a particular pose. Note that the pose in the optimization routine was previously mentioned to be a 6-D vector. To get this 6-D vector, which uses a 3-D Euler angle representation (angles of rotation about x, y, and z axes) instead of Inventor's axis-angle convention, the following equations are utilized<sup>25</sup>:

$$R_K(\theta) = \begin{bmatrix} k_x k_x v\theta + c\theta & k_x k_y v\theta - k_z s\theta & k_x k_z v\theta + k_y s\theta \\ k_x k_y v\theta + k_z s\theta & k_y k_y v\theta + c\theta & k_y k_z v\theta - k_x s\theta \\ k_x k_z v\theta - k_y s\theta & k_y k_z v\theta + k_x s\theta & k_z k_z v\theta + c\theta \end{bmatrix}$$

$$\beta = A \tan 2(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2})$$

$$\alpha = A \tan 2(r_{21} / c\beta, r_{11} / c\beta)$$

$$\gamma = A \tan 2(r_{32} / c\beta, r_{33} / c\beta)$$

In the above equations,  $R_K$  is the rotation matrix created from an axis  $(k_x, k_y, k_z)$  and an angle  $(\theta)$ .  $\beta$ ,  $\alpha$ , and  $\gamma$  are the angles of rotation about the y, z, and x axes, respectively. Finally,  $c\theta$  equals  $\cos\theta$ ,  $s\theta$  equals  $\sin\theta$ , and  $v\theta$  equals  $1-\cos\theta$ .

We use Euler angles instead of axis-angle because they are independent of one another. The axis-angle components use four values to represent three degrees of freedom and can result in an invalid orientation (*i.e.*, an axis which is not a unit vector). Euler angles, on the other hand, can assume any value and not produce invalid rotations. Ultimately, the ability to obtain axis-angle representation from Euler angles enables us to render to the position of the current primitive object at each iteration of the fitting routine.

### 3.2.2. The Model Fitting GUI

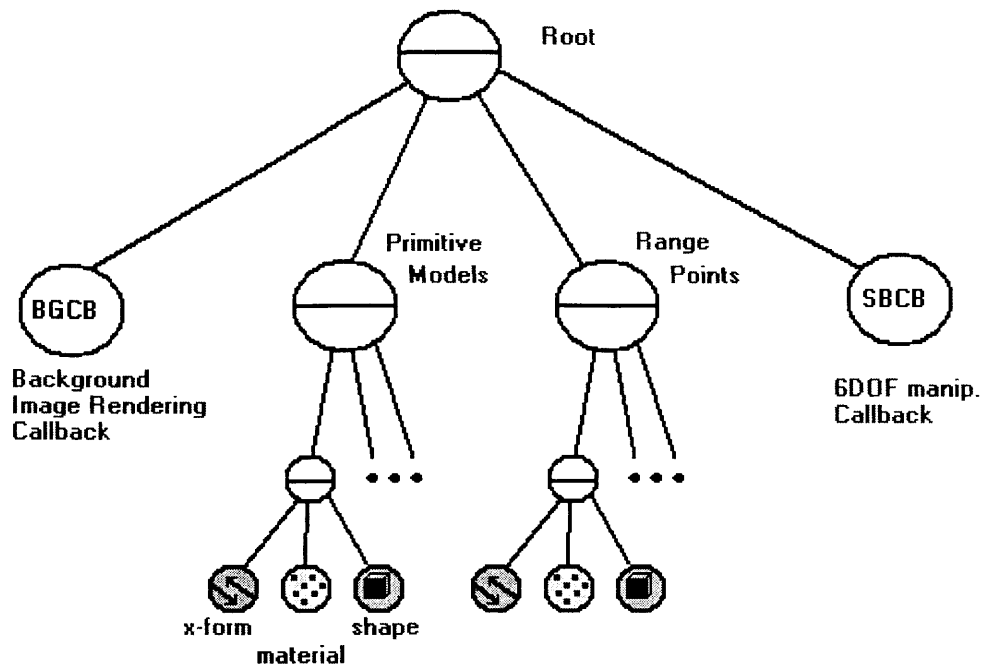
The simulated annealing code from *Numerical Recipes in C* interfaces quite easily with our system for several reasons. First of all, the pose of the model (a 6-D vector) can be used as the state of the system. Also, the ability of C and C++ to be mixed in a program gives RapidApp and the compiler few if any problems. Finally, we can incorporate our own error function. Calls to this function can then evaluate the error associated with a particular system state (or pose of the model, in this case).

Inventor, with the help of the GUI created through RapidApp, enables us to view the state of the system at each iteration of the fitting procedure. Open Inventor displays graphical objects in a *Viewer* window which is attached to a RapidApp object called a

*BulletinBoard* (Figure 7). The contents of the window are dictated by a data structure called a *scene graph*. The scene graph is a tree structure containing objects (graphical depictions of curves, solids, points, and/or surfaces) and their associated attributes. An example of the scene graph associated with the contents of our Viewer window is shown below in Figure 8.



**Figure 7: The application GUI**



**Figure 8: An example of a scene graph (adapted from *The Inventor Mentor*<sup>26</sup>)**

Inventor enables us to monitor the progress of our application by refreshing the scene's depiction in the Viewer window every time one of the models' attributes is changed (see fitting sequence in Figure 9 and Figure 10). This auto-refresh is accomplished via an in-order traversal of the scene graph much as the initial rendering of the scene is done.

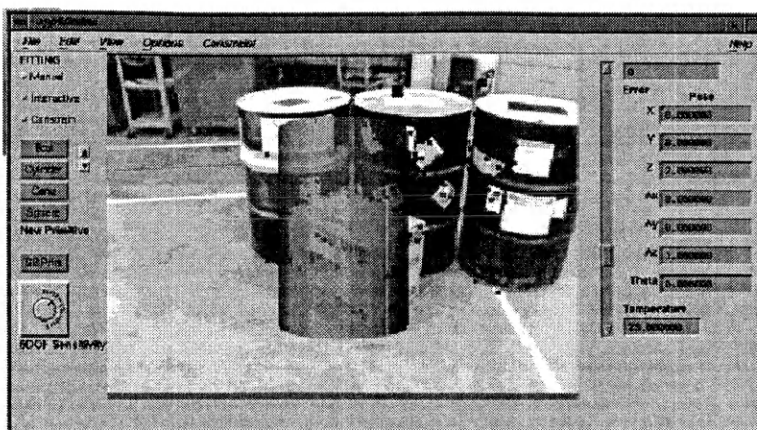


Figure 9: Initial position of the model



Figure 10: Final position of model after fitting

As mentioned previously, the object model as well as the range points and background image of the scene are rendered in a Viewer window. However, the application environment consists of two control panels in addition to the Viewer window (see Figure 11). Also, the pulldown menus at the top contain several operations that were not crucial to our experiments (to be covered in the next section) with the exception of exiting the program.

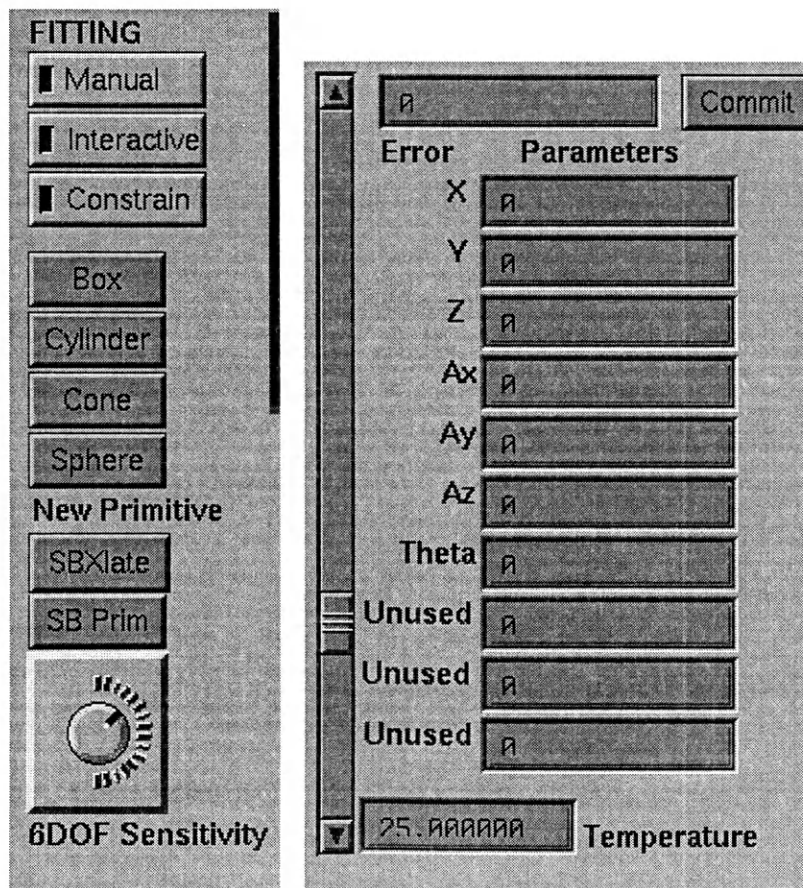


Figure 11: The model-fitting application control panels



The left control panel consists of several controls and buttons. At the top are three buttons for selecting fitting procedures. The “Manual” button chooses manual fitting in which the human operator controls the movement of the primitive object using a 6 degree of freedom (6 DOF) manipulation device -- ours is a Logitech Magellan, but others exist. “Interactive” model fitting uses simulated annealing in conjunction with human supervision via the 6 DOF manipulation device to complete the task (see next subsection on *traded* control). Finally, the button labeled “Constraint” enables *shared* control of the model, which will be discussed in reference to future work in section 5.

Below the buttons which activate the modeling routines are buttons that introduce primitives to the scene. When pressed via the mouse pointer, a primitive object is placed in the scene two meters from the camera center with its major axis aligned with the y-axis. In SIMON, all four primitive types are made available. However, only the cylinder and cube are used since code to evaluate the objective functions for cones and spheres has not yet been developed.

One last control on the left side of the application window is the 6 DOF sensitivity dial. Sensitivity, in this case, refers to the speed with which an object under the 6 DOF manipulation device’s control moves across the screen. The reduction of this sensitivity is crucial when fine-tuning the fit of a primitive to its associated range data.

The entire right control panel is dedicated to displaying vital information about the current model as well as data about the progression of the fitting algorithm. For instance,

the orientation of the model is displayed using the axis-angle orientation convention and the translation of the model (its center's location) is displayed as an  $(x, y, z)$  point in camera-centered coordinates. The error at that particular pose is also displayed as a decimal score falling between 0 and 1000. Toward the bottom, attributes of the model such as height, width, and radius are displayed.

### **3.3. Manual Component (*traded control*)**

Simulated annealing was chosen for its reputed ability to ignore local minima and find the global minimum in error space for several difficult problems. Despite this reputation, the method is still susceptible to getting caught in these minima in our particular problem space. We then concluded that complete confidence in simulated annealing to solve the problem would sacrifice consistent and reliable fitting of the graphical models to the relatively sparse range data.

An interactive model fitting system like ours overcomes this problem by employing human supervisory control in addition to automated optimization. The supervisory control method inherent in our application is known as *traded control*<sup>7</sup>. Traded control, with respect to our application, means that either one or the other of simulated annealing and human control via the 6 DOF manipulation device is manipulating the current primitive at a given time. Therefore, when operating in interactive mode, touching the 6

DOF manipulation device will give control to the human operator. This control will not be returned until the operator releases the device for a short period of time (525 ms).

The implementation of *traded* control was done via the use of a periodic timer class called “VkPeriodic”. Creation of a derived class of VkPeriodic enabled us to redefine a function called “tick” which automatically executes at intervals of 175 ms. When “Interactive” model fitting is running, “tick” calls 100 iterations of the simulated annealing routine and then reduces the temperature prior to the next period. If the spaceball is touched, a flag within the class is tripped and “tick” instead calls functions corresponding to purely manual fitting at every period. Additionally, control remains with the 6 DOF manipulation device until no movements had been attempted during the course of three periods (525 ms).

Previously, it was mentioned that traded control could be used to move the current model out of a local minimum and encourage it toward the correct pose of the object of interest. The experiments whose description follows this section showed that traded control could also be used to place a primitive model initially and then fine tune the fit via simulated annealing. A direct comparison of the performance of this fitting strategy to one where control was initially granted to the simulated annealing routine also highlights differences between the two. In cases where the primitive did encounter a local minimum, the amount of encouragement necessary by the supervisor was minimal.

