Recovery of 3-D Shape of Curved Objects from Multiple Views

by

Eugene S. Lin

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 1996

Author [illegible]
Department of Electrical Engineering and Computer Science

Certified by [illegible]
V. Michael Bove, Jr.
Associate Professor of Media Technology
MIT Media Laboratory

Accepted by [illegible]
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Recovery of 3-D Shape of Curved Objects from Multiple Views

by

Eugene S. Lin

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 1996

Abstract

Image contours in calibrated 2-D views are used to recover the outline of a curved object. This outline, projected as a cone from the center of projection, constrains the volume of the shape in 3-D. Higher-level symmetries specified by the user allow the generation of a 3-D mesh onto which image pixels are sampled to form a textured form. A graphical user interface is also presented that facilitates the input of user-supplied information.

Thesis supervisor: V. Michael Bove, Jr.
Title: Associate Professor of Media Technology, MIT Media Laboratory

Acknowledgments

So many people have helped me in my quest for a thesis, that I am sure to have forgotten sombody. Please forgive me for any omissions. Those who kept me sane in a time of madness include:

Mike Bove, my advisor, for keeping me on track throughout the year, and for a flow of new ideas that could keep me here for decades to come. His support and guidance have been invaluable to me. Without him, none of this work would have been possible..

Shawn Becker, my hero, for so much: advice, inspiration, wisdom, company, rides home, food in times of need, Wallace & Gromit, and for making me feel useful in a lab full of geniuses.

Ross Yu, for convincing me that the Simpsons and a Sega Saturn were necessary to the completion of my thesis.

Matt Antone, for putting up with me for an entire year, and for answering every question I ever asked him despite his being as busy as I was.

Jeff Wong, fellow PDA user and gamer, for always giving me a reason to stop working. His dedication to his work is truly an inspiration (as well as his blading ability).

Jim Clemens, for making me realize that a thesis isn't the most important thing in the world.

Stefan Agamanolis and Chris Verplaetse, for creating the most entertaining office in the world right in front of my workstation.

Henry Holtzman, for the use of the Kodak digital camera, and for moving my workstation to new and exotic locations in times of dire need.

The students of VisMod, for their cooperation in my unofficial administration of the garden SGI machine. My apologies to future users of blue-velvet.

Dave Rahn and Minh Le, my roommates and friends, for making my home a happy one.

My family, for always being there when I needed comforting, or just someone to talk to at all hours of the night.

Joanna, my wonderful girlfriend, for making my life perfect in every way during this project. Her support is responsible for my completing this thesis at all.

The research presented in this thesis was supported by
the Television of Tomorrow Consortium.

Table of Contents

Table of Figures

1. Introduction

Recently, the demand for quick, inexpensive ways of generating 3-D models for computer graphics has been growing at a rapid pace. As the use of virtual environments and computer-rendered visuals in news and entertainment proliferates, the creation of these models is becoming the bottleneck in the production pipeline. This is especially true on the World Wide Web, where sites featuring navigable environments described in Virtual Reality Modeling Language (VRML) are beginning to grow in popularity.

Traditionally, computer graphics models have been created manually through the use of CAD-like software tools. This method, analogous to the creation of 2-D "drawings", suits the problem well when the model is not required to accurately represent a physical object. On the other hand, when what is desired is a 3-D "photograph" of a real-world object, CAD approaches can be expensive in terms of time and human labor. The direct capture of 3-D data (i.e. laser scanners) is a viable alternative when economics permit, such as in Hollywood special effects applications, but such brute-force approaches are not yet within the reach of more moderate budgets.

The capture of 2-D images, on the other hand, is by now a trivial task. Photography in all its forms has been around long enough that one can obtain an image of just about any scene with a minimal outlay of equipment and effort. It would be desirable, then, to be able to recover a 3-D model directly from ordinary 2-D images. If the incremental effort to transform a set of 2-D images to a 3-D model