## 4. EXPERIMENTS

During the course of our research we designed several experiments to demonstrate the performance and reliability of our interactive site modeling application. In particular, we wanted to compare our application to an identical but purely manual site modeling system. Using human subjects, we executed two sets of experiments:

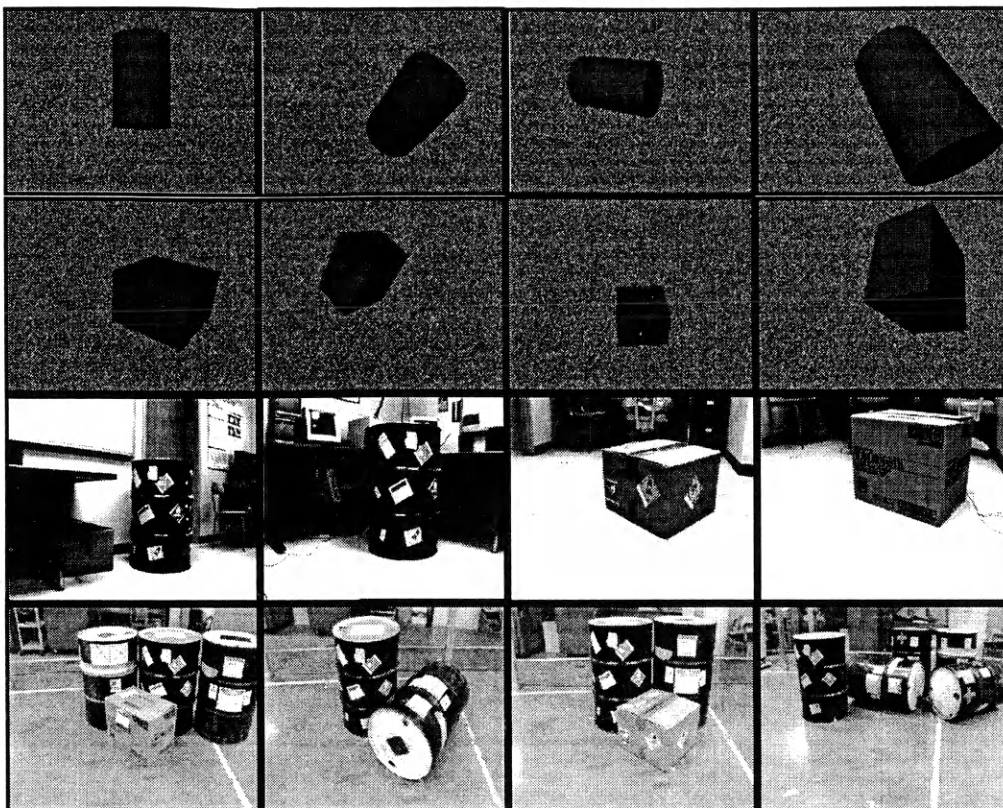
1. An informal set of experiments in the summer of '96
2. A formal set of experiments in November of '96 whose design was based on the preliminary experiments.

This section summarizes these experiments in terms of their design, execution, and an analysis of the data that resulted. Each portion of the formal experiments was designed to bring about a conclusion relative to specific hypotheses set forth prior to the experiments. In addition, rules were made concerning the execution of these experiments to ensure minimal corruption of experimental data. Finally, results were tabulated and analyzed utilizing formal statistical methods. The results were promising: all but a portion of one of the null hypotheses set forth was refuted.

#### **4.1. Preliminary Experiments**

Our first set of experiments were as much a trial in patience as a set of trials. In all, four human subjects devoted no less than an hour-and-a-half apiece to sitting in front of our application and fitting models to data. The first 10 to 15 minutes of each session was devoted to practicing the use of the 6 DOF device. The rest was devoted to fitting the scenes and cutting and pasting the data to a file at the end of each trial. A trial was completed when the subject felt that an acceptable fit to the data had been attained.

The purpose of the experiments was to test the performance in terms of time and accuracy of the interactive application versus an identical but purely manually-driven fitting routine. We compared performances by having the subjects fit 16 different scenes: 8 of the scenes contained “real” data and 8 contained synthetically-generated data. The scenes also contained varying densities of range data on the objects of interest. We had each subject perform both interactive and manual model fitting on each scene (although not necessarily in that order). Trials for the same scene using different fitting techniques were spaced well apart to discourage subjects from memorizing a particular scene. The order of the scenes as well as their depictions are shown in Figure 12 and Table 2.



**Figure 12: Scenes used for preliminary experiment (see Table 2 for order)**

**Table 2: Order in which the scenes were introduced to the subjects (order repeated during second half of trial with opposite fitting methods)**

Scene	Fitting Method
synth1	Manual
synth2	Interactive
synth3	Manual
synth4	Interactive
synth5	Manual
synth6	Interactive
synth7	Manual
synth8	Interactive
drum2	Manual
drum3	Interactive
box2	Manual
box1	Interactive
gantry1	Manual
gantry2	Interactive
gantry3	Manual
gantry6	Interactive

The results of these preliminary experiments were quite promising. From the results shown in Table 3, it is evident that interactive fitting was superior to manual fitting in task time and accuracy. To further support the reliability of SIMON, we ran the interactive routine without the aid of manual supervision (purely **automated** fitting). Automated fitting resulted in a correct solution (nearly identical to the interactive solution) only 50% of the time -- an unacceptable rate of success. The remaining 50% of the time, the final pose was not even close to ground truth. Therefore, we had evidence that *traded* control was vital to consistently fitting models to data within SIMON.

**Table 3: Summary of preliminary evaluation results**

	Manual	Interactive	Improvement
Task Time	83 sec	30 sec	53 sec (63%)
Pose Translation Error	1.1 cm	0.9 cm	0.2 cm (18%)
Pose Orientation Error	2.3°	1.0°	1.4° (57%)
Fit Error Score	527	485	42 (8%)

## **4.2. Formal Experiments**

In November of 1996, we designed and executed a formalized set of experiments which more specifically addressed the differences in performance between interactive and manual model fitting. Similar experiments, designed to realize the effect of an operator aid on performance, were done for a machine-vision-based teleoperation system<sup>27</sup>.

Several of our aims and methods for carrying out the experiments remained similar to the preliminary experiments, however. First of all, human subjects were asked to fit cylinder and box models to range data using both interactive and purely manual fitting. For a fair comparison under experimental conditions, our manual modeling scheme was simply interactive mode without the automated portion (as was done in Hsieh's work on the RADIUS project<sup>14</sup>). This measure required the subject to fit the model using only the 6 DOF device. Also, each scene was fit once interactively and once manually by each subject. This enabled a direct comparison between fitting methods.

The hypotheses we wished to prove through formal experimentation were ultimately based on the results and observations of our preliminary experiments. However, there



were many major differences between the experiments. The formal experiments were developed and executed with the purpose of refuting specific hypotheses. Also, several strategies for interactive fitting were compared. The experiments were shortened (fewer scenes to fit) in the interest of attracting more subjects and creating more well-distributed data. In addition, both the subject and the person administering the experiments needed to agree on when the best fit to the data had been reached. This measure ensured that the subject would not terminate the fitting process prematurely or would not commit a gross fitting error. Finally, only synthetic data was used for the actual experimental trials. This ensured consistent results since the ground truth pose of the object was known.

#### 4.2.1. Discussion of hypotheses

Prior to designing our experimental evaluations, we set forth the following hypotheses concerning the performance of our interactive model fitting application (which we wished to refute):

- 1) Interactive modeling should not significantly outperform purely manual modeling in total task time and fitting accuracy (where fitting accuracy includes both pose error and error in fitting to the range data).
- 2) Operator effort (measured in terms of user time) will not decrease significantly when utilizing interactive modeling.

- 3) Interactive modeling does not make up for a lack of expertise. Specifically, subjects who differ greatly in mean task time using purely manual modeling will be statistically distinguishable on the basis of task time using interactive modeling.
- 4) Initial placement of the primitive object model by the human supervisor will not significantly improve performance when interactively modeling a “difficult” scene, while minimal human interaction (no initial placement) will not significantly increase performance on “easy” scenes.

These hypotheses were based on the results and observations of the preliminary experiments done in the summer of '96.

The first hypothesis specifically states what we set out to refute in our preliminary experiments. This hypothesis embodies the assumption that the automated component of an interactive system such as this will speed the process of model fitting through fast computation. Enhanced computational capabilities should also allow more accurate model fitting since the computer can recognize more subtle differences in translation and orientation with respect to the data than can the human operator.

Hypothesis two focuses on the amount of effort exerted by the human operator. This is an especially important consideration in an industrial setting since a relaxed operator typically performs better. “Effort,” in this case, would be measured in *user time* or the time spent using the 6 DOF manipulation device during completion of a task. A significant reduction, say on the order of 50 percent, would be helpful to people who do

site modeling for extensive periods of time. It was assumed that the time saved would be a result of the savings in total task time realized using interactive fitting as well as the fact that control would be traded between the human operator and the automated fitting routine.

The third hypothesis dictated a division between novice and expert users. The assumption here was based on our observation that subjects in the preliminary experiments differed greatly in manual task times and very little when using interactive mode. As a result, interactive fitting seemed to have closed the gap between the proficient and non-proficient manual modelers. Refutation of the third hypothesis would support this assumption.

The fourth and final hypothesis was derived from the observation that different fitting strategies seemed to work better for scenes of differing difficulties. We defined difficulty as a function of the time it takes the average subject to manually fit a model to data for the given scene. We then assumed that since the more difficult scenes were more likely to contain more local minima, that initial placement of a model in the region of the global minimum would reduce simulated annealing's chances of being influenced by local minima. Conversely, minimal human interaction should be necessary for easy scenes in interactive mode because of an obvious global minimum or lack of local minima.

Thus, we designed our experiments around these hypotheses. The next subsection describes how our experimental design proceeded relative to the hypotheses.

#### 4.2.2. Experimental design

The design of our experiments followed logically from the hypotheses set forth prior to this stage. Specifically, each hypothesis implied the existence of certain experimental variable(s) whose values would need to differ amongst the trials. Several sources, including works by Hsieh<sup>14</sup>, Hoff, *et al.*<sup>27</sup>, Hockman and Jenkins<sup>28</sup>, and Knotts<sup>29</sup>, cite experimental design or the design of experiments (DOE) methodology as a crucial step in testing one's research.

**Table 4: Independent experimental variables**

Variable Name	Possible Values
Modeling Method	manual, interactive
User Expertise	expert, novice
Scene Difficulty	easy, difficult
Fitting Strategy	initial placement, no initial placement

The independent variables present in our experiments are summarized in Table 4. The first of these, the “Modeling Method” employed for a particular trial, was introduced as a result of the first and second hypotheses. These two hypotheses stated the need for a direct comparison of interactive fitting's performance in terms of time (total and user) as well as accuracy. Thus, each subject was asked to fit each scene once using manual fitting and once using interactive fitting during the course of the experiment.

The second variable, “User Expertise”, was calculated following the experiments. Upon their completion, we calculated the average time taken by each subject to fit the

scenes manually. Those in the top 50 percent were deemed experts, while the remaining 50 percent were given a rating of novice. By rating expertise, we could use the subjects' interactive task times to support the third hypothesis; that is, the difference between average interactive task times for novices and experts could be compared to see if there was a negligible difference amongst them.

The final two experimental variables, "Scene Difficulty" and "Fitting Strategy", stem from the fourth hypothesis. Scene Difficulty was calculated following the experiments based on the average time taken to fit that particular scene manually. Similar to User Expertise, the toughest 50 percent were labeled difficult while the other 50 percent received a rating of easy. It was our intention to create scenes of varying difficulty by varying the attributes of these scenes such as number and types of objects (cylinders and cubes, to be exact) as well as the density of range data. Fitting Strategy, on the other hand, was varied in much the same way as Modeling Method -- for the interactive trials, half of the scenes were done with initial placement of the model and the other half with no initial placement. The distribution of Modeling Methods and Fitting Strategies for every subject is shown in Table 5.

**Table 5: Schedule of scenes encountered during experiments**

Scene	Modeling Method	Fitting Strategy
syn1	manual	
syn3	interactive	initial placement
syn9	manual	
syn10	interactive	no initial placement
syn4	manual	
syn2	interactive	no initial placement
syn12	manual	
syn11	interactive	initial placement
syn2	manual	
syn12	interactive	no initial placement
syn10	manual	
syn1	interactive	initial placement
syn3	manual	
syn9	interactive	initial placement
syn11	manual	
syn4	interactive	no initial placement

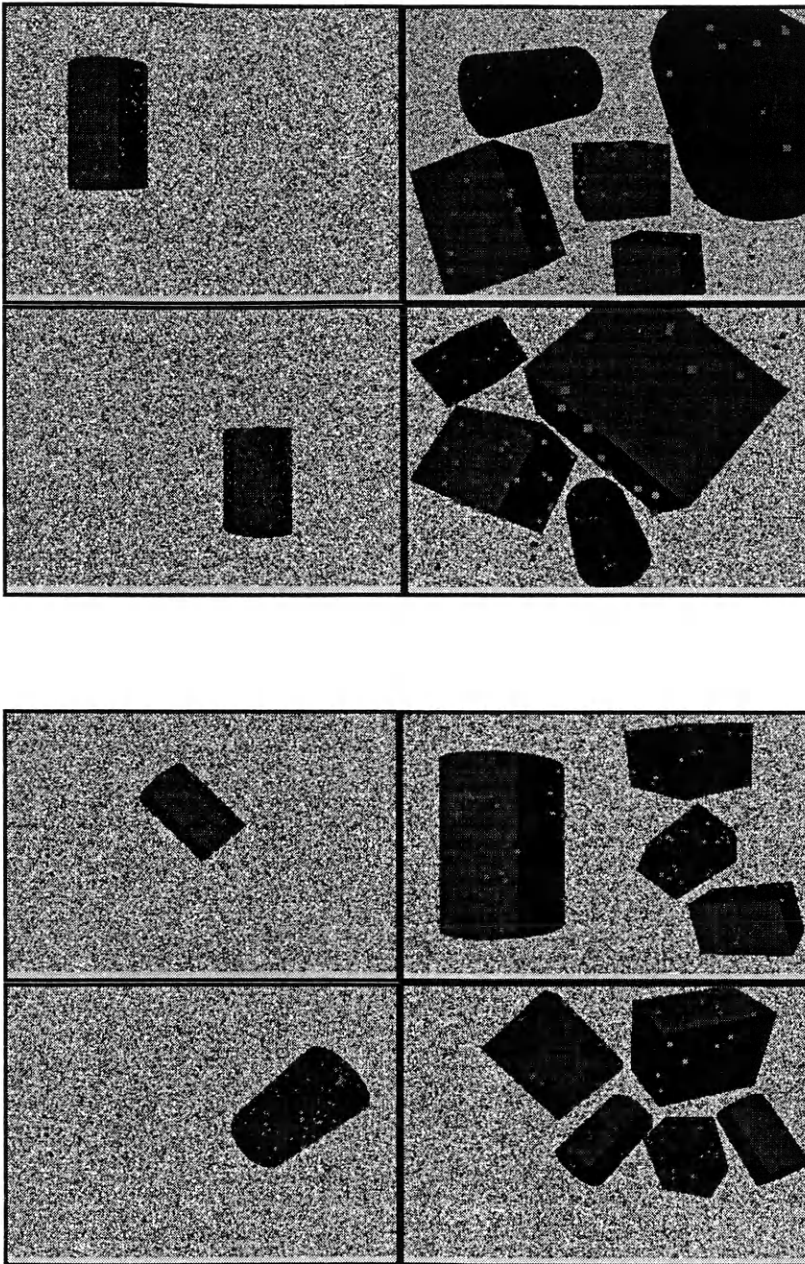
#### 4.2.3. Procedure/Execution

The need for more experimental subjects caused us to use fewer scenes in our evaluations. As a result, we obtained the services of 14 subjects during mid-to-late November of 1996. The subjects: were all students at the School of Mines; were all males save for three; were all fairly computer literate (most exceptionally so). In addition, all had normal or corrected vision and depth perception. Also, all but three were in their early twenties (two older, one younger). Left or right-handedness was not an issue due to the design of the 6 DOF device.

To begin with, each subject was briefed by the administrator (the author) on the scientific and industrial significance of the application. This briefing was followed by a

training session on the use of the 6 DOF manipulation device. The training session covered the three translations ( $x$ ,  $y$ ,  $z$ ) and changes of orientation (roll, pitch, yaw). Also, the subjects were asked to fit several practice scenes not in the list for the formal procedure. Each of manual modeling, interactive with initial placement, and interactive modeling without initial placement were used in this 10 to 15 minute practice session. The intent of the training session was also to minimize the learning curve in the trials that followed.

When practice was completed, the subject proceeded to complete all 16 trials for the 8 synthetically-generated scenes. These scenes are shown in Figure 13. Each scene contained approximately 80 to 100 range points, of which anywhere from 20 to 75 percent were outliers with respect to the object to be fit, and a background consisting of random grayscale noise placed at infinity. Points on the object were located exactly on the surface. Completion of a trial required the fitting of one model to data corresponding to the preselected object. This entire phase took anywhere from 30 to 50 minutes. This meant that 55 minutes to an hour on average was taken to complete the experiment for a particular subject. Scenes were introduced in pseudo-random fashion to reduce the possibility of the user having learned the sequence of the scenes. The object to be fit to was always that which contained the highest density of range points.



**Figure 13: Scenes utilized in the formal experiments, L to R from top: syn1, syn11, syn2, and syn10 (easy); syn3, syn9, syn4, and syn12 (difficult)**



To keep from skewing the experimental results, several special considerations were built into the experimental procedure. First of all, the training time for each subject on the 6 DOF manipulation device was held fairly constant. Second, room lighting for the subjects was kept constant, as well. Third, the evaluator and evaluatee had to agree as to when the closest possible fit of the model to the data was obtained. Fourth, the same number of applications and application windows were open on the workstation for each subject (an attempt to eliminate any machine performance issues, if they existed). Finally, each subject was asked to reduce the annealing temperature to zero when an interactive modeling task was completed so that each had the opportunity to achieve the best possible score.

Figure 14 and Figure 15 contain a blank evaluation sheet as well as the formal guidelines as they were written prior to the commencement of the evaluations.

Evaluation Form

Subject:  
**Expertise Notes**  
SB translate drum1:

SB Rotate drum1:

Practice scenes --

drum1 (manual)

box1 (int. IP)

gantry1 (int. A-C)

Scenes

		restarts	score	~time
syn1	manual			
syn3	int. IP			
syn9	manual			
syn10	int. A-C			
syn4	manual			
syn2	int. A-C			
syn12	manual			
syn11	int. IP			
syn2	manual			
syn12	int. A-C			
syn10	manual			
syn1	int. IP			
syn3	manual			
syn9	int. IP			
syn11	manual			
syn4	int. A-C			

Figure 14: Individual subject evaluation form

## Formal Evaluation Guidelines

### Schedule

- 1) Introduction to the application (2-3 minutes).
- 2) Introduction to the spaceball (6 DOF device) and begin evaluating expertise of subject. (2-3 minutes)
- 3) Practice scenes (5-10 minutes):
  - a) Drum1 -- fit manually, explain error score, color significance, and sensitivity control.
  - b) Box1 -- fit interactively using initial placement (IP) strategy, explain temperature significance.
  - c) Gantry1 -- fit interactively using automated-catch (A-C) strategy, end of initial expertise evaluation.
- 4) 16 trials, 8 different scenes, final expertise evaluation (35-45 minutes):
  - a) 8 manual fitting trials (4 easy, 4 difficult)
  - b) 4 interactive (IP) fitting trials (2 easy, 2 difficult)
  - c) 4 interactive (A-C) fitting trials (2 easy, 2 difficult)

\*\*\*NOTE -- Trial will be restarted if primitive "escapes" from view for more than five seconds.

### Summary of Trials and Scenes

#### EASY:

syn1	interactive(IP)	manual
syn2	interactive(A-C)	manual
syn3	interactive(IP)	manual
syn4	interactive(A-C)	manual

#### DIFFICULT:

syn9	interactive(IP)	manual
syn10	interactive(A-C)	manual
syn11	interactive(IP)	manual
syn12	interactive(A-C)	manual

### Specifics

#### IP:

The subject will be asked to place the model in the region of interest initially, then allow the automated fitting procedure to fine-tune the fit with minimal interaction following.

#### A-C:

The subject will allow the automated fitting procedure five to ten seconds as needed to snap the model into the region of interest, then coax the model into the final position through interaction.

#### Interactive:

The subject must reduce the temperature to zero for the final placement before terminating the fitting routine.

#### Manual:

The spaceball sensitivity must be reduced at least once in the subject's attempt to fine-tune the fit.

#### All:

The subject and evaluator must agree on the closest fit. The subject may ask the evaluator for hints as to the proper orientation if it is not obvious.

## Figure 15: Sheet containing formal evaluation guidelines

#### 4.2.4. Results

To verify which experimental variables had a significant effect on the performance of our system (and which did not), we used a statistical process known as *analysis of variance* (ANOVA)<sup>30</sup>. ANOVA, when applied to a particular experimental variable, resulted in a confidence measure expressed as a value of the one-sided F distribution. The confidence in terms of percentage can then be verified using a look-up table. The locations in this table are mapped using the number of divisions (*i.e.*, the number of possible values for the experimental variable) and the number of degrees of freedom (in this case, degrees of freedom refers to the number of trials associated with a particular value of the experimental variable). If the value obtained from the analysis meets or exceeds the value in the table, then the confidence associated with the values in the table can be assumed. The results of ANOVA for our experiments are summarized in Table 6. Comparisons that passed the ANOVA test exceeded 99 percent confidence. However, those that failed fell well below the values in the 95 percent confidence table.

**Table 6: Results of ANOVA**

<b>Dependent Variable</b>	<b>Independent Variable</b>	<b>Independent Affects Dependent?</b>
Total Task Time	Modeling Method	Yes
User Time	Modeling Method	Yes
Translation Error	Modeling Method	No
Orientation Error	Modeling Method	Yes
Error Score	Modeling Method	Yes
Task Time (easy interactive scenes)	User Expertise	No
Task Time (difficult int. scenes)	User Expertise	No
Task Time (easy interactive scenes)	Fitting Strategy	Yes
Task Time (difficult int. scenes)	Fitting Strategy	No

The comparisons that passed the ANOVA test (*i.e.*, returned a confidence above 99 percent) indicated that they had a significant effect on the associated *dependent* variables. Note also the comparisons that did not pass the ANOVA test. We concluded from the resulting confidence level that these independent variables did not have a significant effect on the associated *dependent* variables. For instance, User Expertise did not have a significant effect on interactive task times. Also, note that Modeling Method did not significantly affect Translation Error and Fitting Strategy did not have a discernible effect on task times for difficult interactive scenes. This indicates that assumptions concerning these variables were neither proven nor disproven.

Table 7, Table 8, Table 9, Table 10, and Figure 16 summarize the experimental performance of our system. indicates the improvement that interactive fitting brought about for each subject. Table 8 shows the improvement by the group as a whole in terms of the mean performance. Table 9 presents evidence that user expertise does not have a

significant effect on interactive modeling task time. Table 10 shows evidence of the improvement in fitting “easy” scenes when no initial placement was used. Finally, Figure 16 graphically indicates the performance gains realized using interactive methods. These results, when combined with the results of ANOVA and the fact that user time was reduced by 88 percent using interactive modeling, confirmed the following:

- 1) Interactive modeling outperformed purely manual modeling in total task time and fitting accuracy. This was by virtue of the fact that ANOVA showed Modeling Method to have a significant effect on Total Task Time, Orientation Error, and Error Score (refuted Hypothesis 1).
- 2) Operator effort (in user time) decreased significantly when utilizing interactive modeling. This was due to ANOVA’s conclusion that Modeling Method significantly affected User Time (refuted Hypothesis 2).
- 3) User expertise did not have a significant effect on interactive modeling task time. ANOVA showed that there was not a statistically discernible difference in Task Time based on User Expertise (refuted Hypothesis 3).
- 4) Minimal human interaction increased performance on “easy” scenes. To demonstrate this, the results of ANOVA indicated that Fitting Strategy had a significant effect on Task Time for easy interactively-fit scenes (refuted Hypothesis 4, part 2).

Therefore, all but the first part of hypothesis 4 were refuted based on our experimental evaluations.

**Table 7: Mean experimental results for the individual test subjects**

<b>Subject</b>	<b>Manual Time (sec)</b>	<b>Interactive Time (sec)</b>	<b>Mean Unscaled Error (Manual)</b>	<b>Mean Unscaled Error (Interactive)</b>
<b>1</b>	121.75	41	470.6738	414.185
<b>2</b>	143.75	49.25	429.67	414.1688
<b>3</b>	216.25	44	454.9838	414.2013
<b>4</b>	183.375	53.125	462.2038	414.18
<b>5</b>	197.375	50.125	472.3113	414.1788
<b>6</b>	127.25	36	429.11	414.2188
<b>7</b>	211.375	41.75	415.9325	414.2363
<b>8</b>	162.5	57.125	455.8113	414.385
<b>9</b>	142.5	31.75	428.4438	414.1325
<b>10</b>	223	34.5	426.3813	414.5738
<b>11</b>	162	26.125	416.0738	414.295
<b>12</b>	252.625	36.75	419.79	414.1575
<b>13</b>	233.25	25.125	416.2613	414.225
<b>14</b>	321.5	42.875	418.7938	414.1338

**Table 8: Mean experimental results for all scenes**

	<b>Manual</b>	<b>Interactive</b>
Task time (sec)	192.75	40.68
User time (sec)	192.75	23.23
Scaled error score (min 1000.0)	1101.48	1000.62
Orientation error (deg)	1.886	.758
Translation error (m)	.014	.025

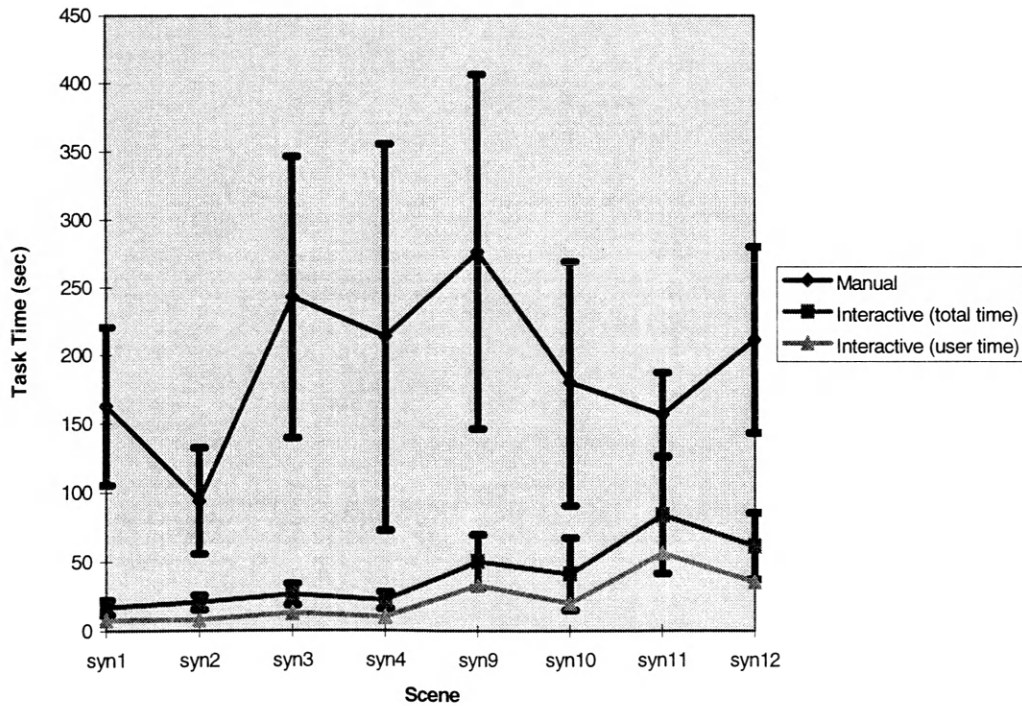


**Table 9: Interactive task times for varying user expertise**

User Expertise	Modeling	Mean Task Time (seconds)
Novice	Interactive	39.30
Expert	Interactive	42.05

**Table 10: Interactive task times using various fitting strategies**

Fitting Strategy	Scene Difficulty	Mean Task Time (seconds)
Initial placement	Easy	50.75
No init. placement	Easy	31.14
Initial placement	Difficult	38.71
No init. placement	Difficult	42.11

**Figure 16: Comparison of mean task times for each scene (error bars represent standard deviations)**

## 5. CONCLUSIONS AND FUTURE WORK

We have created an interactive site modeling system for use in hazardous and unstructured environments. The system combines simulated annealing and *traded* control to quickly and accurately fit primitive graphical models to sparse 3-D range data. We performed several experiments to demonstrate the quickness and accuracy of our interactive system compared to an identical but purely manually-driven system. In all but a few cases, the interactive system outperformed the manual system by a significant margin. In particular, our experimental results supported the following claims:

- 1) Interactive modeling outperforms purely manual modeling in total task time and fitting accuracy.
- 2) Operator effort (in user time) decreases significantly when utilizing interactive modeling.
- 3) User expertise does not have a significant effect on interactive modeling task time.
- 4) Minimal human interaction increases performance on “easy” scenes.

Also, through preliminary experimentation, we demonstrated how a purely automated system lacked the reliability to solve the site modeling problem consistently.

Several directions for future work and improving the site modeling application are possible. One such improvement was completed recently and involved building a finished site model. The previous application had the ability to fit multiple objects in scene, but the effects of data from objects that had already been fit made accurate fitting of the remaining objects difficult. Therefore, the removal of points associated with an object after it had been fit would eliminate the influence of such data. Our means of creating finished site models has not been formally tested as yet, but an example of a completed site model using the removal of points from “fit” objects is shown in Figure 17.

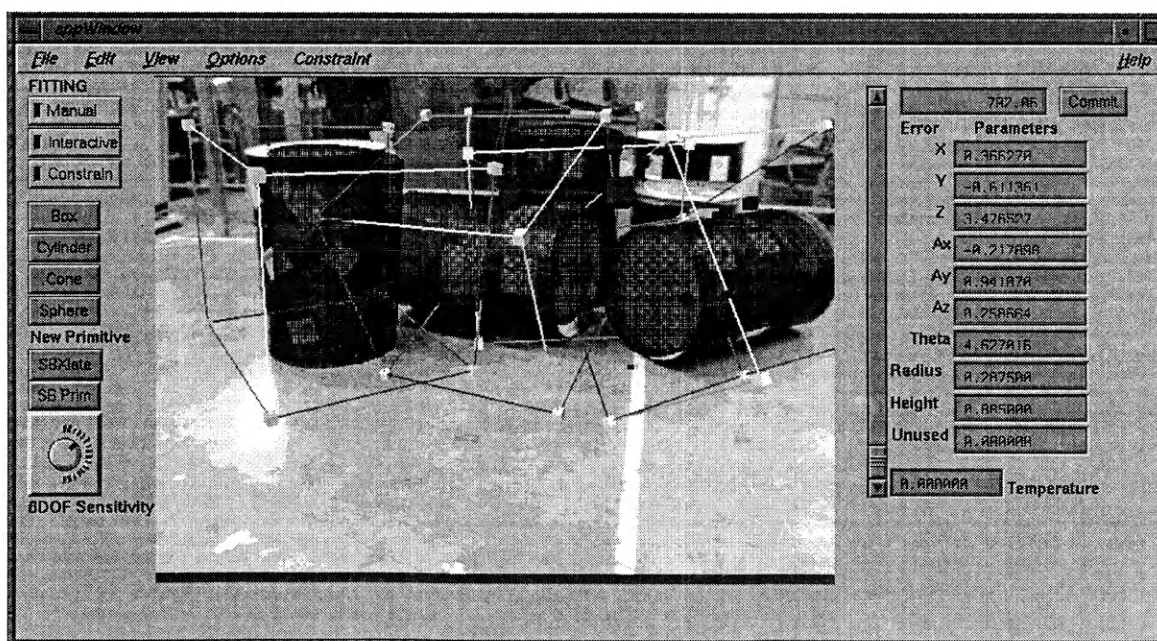


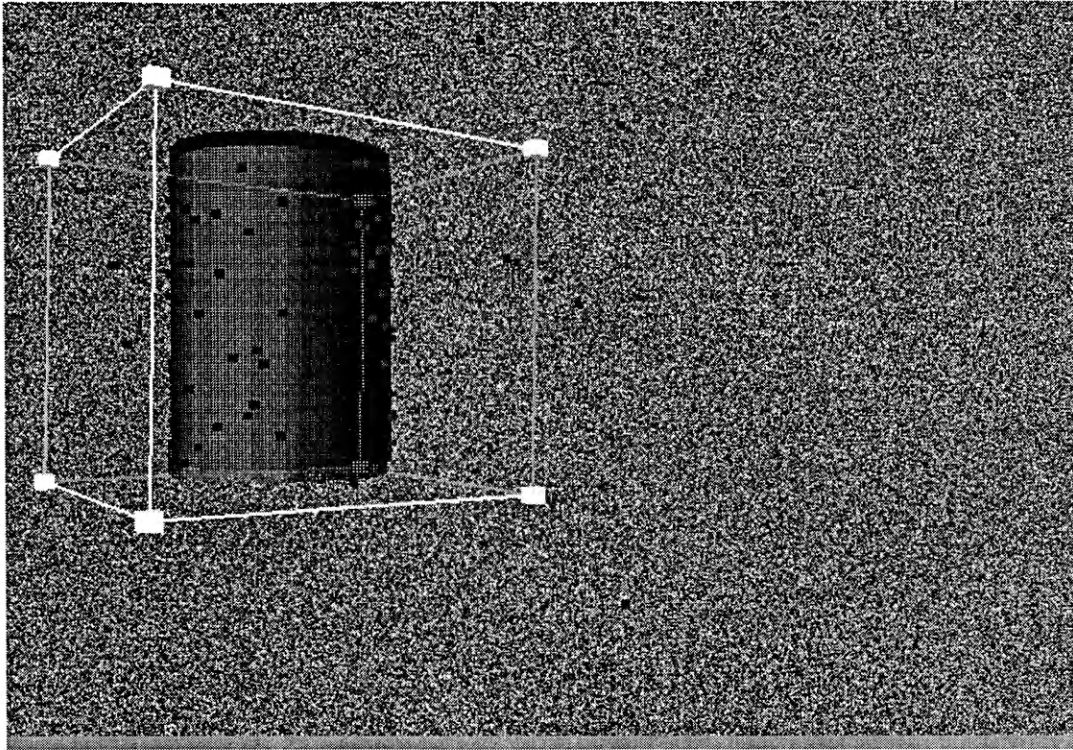
Figure 17: A completed site model

Note the wireframe boxes surrounding the objects. These boxes enable manipulation of the primitives via a 2-D mouse as opposed to a 6 DOF manipulation device.

Incomplete data, which can occur when using stereo range data, presents a problem for a system like ours. Although our application performs an accurate fit to sparse data, data that is not well-distributed can cause a poor approximation to the pose of the object itself. Figure 18 demonstrates an accurate fit to the data, but a much less accurate fit to the object. The poorer fit results from a lack of range points gathered from the top rim of the cylinder as well as the presence of a single, “faulty” point immediately below it. With more range data on the top rim, the object would not have the freedom to move vertically without increasing the distance from its surface to the range points. This shortcoming suggests that another source of range data could be beneficial although stereo is a relatively inexpensive source.

Another solution could be the use of *shared* control<sup>7</sup>. We already have some of these measures installed in the form of “constraints”. *Shared* control is just as one might expect: the human operator controls one or more degrees of freedom, while the machine simultaneously manipulates the remaining degrees. Our “constraints” were implemented by allowing the operator to specify the degrees constrained and the values desired, then adding a penalty to the objective function if the constraints were not followed. This improvement has not been thoroughly tested and we are still exploring alternatives in our

implementation. In the figure, a vertical constraint could have been used to fit the object pose more accurately.



**Figure 18: An example of how incomplete data can skew a model's fit**

Another possible improvement would be to add scaling to the state vector. For instance, allow the fitting routine to manipulate the radius of a cylinder as well as its position and orientation. This would give us a wider variety of models to work with. The addition of cones and spheres to the models already available would add to the variety, as well. Ultimately, it would be useful to have more “generic models which can represent a wider variety of shapes, such as superquadrics<sup>31</sup>.

The addition of “smart” subroutines to the automated portion of the system might also improve performance. Contrast, edge detection, and other visual cues could help segment objects from the background and thus, aid fitting. Use of the SGI stereo display feature could aid operator depth perception and, thus, improve interaction. Similarly, a simulated annealing schedule which is optimized for the site modeling problem (if such a schedule exists) could aid performance, as well. Also, we may find a more suitable algorithm than simulated annealing to further decrease user interaction and total task time.

Finally, the ultimate goal of the project is to integrate the system with an operational robotics environment. This would necessitate the development of a completed site model representation that is understandable by the robots in that particular environment.

Rewriting of the application code has begun to facilitate this goal. Particularly, we wish to bring about a more definitive separation of the user interface from the modeling code. In this way, we can keep the modeling routine the same and build user interfaces that work best with the given robotic environments.

## REFERENCES CITED

- [1] S. Heuel and R. Nevatia, "Including Interaction in an Automated Modeling System," *Proc. of Image Understanding Workshop*, Morgan Kaufmann, Palm Springs, pp. 429-434, 1996.
- [2] MTI, "Measurement and Diagnostics: Precision Mapping, Visualization, and Modeling Systems," . <http://www.mechtech.com>.
- [3] W. E. L. Grimson, T. Lozano-Perez, W. M. W. III, G. J. Ettinger, S. J. White, and R. Kikinis, "An Automatic Registration Method for Frameless Stereotaxy, Image Guided Surgery, and Enhanced Reality Visualization," *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, Seattle, pp. 430-436, 1994.
- [4] Y. Roth-Tabak and R. Jain, "Building an Environment Model Using Depth Information," *Computer Magazine (IEEE)*, Vol. 22, No. 6, pp. 85-90, 1989.
- [5] M. Hebert, R. Hoffman, A. Johnson, and J. Osborn, "Sensor-Based Interior Modeling," *Proc. of Robotics and Remote Systems*, American Nuclear Society, Monterey, CA, pp. 731-737, 1995.
- [6] R. Nevatia, "USC RADIUS Related Research: An Overview," *Proc. of Image Understanding Workshop*, Vol. 1, Morgan Kaufmann, Palm Springs, pp. 317-323, 1996.

- [7] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*, Cambridge, Massachusetts, MIT Press, 1992.
- [8] M. J. McDonald and R. D. Palmquist, "Graphical programming: On-line Robot Simulation for Telerobotic Control," *Proc. of International Robots and Vision Automation Conference*, pp. 22-59..22-73, 1993.
- [9] C. Wang and D. J. Cannon, "Virtual Reality-Based Point-and-Direct Robotic Inspection in Manufacturing," *IEEE Transactions in Manufacturing*, Vol. 12, No. 4, pp. 516-530, 1996.
- [10] Vexcel, "Product Profile: FotoG-FMS (tm) Industrial Photogrammetry," .  
<http://www.vexcel.com>.
- [11] A. Leonardis, A. Gupta, and R. Bajcsy, "Segmentation of Range Images as the Search for Geometric Parametric Models," *International Journal of Computer Vision*, Vol. 14, No. 3, pp. 253-277, 1995.
- [12] R. Krishnapuram and D. Cassent, "Determination of Three-Dimensional Object Location and Orientation from Range Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 11, pp. 1158-1167, 1989.
- [13] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 2, pp. 239-256, 1992.



- [14] Y. Hsieh, "Design and Evaluation of a Semi-Automated Site Modeling System," *Proc. of Image Understanding Workshop*, Vol. 1, Morgan Kaufmann, Palm Springs, CA, pp. 435-459, 1996.
- [15] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, Vol. 24, No. pp. 381-395, 1981.
- [16] S. P. Banks and R. F. Harrison, "Simple Object Recognition by Neural Networks: Application of the Hough Transform," *International Journal of Control*, Vol. 54, No. 6, pp. 1469-1476, 1991.
- [17] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed., Cambridge University Press, 1992.
- [18] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*, New York, McGraw-Hill, 1995.
- [19] S. Kirkpatrick, J. C.D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, pp. 671-680, 1983.
- [20] F. Durbin, J. Haussy, G. Berthiau, and P. Siarry, "Circuit Performance Optimization and Model Fitting Based on Simulated Annealing," *International Journal of Electronics*, Vol. 73, No. 6, pp. 1267-1271, 1992.
- [21] B. Bhanu and S. Lee, *Genetic Learning for Adaptive Image Segmentation*, Norwell, Mass., Kluwer Academic, 1994.

- [22] N. Ayache and F. Lustman, "Fast and reliable passive trinocular stereo vision," *Proc. of First Intl. Conf. Computer Vision*, IEEE, pp. 422-427, 1987.
- [23] W. A. Hoff, F. W. Hood, and R. King, "An Interactive System for Creating Object Models from Range Data Using Simulated Annealing," *Proc. of International Conference on Robotics and Automation*, Albuquerque, NM, 1997.
- [24] K. H. Hoffman and P. Salamon, "The Optimal Simulated Annealing Schedule for a Simple Model," *Journal of Physics A: Mathematical and General*, Vol. 23, No. 15, pp. 3511-3523, 1990.
- [25] J. Craig, *Introduction to Robotics, Mechanics, and Control*, 2nd ed., Addison Wesley, 1990.
- [26] J. Wernecke, *The Inventor Mentor: Programming Object-Oriented 3-D Graphics with Open Inventor*, 2 ed., Reading, Mass., Addison Wesley, 1994.
- [27] W. Hoff, L. Gatrell, and J. Spofford, "Machine Vision Based Teleoperation Aid," *Telematics and Informatics*, Vol. 8, No. 4, pp. 403-423, 1991.
- [28] K. K. Hockman and M. W. Jenkins, "Design of Experiments: Neglected Key to Competitive R&D," *Industrial Engineering*, 1994, pp. 50-51.
- [29] W. Knotts, "Problem Solving Using Experimental Design Techniques," *Ceramic Engineering Science Proceedings*, Vol. 16, No. 3, pp. 119-122, 1995.
- [30] R. R. Johnson, *Elementary Statistics*, 3 ed., Boston, Duxbury Press, 1980.

[31] R. Bajcsy and F. Solina, "Three Dimensional Object Representation Revisited," *Proc. of First International Conference on Computer Vision*, Computer Society Press, London, pp. 231-240, 1987.

## APPENDIX

//Appendix Code

```

/*****
* Function: funk_cyl
* Edited by : Dr. W. Hoff, Fred Hood
* Description:
*   Evaluates fit of a primitive to the data points.
*   For now, just do translation, not rotation
*   Fit is defined as:
*     NPOINTS/dmin - sum [ 1 / max(dmin, d) ]
*   where d = dist to surface
*****/
float funk_cyl(float p[])
{
    static int nPoints=0;           // number of points
    static float points[MAXPOINTS][3]; // array of points
    float f;                        // total error score
    float x0 = p[1], y0 = p[2], z0 = p[3]; /* primitive center */
    float ax, ay, az;                /* primitive axis */
    float H_c_w[4][4];
    float closest[3];                // closest point to given point
    float kx, ky, kz, theta;         /* unit axis, angle */
    float d;                         // distances from point
    int i, debug=-1;
    float cnst_error_score=0;
    float X0, Y0, Z0, AX, AY, AZ;
    SbVec3f posVec, rotVec;
    float rotAngle;
    AppBB *myBB=getCurrentBB();
    SbRotation rVec, rot;
    Cylinder *Cyl=(Cylinder *)myBB->currentModel;

    // read points if haven't already
    if (myBB->ptflag==0) {
        ReadData(points, nPoints);
        myBB->ptflag=1;
    }

    // set initial error score
    f = G * nPoints / DIST_MIN;

```

```

#if DO_ORIENT
    /* Get the rotation from the p vector */
    ax = p[4];
    ay = p[5];
    az = p[6];
    angleTOaxangle(ax, ay, az, theta);
    rVec.setValue(SbVec3f(ax, ay, az), theta);
#else
    /* Get the rotation from the current pose */

    rVec=myBB->pinfo.pnodes.primitive_xForm->rotation.getValue();

#endif
    // axis conversion to a 3D vector
    axConvert(rVec, ax, ay, az);
    // output parameters for debugging
    if (debug >= 2) {
        printf("In funk_cyl: x0, y0, z0, ax, ay, az = %f, %f, %f, %f, %f, %f\n",
            x0, y0, z0, ax, ay, az);
    }

    /*
    * Force the x,y,z location to be within some sane bounds. We'll do this
    * by abruptly changing the error function to some large value if we
    * exit this bounding box.
    */
    if (x0 > MAX_BOUND || y0 > MAX_BOUND || z0 > MAX_BOUND ||
        x0 < -MAX_BOUND || y0 < -MAX_BOUND || z0 < 0)
        return (100000000.0);

    /*
    * Figure out how to rotate the world frame to align with the
    * cylinder frame (Calculate H_c_w). This can be done by rotating the y axis about k
    * to align it with the cylinder axis. The axis of rotation for this is given
    * by the cross product of y x a. The cross product of y x a is
    * just (az, 0, -ax). Now normalize this vector.
    */
    float xzmag = sqrt(az*az + ax*ax);
    if (xzmag < 1e-5) {
        kx = 0.0;
        ky = 1.0;
        kz = 0.0;
    } else {
        kx = az / xzmag;
        kz = -ax / xzmag;
        ky = 0.0;
    }

```

```

}
theta = acos(ay);

/*
 * Figure out H_c_w; i.e., the transformation matrix to transform world points
 * to cylinder points. These equations are from Craig's book on robotics.
 */
float vt = 1 - cos(theta);
float ct = cos(theta);
float st = sin(theta);
H_c_w[0][0] = kx*kx*vt + ct;
H_c_w[0][1] = kx*ky*vt - kz*st;
H_c_w[0][2] = kx*kz*vt + ky*st;

H_c_w[1][0] = kx*ky*vt + kz*st;
H_c_w[1][1] = ky*ky*vt + ct;
H_c_w[1][2] = ky*kz*vt - kx*st;

H_c_w[2][0] = kx*kz*vt - ky*st;
H_c_w[2][1] = ky*kz*vt + kx*st;
H_c_w[2][2] = kz*kz*vt + ct;

/*
 * Set the location of the cylinder in world coordinates.
 */
H_c_w[0][3] = x0;
H_c_w[1][3] = y0;
H_c_w[2][3] = z0;

H_c_w[3][0] = 0;
H_c_w[3][1] = 0;
H_c_w[3][2] = 0;
H_c_w[3][3] = 1;

//printf("Transformation matrix (H_c_w):\n");
//for (i=0; i < 4; i++)
//  printf("%f %f %f %f\n",
//    H_c_w[i][0], H_c_w[i][1], H_c_w[i][2], H_c_w[i][3]);

Cyl->ComputeVisible(H_c_w);
//Cyl->Print();

for (i=0; i < nPoints; i++) {

    // Find closest point on primitive
    Cyl->ClosestPoint(points[i], closest, d, i);

```

```

    if((myBB->lineFlag)&&(i==10))
        myBB->addLine(points[i], closest);

    // output straight line distance from point to primitive
    if (debug >= 3)    printf("  d = %f\n", d);

    // calculate error score for point and subtract from total
    f -= G / MAX(DIST_MIN, d);
}

// output total error score
f=1000*(f/(G*nPoints/DIST_MIN));

if(myBB->pinfo.constraintOn==1){
    posVec=myBB->pinfo.pnodes.constraint_xForm->translation.getValue();
    rot=myBB->pinfo.pnodes.constraint_xForm->rotation.getValue();
    posVec.getValue(X0, Y0, Z0);
    axConvert(rot, AX, AY, AZ);
    switch(myBB->pinfo.crossOn){

        case(D2):      cnst_error_score=constraint2D(x0, y0, z0, X0, Y0, Z0);
                        break;
        case(D3):      cnst_error_score=constraint3D(x0, y0, z0, X0, Y0, Z0);
                        break;
        case(ROTATE):  cnst_error_score=constraintOrientCyl(ax, ay, az, AX, AY, AZ);
                        break;
        case(NONE):    break;
    }
}

if (cnst_error_score!=0){
    f=cnst_error_score;
}

if (myBB->mytimer->>manualFlag)
    myBB->writeErrorText(f);
if (debug >= 2)    printf("score = %f\n", f);
if(f<=1000)
    ((SoMaterial*)(myBB->pinfo.currentPrimSep->getChild(1)))->
        diffuseColor.setValue((f/1000), (1-f/1000), 0);
else
    ((SoMaterial*)(myBB->pinfo.currentPrimSep->getChild(1)))->
        diffuseColor.setValue(1, 0, 0);

// return total error score
return f;
}

```

```

/*****
* Function: funk_box
* Edited by : Dr. W. Hoff, Fred Hood
* Description:
* Evaluates fit of a primitive to the data points.
* Fit is defined as:
* 
$$\text{NPOINTS}/\text{dmin} - \sum [ 1 / \max(\text{dmin}, d) ]$$

* where d = dist to surface
*****/
float funk_box(float p[])
{
    static int nPoints=0;           // number of points
    static float points[MAXPOINTS][3]; // array of points
    float f;                        // total error score
    float x0 = p[1], y0 = p[2], z0 = p[3]; /* primitive center */
    float ax, ay, az, theta;         /* rotation axis, angle */
    float H_b_w[4][4];
    float closest[3];                // closest point to given point
    float d;                         // distances from point
    int i, j, debug=0;
    float cnst_error_score=0;
    float X0, Y0, Z0, AX, AY, AZ;
    SbVec3f posVec, rotVec;
    float rotAngle;
    SbRotation rVec, rot;
    SbMatrix rMat;

    AppBB *myBB=getCurrentBB();
    Box *box=(Box *)myBB->currentModel;

    // read points if haven't already
    if (myBB->ptflag==0){
        ReadData(points, nPoints);
        myBB->ptflag=1;
    }

    // set initial error score
    f = G * nPoints / DIST_MIN;

#ifdef DO_ORIENT
    /* Get the rotation from the p vector */
    ax = p[4];
    ay = p[5];
    az = p[6];

    // output parameters for debugging

```



```

if (debug >= 2) {
    printf("In funk_box: x0, y0, z0, ax, ay, az, theta = \n");
    printf("  %f, %f, %f, %f, %f, %f, %f\n",
        x0, y0, z0, ax, ay, az, theta);
}

angleTOaxangle(ax, ay, az, theta);
rVec.setValue(SbVec3f(ax, ay, az), theta);
#else
/* Get the rotation from the current pose */
rVec=myBB->pinfo.pnodes.primitive_xForm->rotation.getValue();
#endif
// matrix equivalent
rMat.setTransform(SbVec3f(x0, y0, z0), rVec, SbVec3f(1, 1, 1));
for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
        H_b_w[i][j] = rMat[j][i]; // have to transpose since inventor
    } // stores row-major
}

/*
 * Force the x,y,z location to be within some sane bounds. We'll do this
 * by abruptly changing the error function to some large value if we
 * exit this bounding box.
 */
if (x0 > MAX_BOUND || y0 > MAX_BOUND || z0 > MAX_BOUND ||
    x0 < -MAX_BOUND || y0 < -MAX_BOUND || z0 < 0)
    return (100000000.0);

box->ComputeVisible(H_b_w);
//box->Print();

for (i=0; i < nPoints; i++) {

    // Find closest point on primitive
    box->ClosestPoint(points[i], closest, d);

    if((myBB->lineFlag)&&(i==10))
        myBB->addLine(points[i], closest);

    // output straight line distance from point to primitive
    if (debug >= 3) printf("  d = %f\n", d);

    // calculate error score for point and subtract from total
    f -= G / MAX(DIST_MIN, d);
}

```

```

// output total error score
f=1000*(f/(G*nPoints/DIST_MIN));

if(myBB->pinfo.constraintOn==1){
    posVec=myBB->pinfo.pnodes.constraint_xForm->translation.getValue();
    rot=myBB->pinfo.pnodes.constraint_xForm->rotation.getValue();
    rot.getValue(rotVec, rotAngle);
    posVec.getValue(X0, Y0, Z0);
    rotVec.getValue(AX, AY, AZ);
    switch(myBB->pinfo.crossOn){

        case(D2):      cnst_error_score=constraint2D(x0, y0, z0, X0, Y0, Z0);
                        break;
        case(D3):      cnst_error_score=constraint3D(x0, y0, z0, X0, Y0, Z0);
                        break;
        case(ROTATE):  cnst_error_score=constraintOrient(ax, ay, az, theta,
                                                         AX, AY, AZ, rotAngle);
                        break;
        case(NONE):    break;
    }
}

if (cnst_error_score!=0){
    f=cnst_error_score;
}

if (myBB->mytimer->>manualFlag)
    myBB->writeErrorText(f);
if (debug >= 2)    printf("score = %f\n", f);
if(f<=1000)
    ((SoMaterial*)(myBB->pinfo.currentPrimSep->getChild(1)))->
        diffuseColor.setValue((f/1000), (1-f/1000), 0);
else
    ((SoMaterial*)(myBB->pinfo.currentPrimSep->getChild(1)))->
        diffuseColor.setValue(1, 0, 0);

// return total error score
return f;
}

```

```

/*****
* Function: Cylinder::ComputeVisible
* Created by : Bill Hoff
* Description:
*   Computes which parts of a cylinder are visible.
*   This function MUST be called BEFORE calling ClosestPoint!
*
* Input parameters:
*   H[4][4] = pose of model wrt camera
* Side effects:
*   Sets some internal data fields of this object:
*       H_mod_cam, H_cam_mod, fCapTop, fCapBot, fSide, thetaCam
*****/
void Cylinder::
ComputeVisible(float H[4][4])
{
    int    i,j, debug=0;
    float  n_cam[3], cap_cam[3], a_cam[3], dperp_cam[3], dperp_cyl[3];
    float  v_cyl[3], v_cam[3], vparallel_cam[3], vperp_cam[3];
    float  dotProd, vperp_len, r, h;

    if((debug>2))
        cout << "computing visible portions of cylinder" << endl;

/*****
* Copy the input pose to our own data structure.
* Then compute the inverse.
*****/
    for (i=0; i < 4; i++)
        for (j=0; j < 4; j++)
            H_mod_cam[i][j] = H[i][j];
    for (i=0; i < 3; i++)
        for (j=0; j < 3; j++)
            H_cam_mod[j][i] = H[i][j];    /* transpose */
    for (i=0; i < 3; i++) {
        H_cam_mod[i][3] = 0;
        for (j=0; j < 3; j++)
            H_cam_mod[i][3] -= H_cam_mod[i][j]*H[j][3];
    }

/*****
* See if we can see top cap.  The normal vector n to this cap
* is just the cylinder axis, which is the same as the y axis.
* The 2nd column of H_mod_cam is ymod_cam.
*****/
    n_cam[0] = H_mod_cam[0][1];

```

```

n_cam[1] = H_mod_cam[1][1];
n_cam[2] = H_mod_cam[2][1];

/*
 * The center of this cap is (0,height/2,0) in cyl coords.
 * Compute location of the center of cap in camera coords.
 */
for (i=0; i < 3; i++)
    cap_cam[i] = H_mod_cam[i][1]*height/2 + H_mod_cam[i][3];

/*
 * Test if we can see this cap. The angle between the
 * vectors cap_cam and n_cam must be greater than 90 degrees.
 * Or, equivalently, their dot product must be < 0.
 */
dotProd = 0;
for (i=0; i < 3; i++)
    dotProd += cap_cam[i]*n_cam[i];
if (dotProd < 0)
    fCapTop = 1;
else
    fCapTop = 0;

/*****
 * See if we can see bottom cap. Normal vector n to this cap
 * is the negative of cylinder axis, which is the -y axis.
 * The 2nd column of H_mod_cam is ymod_cam.
 *****/
n_cam[0] = -H_mod_cam[0][1];
n_cam[1] = -H_mod_cam[1][1];
n_cam[2] = -H_mod_cam[2][1];

/*
 * The center of this cap is (0,-height/2,0) in cyl coords.
 * Compute location of the center of cap in camera coords.
 */
for (i=0; i < 3; i++)
    cap_cam[i] = -H_mod_cam[i][1]*height/2 + H_mod_cam[i][3];

/*
 * Test if we can see this cap. The angle between the
 * vectors cap_cam and n_cam must be greater than 90 degrees.
 * Or, equivalently, their dot product must be < 0.
 */
dotProd = 0;
for (i=0; i < 3; i++)
    dotProd += cap_cam[i]*n_cam[i];
if (dotProd < 0)

```

```

    fCapBot = 1;
else
    fCapBot = 0;

/*****
* Identify which hemicylinder is visible, based on the pose.
* Compute the angle thetaCam in the direction of the camera.
* To do this:
*   Compute the vector v from the camera to the cyl center.
*   Find vperp, the component of v that is perpendicular
*   to the cylinder axis. This is  $v - a(v \cdot a)$ .
*   If vperp = 0, just set thetaCam = 0
*   else
*   thetaCam = the angle between -vperp and the cyl x axis
*   (which is  $\text{acos}(-vperp \cdot x)$ )
*****/
for (i=0; i < 3; i++)
    v_cam[i] = H_mod_cam[i][3];      /* vector to center */
if((debug>2))
    vector_print("v_cam", v_cam);
for (i=0; i < 3; i++)
    a_cam[i] = H_mod_cam[i][1];      /* cylinder axis */
if((debug>2))
    vector_print("a_cam", a_cam);
dotProd = 0;
for (i=0; i < 3; i++)
    dotProd += v_cam[i]*a_cam[i];
for (i=0; i < 3; i++)                /* component parallel */
    vparallel_cam[i] = dotProd*a_cam[i];
if((debug>2))
    vector_print("vparallel_cam", vparallel_cam);
for (i=0; i < 3; i++)                /* component perpendic */
    vperp_cam[i] = v_cam[i] - vparallel_cam[i];
if((debug>2))
    vector_print("vperp_cam", vperp_cam);
vperp_len = sqrt(SQR(vperp_cam[0])+SQR(vperp_cam[1])+SQR(vperp_cam[2]));
if (vperp_len < TINY) {
    thetaCam = 0;
    fSide = 0;
    return;
}
for (i=0; i < 3; i++)                /* dir perpendic */
    dperp_cam[i] = -vperp_cam[i]/vperp_len;
if((debug>2))
    vector_print("dperp_cam", dperp_cam);
for (i=0; i < 3; i++)
    dperp_cyl[i] =                      /* dir in cyl coords */

```

```

        H_cam_mod[i][0]*dperp_cam[0] +
        H_cam_mod[i][1]*dperp_cam[1] +
        H_cam_mod[i][2]*dperp_cam[2];
if((debug>2))
    vector_print("dperp_cyl", dperp_cyl);
cart_to_cyl(dperp_cyl, r, h, thetaCam);          /* get angle */

/*****
* Now, the stripe on the side of the cylinder, that consists
* of points with theta=thetaCam, should be the closest stripe
* to the camera. Test if we can see this stripe:
*   Compute the vector v to the point (RAD,0,thetaCam) in
*   cylindrical coordinates
*   Compute the surface normal n at that point.
*   The angle between v and n must be greater than 90 degrees.
*   Or, equivalently, their dot product must be < 0.
* If we can see the stripe, then we can see the hemicylinder
* defined by the angles (thetaCam-90 .. thetaCam+90).
*****/
cyl_to_cart(radius,0,thetaCam, v_cyl); /* get pt in cyl coords */
if((debug>2))
    vector_print("v_cyl", v_cyl);
for (i=0; i < 3; i++)
    v_cam[i] =                                /* get pt in cam coords */
        H_mod_cam[i][0]*v_cyl[0] +
        H_mod_cam[i][1]*v_cyl[1] +
        H_mod_cam[i][2]*v_cyl[2] +
        H_mod_cam[i][3];
if((debug>2))
    vector_print("v_cam", v_cam);
for (i=0; i < 3; i++)
    n_cam[i] =                                /* surface normal */
        H_mod_cam[i][0]*v_cyl[0] +
        H_mod_cam[i][1]*v_cyl[1] +
        H_mod_cam[i][2]*v_cyl[2];
if((debug>2))
    vector_print("n_cam", n_cam);
dotProd = 0;
for (i=0; i < 3; i++)
    dotProd += v_cam[i]*n_cam[i];
if (dotProd < 0)
    fSide = 1;
else
    fSide = 0;
}

```

```

/*****
* Function: Cylinder::ClosestPoint
* Created by : Bill Hoff
* Description:
*   Computes which parts of a cylinder are visible.
*
* Input parameters:
*   P0_cam[3] = input point in camera coords
* Output parameters:
*   P1_cam[3] = closest point on cylinder, in camera coords
*   dClosest = distance to closest point
*
* Detailed description:
*   Given a point and the cylinder pose, find which point on the cylinder
*   it is closest to. Specifically, given:
*   P0_c = (x0_c, y0_c, z0_c) = point in camera coords
*   H_m_c = pose of model to camera
*   fCapTop, fCapBot, fSide = flags for which patch we can see
*   thetaCam - angle towards camera
* Find:
*   P1_c = (x1_c, y1_c, z1_c) = closest point on cylinder, cam coords
*   d = distance to that point
* Our cylinder is actually only half a cylinder. At most one cap is
* visible, and at most half of the curved surface. Parts:
*   Surfaces:   side, cap
*   Curves: edge (2), half rim, full rim
*   Points: corner (up to 4)
*****/
void Cylinder::
ClosestPoint(float P0_cam[3], float P1_cam[3], float &dClosest, int index)
{
    int      i, debug=0;
    int      fCap, fInsideR, fInsideH, fFront;
    float     P0_mod[3], P1_mod[3];
    float     r, h, theta;          /* P0 in cylindrical coords */
    float     r1, h1, theta1;       /* P1 in cylindrical coords */
    float     hCap, hRim;

    if((debug>2)&&(index==10)){
        cout << "computing closest point to cylinder" << endl;

        cout << "input point (cam coords):" << endl;

        for (i=0; i < 3; i++) cout << P0_cam[i] << " ";

        cout << endl;
    }
}
/*****

```

```

* Transform point P0_cam to P0_mod, and then from cartesian
* coords to cylindrical coords (r,h,theta).
*****/
for (i=0; i < 3; i++)
    P0_mod[i] =
        H_cam_mod[i][0]*P0_cam[0] +
        H_cam_mod[i][1]*P0_cam[1] +
        H_cam_mod[i][2]*P0_cam[2] +
        H_cam_mod[i][3];
if((debug>2)&&(index==10)){
    cout << "input point (cyl coords):" << endl;
    for (i=0; i < 3; i++) cout << P0_mod[i] << " ";
    cout << endl;
}
cart_to_cyl(P0_mod, r, h, theta);
if((debug>2)&&(index==10)){
    cout << " Input pt (cyl coords): r = " << r << " h = " << h;
    cout << " theta = " << theta << endl;
}

/*****
* Calculate some flags indicating whether the point lies within
* the bounds of the cylinder's individual degrees of freedom.
*****/
fInsideR = (r < radius ? 1 : 0);
fInsideH = (fabs(h) < height/2 ? 1 : 0);
fFront = (diffAngle(theta, thetaCam) < M_PI/2 ? 1 : 0);

if (fCapTop) {
    hCap = height/2; hRim = -height/2; fCap = 1;
} else if (fCapBot) {
    hCap = -height/2; hRim = height/2; fCap = 1;
} else
    fCap = 0;

if((debug>2)&&(index==10)){
    cout << "fCap = " << fCap << endl;
    cout << "fSide = " << fSide << endl;
    cout << "fInsideR = " << fInsideR << endl;
    cout << "fInsideH = " << fInsideH << endl;
    cout << "fFront = " << fFront << endl;
}

/*****
* Form a number out of the combination of flags:
* (fCap, fSide, fInsideR, fInsideH, fFront)
* Then process the combination that we have.
* This switch statement calculates r1, h1, theta1.
* Initialize r1, h1, theta1 to some far away point.

```



```

*****/
r1 = 5;
h1 = 5;
theta1 = 0;
int cc = 0;
switch (combination) {
    case 0x00000:
    case 0x00001:
    case 0x00010:
    case 0x00011:
    case 0x00100:
    case 0x00101:
    case 0x00110:
    case 0x00111:
        /* we can't see neither side nor cap - inside cylinder??? */
        //cerr << "Hey! can't see neither side nor cap!" << endl;
        break;

    case 0x01000: /* one of the 4 corners */
    case 0x01100:
        GetClosestPtCyl(r,h,theta, r1,h1,theta1,
            radius,-height/2,thetaCam-M_PI/2, /* first corner */
            radius,-height/2,thetaCam+M_PI/2, /* second corner */
            radius,height/2,thetaCam-M_PI/2, /* third corner */
            radius,height/2,thetaCam+M_PI/2 /* fourth corner */
        );
        break;

    case 0x01001: /* either half rim */
    case 0x01101:
        GetClosestPtCyl(r,h,theta, r1,h1,theta1,
            radius,height/2,theta, /* pt on first half rim */
            radius,-height/2,theta /* pt on second half rim */
        );
        break;

    case 0x01010: /* one of the edges */
    case 0x01110:
        GetClosestPtCyl(r,h,theta, r1,h1,theta1,
            radius,h,thetaCam-M_PI/2, /* pt on first edge */
            radius,h,thetaCam+M_PI/2 /* pt on second edge */
        );
        break;

    case 0x01011: /* side */
    case 0x01111:
        r1 = radius; h1 = h; theta1 = theta;
        break;
}
ideH*0x10 + fFront;

```

```

case 0x10000:      /* rim */
case 0x10001:
case 0x10010:
case 0x10011:
    r1 = radius;    h1 = hCap;    theta1 = theta;
    break;

case 0x10100:      /* cap */
case 0x10101:
case 0x10110:
case 0x10111:
    r1 = r;    h1 = hCap;    theta1 = theta;
    break;

case 0x11000:      /* full rim or the corners */
    GetClosestPtCyl(r,h,theta, r1,h1,theta1,
        radius,hCap,theta,          /* pt on full rim */
        radius,hRim,thetaCam-M_PI/2, /* first corner */
        radius,hRim,thetaCam+M_PI/2  /* second corner */
    );
    break;

case 0x11001:      /* either full or half rim */
    GetClosestPtCyl(r,h,theta, r1,h1,theta1,
        radius,hCap,theta,          /* pt on full rim */
        radius,hRim,theta          /* pt on half rim */
    );
    break;

case 0x11010:      /* either full rim or the edges */
    GetClosestPtCyl(r,h,theta, r1,h1,theta1,
        radius,hCap,theta,          /* pt on full rim */
        radius,h,thetaCam-M_PI/2,   /* pt on first edge */
        radius,h,thetaCam+M_PI/2    /* pt on second edge */
    );
    break;

case 0x11011:      /* side */
    r1 = radius;    h1 = h;    theta1 = theta;
    break;

case 0x11100:      /* either the cap or the corners */
    GetClosestPtCyl(r,h,theta, r1,h1,theta1,
        r,hCap,theta,          /* pt on cap */
        radius,hRim,thetaCam-M_PI/2, /* first corner */
    );

```

```

        radius,hRim,thetaCam+M_PI/2 /* second corner */
    );
    break;

case    0x11101:                /* either the cap or the half rim */
    GetClosestPtCyl(r,h,theta, r1,h1,theta1,
        r,hCap,theta,          /* pt on cap */
        radius,hRim,theta      /* pt on half rim */
    );
    break;

case    0x11110:                /* either the cap or the edges */
    GetClosestPtCyl(r,h,theta, r1,h1,theta1,
        r,hCap,theta,          /* pt on cap */
        radius,h,thetaCam-M_PI/2, /* pt on first edge */
        radius,h,thetaCam+M_PI/2  /* pt on second edge */
    );
    break;

case    0x11111:                /* either the cap or the side */
    GetClosestPtCyl(r,h,theta, r1,h1,theta1,
        r,hCap,theta,          /* pt on cap */
        radius,h,theta         /* pt on side */
    );
    break;

default:
    cerr << "Whoops! No way.. bad combination number" << endl;
} // end switch
if((debug>2)&&(index==10)){
    cout << " Closest pt (cyl coords): r1 = " << r1 << " h1 = " << h1;
    cout << " theta1 = " << theta1 << endl;
}

/*****
* We now have the closest point (r1,h1,theta1) in cylindrical coords.
* Transform it to cartesian coords and then to camera coords.
* Then calculate the distance.
*****/
cyl_to_cart(r1,h1,theta1, P1_mod);
for (i=0; i < 3; i++)
    P1_cam[i] =
        H_mod_cam[i][0]*P1_mod[0] +
        H_mod_cam[i][1]*P1_mod[1] +
        H_mod_cam[i][2]*P1_mod[2] +
        H_mod_cam[i][3];
dClosest = sqrt(
    SQR(P1_cam[0]-P0_cam[0]) +

```

```

        SQR(P1_cam[1]-P0_cam[1]) +
        SQR(P1_cam[2]-P0_cam[2])
    );
    if((debug>2)&&(index==10)){
        cout << " Closest pt (cam coords): x = " << P1_cam[0] << " y = " << P1_cam[1];
        cout << " z = " << P1_cam[2] << endl;
        cout << " Closest distance = " << dClosest << endl;
    }
}

```

```

/*****
* Function: Cylinder::GetClosestPtCyl
* Created by : Bill Hoff
* Description:
*   Computes which of 2 points is closest to the given point.
*   All points are in cylindrical coords.
*
* Input parameters:
*   r0, h0, theta0 = input point
*   r1, h1, theta1 = first candidate point
*   r2, h2, theta2 = second candidate point
* Output parameters:
*   rc, hc, thetac = closest point
*****/
void Cylinder::
GetClosestPtCyl(
    float r0, float h0, float theta0,    /* input original point */
    float &rc, float &hc, float &thetac, /* output closest point */
    float r1, float h1, float theta1,    /* first candidate point */
    float r2, float h2, float theta2     /* second candidate point */
)
{
    float P0[3], P1[3], P2[3];

    /* Transform cylindrical coords to cartesian coords */
    cyl_to_cart(r0, h0, theta0, P0);
    cyl_to_cart(r1, h1, theta1, P1);
    cyl_to_cart(r2, h2, theta2, P2);

    float d01 = sqrt(SQR(P0[0]-P1[0]) + SQR(P0[1]-P1[1]) + SQR(P0[2]-P1[2]));
    float d02 = sqrt(SQR(P0[0]-P2[0]) + SQR(P0[1]-P2[1]) + SQR(P0[2]-P2[2]));

    if (d01 < d02)    {
        rc = r1; hc = h1; thetac = theta1;
    } else    {

```

```

        rc = r2; hc = h2; thetac = theta2;
    }
}

/*****
* Function: Cylinder::GetClosestPtCyl
* Created by : Bill Hoff
* Description:
*   Computes which of 3 points is closest to the given point.
*   All points are in cylindrical coords.
*
* Input parameters:
*   r0, h0, theta0 = input point
*   r1,h1,theta1   = first candidate point
*   r2,h2,theta2   = second candidate point
*   r3,h3,theta3   = third candidate point
* Output parameters:
*   rc, hc, thetac = closest point
*****/
void Cylinder::
GetClosestPtCyl(
    float r0, float h0, float theta0,    /* input original point */
    float &rc, float &hc, float &thetac, /* output closest point */
    float r1, float h1, float theta1,    /* first candidate point */
    float r2, float h2, float theta2,    /* second candidate point */
    float r3, float h3, float theta3     /* third candidate point */
)
{
    float P0[3], P1[3], P2[3], P3[3];

    /* Transform cylindrical coords to cartesian coords */
    cyl_to_cart(r0,h0,theta0, P0);
    cyl_to_cart(r1,h1,theta1, P1);
    cyl_to_cart(r2,h2,theta2, P2);
    cyl_to_cart(r3,h3,theta3, P3);

    float d01 = sqrt(SQR(P0[0]-P1[0]) + SQR(P0[1]-P1[1]) + SQR(P0[2]-P1[2]));
    float d02 = sqrt(SQR(P0[0]-P2[0]) + SQR(P0[1]-P2[1]) + SQR(P0[2]-P2[2]));
    float d03 = sqrt(SQR(P0[0]-P3[0]) + SQR(P0[1]-P3[1]) + SQR(P0[2]-P3[2]));

    if (d01 < d02 && d01 < d03) {
        rc = r1; hc = h1; thetac = theta1;
    } else if (d02 < d01 && d02 < d03) {
        rc = r2; hc = h2; thetac = theta2;
    } else {
        rc = r3; hc = h3; thetac = theta3;
    }
}

```

```

    }
}

```

```

/*****
* Function: Cylinder::GetClosestPtCyl
* Created by : Bill Hoff
* Description:
*   Computes which of 4 points is closest to the given point.
*   All points are in cylindrical coords.
*
* Input parameters:
*   r0, h0, theta0 = input point
*   r1,h1,theta1   = first candidate point
*   r2,h2,theta2   = second candidate point
*   r3,h3,theta3   = third candidate point
*   r4,h4,theta4   = fourth candidate point
* Output parameters:
*   rc, hc, thetac = closest point
*****/
void Cylinder::
GetClosestPtCyl(
    float r0, float h0, float theta0,    /* input original point */
    float &rc, float &hc, float &thetac, /* output closest point */
    float r1, float h1, float theta1,    /* first candidate point */
    float r2, float h2, float theta2,    /* second candidate point */
    float r3, float h3, float theta3,    /* third candidate point */
    float r4, float h4, float theta4     /* fourth candidate point */
)
{
    float P0[3], P1[3], P2[3], P3[3], P4[3];

    /* Transform cylindrical coords to cartesian coords */
    cyl_to_cart(r0,h0,theta0, P0);
    cyl_to_cart(r1,h1,theta1, P1);
    cyl_to_cart(r2,h2,theta2, P2);
    cyl_to_cart(r3,h3,theta3, P3);
    cyl_to_cart(r4,h4,theta4, P4);

    float d01 = sqrt(SQR(P0[0]-P1[0]) + SQR(P0[1]-P1[1]) + SQR(P0[2]-P1[2]));
    float d02 = sqrt(SQR(P0[0]-P2[0]) + SQR(P0[1]-P2[1]) + SQR(P0[2]-P2[2]));
    float d03 = sqrt(SQR(P0[0]-P3[0]) + SQR(P0[1]-P3[1]) + SQR(P0[2]-P3[2]));
    float d04 = sqrt(SQR(P0[0]-P4[0]) + SQR(P0[1]-P4[1]) + SQR(P0[2]-P4[2]));

    if (d01 < d02 && d01 < d03 && d01 < d04)    {
        rc = r1; hc = h1; thetac = theta1;
    }
}

```

```
} else if (d02 < d01 && d02 < d03 && d02 < d04)    {  
    rc = r2; hc = h2; thetac = theta2;  
} else if (d03 < d01 && d03 < d02 && d03 < d04)    {  
    rc = r3; hc = h3; thetac = theta3;  
} else      {  
    rc = r4; hc = h4; thetac = theta4;  
}  
}
```

```

/*****
* Function: Box::ComputeVisible
* Created by : Bill Hoff
* Description:
*   Computes which parts of a box are visible.
*   This function MUST be called BEFORE calling ClosestPoint!
*
* Input parameters:
*   H[4][4] = pose of model wrt camera
* Side effects:
*   Sets some internal data fields of this object:
*       H_mod_cam, H_cam_mod,
*       fFaceW[2], fFaceH[2], fFaceD[2],
*       fEdgeWH[4], fEdgeHD[4], fEdgeWD[4];
*       fVertex[8]
*****/
void Box::
ComputeVisible(float H[4][4])
{
    int i,j;

    //cout << "computing visible portions of box" << endl;

    /*****
    * Copy the input pose to our own data structure.
    * Then compute the inverse.
    *****/
    for (i=0; i < 4; i++)
        for (j=0; j < 4; j++)
            H_mod_cam[i][j] = H[i][j];
    for (i=0; i < 3; i++)
        for (j=0; j < 3; j++)
            H_cam_mod[j][i] = H[i][j];    /* transpose */
    for (i=0; i < 3; i++) {
        H_cam_mod[i][3] = 0;
        for (j=0; j < 3; j++)
            H_cam_mod[i][3] -= H_cam_mod[i][j]*H[j][3];
    }
    for (i=0; i < 3; i++)
        H_cam_mod[3][i] = 0;                /* last row */
    H_cam_mod[3][3] = 1.0;

    /* Compute coordinates of the faces (for convenience) */
    xLeft = -width/2;
    xRight = width/2;
    yTop = -height/2;
    yBot = height/2;
    zFront = -depth/2;

```



```

zBack = depth/2;

/*****
* See which faces are visible. A face is visible iff
* its surface normal (pointing outwards) makes an angle
* of greater than 90 degrees with respect to the vector
* from the camera origin to the center of the face.
*****/

fFaceLeft = checkFaceVisible(H_mod_cam, /* left */
    -1.0, 0.0, 0.0, /* normal to face, in model coords */
    -width/2, 0.0, 0.0 /* center of face, in model coords */
);
fFaceRight = checkFaceVisible(H_mod_cam, /* right */
    1.0, 0.0, 0.0, /* normal to face, in model coords */
    width/2, 0.0, 0.0 /* center of face, in model coords */
);
fFaceTop = checkFaceVisible(H_mod_cam, /* top */
    0.0, -1.0, 0.0, /* normal to face, in model coords */
    0.0, -height/2, 0.0 /* center of face, in model coords */
);
fFaceBot = checkFaceVisible(H_mod_cam, /* bottom */
    0.0, 1.0, 0.0, /* normal to face, in model coords */
    0.0, height/2, 0.0 /* center of face, in model coords */
);
fFaceFront = checkFaceVisible(H_mod_cam, /* front */
    0.0, 0.0, -1.0, /* normal to face, in model coords */
    0.0, 0.0, -depth/2 /* center of face, in model coords */
);
fFaceBack = checkFaceVisible(H_mod_cam, /* back */
    0.0, 0.0, 1.0, /* normal to face, in model coords */
    0.0, 0.0, depth/2 /* center of face, in model coords */
);
}

/*****
* Function: Box::checkFaceVisible
* Created by : Bill Hoff
* Description:
*   Determines if a face is visible.
*
* Input parameters:
*   H[4][4] = pose of model wrt camera
*   nx, ny, nz = unit normal vector to face, model coords
*   cx, cy, cz = center of face, model coords
* Returns:
*   True (1) if face is visible, false (0) otherwise.

```

```

*****/
int Box::
checkFaceVisible(float H[4][4],
    float nx, float ny, float nz,
    float cx, float cy, float cz)
{
    float nx_cam, ny_cam, nz_cam;
    float cx_cam, cy_cam, cz_cam;

    /* Transform n, c from model to camera coords */
    nx_cam = H[0][0]*nx + H[0][1]*ny + H[0][2]*nz;
    ny_cam = H[1][0]*nx + H[1][1]*ny + H[1][2]*nz;
    nz_cam = H[2][0]*nx + H[2][1]*ny + H[2][2]*nz;

    cx_cam = H[0][0]*cx + H[0][1]*cy + H[0][2]*cz + H[0][3];
    cy_cam = H[1][0]*cx + H[1][1]*cy + H[1][2]*cz + H[1][3];
    cz_cam = H[2][0]*cx + H[2][1]*cy + H[2][2]*cz + H[2][3];

    /*
     * Face is visible iff the angle between n and c is greater than
     * 90 degrees; or equivalently, the dot product of n and c < 0.
     */
    float dotProd = nx_cam*cx_cam + ny_cam*cy_cam + nz_cam*cz_cam;
    if (dotProd < 0)    return 1;
    else                return 0;
}

/*****
 * Function: Box::ClosestPoint
 * Created by : Bill Hoff
 * Description:
 *   Computes the closest point on a box to the given point.
 *
 * Input parameters:
 *   P0_cam[3] = input point in camera coords
 * Output parameters:
 *   P1_cam[3] = closest point on box, in camera coords
 *   dClosest = distance to closest point
 *
 * Our box is actually only half a box. At most three faces, 9 edges,
 * and 7 vertices are visible.
 *****/
void Box::
ClosestPoint(float P0_cam[3], float P1_cam[3], float &dClosest)
{
    int        i;

```

```

int      fInsideW, fInsideH, fInsideD;
float    P0_mod[3], P1_mod[3];

/*****
cout << "computing closest point to box" << endl;
cout << "input point (camera coords):" << endl;
for (i=0; i < 3; i++) cout << P0_cam[i] << " ";
cout << endl;
*****/

/* Transform the given point to model coords */
for (i=0; i < 3; i++)
    P0_mod[i] =
        H_cam_mod[i][0]*P0_cam[0] +
        H_cam_mod[i][1]*P0_cam[1] +
        H_cam_mod[i][2]*P0_cam[2] +
        H_cam_mod[i][3];

/*****
cout << "input point (model coords):" << endl;
for (i=0; i < 3; i++) cout << P0_mod[i] << " ";
cout << endl;
*****/

/* For convenience, get individual x,y,z */
float x0 = P0_mod[0];
float y0 = P0_mod[1];
float z0 = P0_mod[2];

/*****
* Determine whether the x,y,z components individually
* lie within the bounds of the box.
*****/
fInsideW = (fabs(x0) < width/2 ? 1 : 0);
fInsideH = (fabs(y0) < height/2 ? 1 : 0);
fInsideD = (fabs(z0) < depth/2 ? 1 : 0);

/*****
cout << "fInsideW = " << fInsideW << endl;
cout << "fInsideH = " << fInsideH << endl;
cout << "fInsideD = " << fInsideD << endl;
*****/

/*****
* Now, look at each face individually. If the face is
* visible, we may have the shortest distance to a point on
* that face. There are three possibilities:
* 1. The shortest distance to the face is a perpendicular

```

```

*   to the face itself.
* 2. The shortest distance to the face is a perpendicular
*   to one of the edges of the face.
* 3. The shortest distance to the face is a line segment
*   to one of the vertices of the face.
* Check for these three cases, in this order.
*****/
dClosest = HUGE; /* Keeps track of minimum distance found */
P1_mod[0] = 0.0; /* Closest point */
P1_mod[1] = 0.0;
P1_mod[2] = 0.0;

if (fFaceLeft) {
    if (fInsideH && fInsideD) {
        /* Closest to the face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, y0, z0);
    } else if (fInsideH) {
        /* Closest to one of the up-down edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, y0, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, y0, zBack);
    } else if (fInsideD) {
        /* Closest to one of the front-back edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yTop, z0);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yBot, z0);
    } else {
        /* Closest to one of the vertices of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yTop, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yTop, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yBot, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yBot, zBack);
    }
} /* fFaceLeft */

if (fFaceRight) {
    if (fInsideH && fInsideD) {
        /* Closest to the face */
        PickClosest(P0_mod, P1_mod, dClosest, xRight, y0, z0);
    } else if (fInsideH) {
        /* Closest to one of the up-down edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xRight, y0, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, y0, zBack);
    } else if (fInsideD) {
        /* Closest to one of the front-back edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yTop, z0);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yBot, z0);
    } else {
        /* Closest to one of the vertices of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yTop, zFront);
    }
}

```

```

        PickClosest(P0_mod, P1_mod, dClosest, xRight, yTop, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yBot, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yBot, zBack);
    }
} /* fFaceRight */

if (fFaceTop) {
    if (fInsideW && fInsideD){
        /* Closest to the face */
        PickClosest(P0_mod, P1_mod, dClosest, x0, yTop, z0);
    } else if (fInsideW) {
        /* Closest to one of the left-right edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, x0, yTop, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, x0, yTop, zBack);
    } else if (fInsideD) {
        /* Closest to one of the front-back edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yTop, z0);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yTop, z0);
    } else {
        /* Closest to one of the vertices of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yTop, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yTop, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yTop, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yTop, zBack);
    }
} /* fFaceTop */

if (fFaceBot) {
    if (fInsideW && fInsideD){
        /* Closest to the face */
        PickClosest(P0_mod, P1_mod, dClosest, x0, yBot, z0);
    } else if (fInsideW) {
        /* Closest to one of the left-right edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, x0, yBot, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, x0, yBot, zBack);
    } else if (fInsideD) {
        /* Closest to one of the front-back edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yBot, z0);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yBot, z0);
    } else {
        /* Closest to one of the vertices of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yBot, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yBot, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yBot, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yBot, zBack);
    }
} /* fFaceBot */

```

```

if (fFaceFront)    {
    if (fInsideW && fInsideH){
        /* Closest to the face */
        PickClosest(P0_mod, P1_mod, dClosest, x0, y0, zFront);
    } else if (fInsideW)    {
        /* Closest to one of the left-right edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, x0, yTop, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, x0, yBot, zFront);
    } else if (fInsideH)    {
        /* Closest to one of the up-down edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, y0, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, y0, zFront);
    } else    {
        /* Closest to one of the vertices of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yBot, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yTop, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yBot, zFront);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yTop, zFront);
    }
} /* fFaceFront */

if (fFaceBack)    {
    if (fInsideW && fInsideH){
        /* Closest to the face */
        PickClosest(P0_mod, P1_mod, dClosest, x0, y0, zBack);
    } else if (fInsideW)    {
        /* Closest to one of the left-right edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, x0, yTop, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, x0, yBot, zBack);
    } else if (fInsideH)    {
        /* Closest to one of the up-down edges of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, y0, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, y0, zBack);
    } else    {
        /* Closest to one of the vertices of this face */
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yBot, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, xLeft, yTop, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yBot, zBack);
        PickClosest(P0_mod, P1_mod, dClosest, xRight, yTop, zBack);
    }
} /* fFaceBack */

/*****
cout << "closest point (model coords):" << endl;
for (i=0; i < 3; i++)  cout << P1_mod[i] << " ";
cout << endl;
*****/

```

```

/* Transform the closest point to camera coords */
for (i=0; i < 3; i++)
    P1_cam[i] =
        H_mod_cam[i][0]*P1_mod[0] +
        H_mod_cam[i][1]*P1_mod[1] +
        H_mod_cam[i][2]*P1_mod[2] +
        H_mod_cam[i][3];

/*****
cout << "closest point (camera coords):" << endl;
for (i=0; i < 3; i++)  cout << P1_cam[i] << " ";
cout << endl;
*****/

} // Box::ClosestPoint

/*****
* Function: Box::PickClosest:
* Created by : Bill Hoff
* Description:
* A utility function that picks the closest point to the given
* point.
*
* Input parameters:
* P0[3] = the given point
* P1[3] = the closest point so far
* dClosest = the closest distance so far (to P1)
* x,y,z = the coords of the new candidate point
* Output parameters:
* P1[3] = possibly revised closest point
* dClosest = possibly revised closest distance
*****/
void Box::
PickClosest(float P0[3],
            float P1[3], float &dClosest,
            float x, float y, float z)
{
    /* Calculate new distance */
    float d = sqrt( SQR(x-P0[0]) + SQR(y-P0[1]) + SQR(z-P0[2]) );

    if (d < dClosest)    {
        dClosest = d;
        P1[0] = x;      P1[1] = y;      P1[2] = z;
    }
} // Box::PickClosest

```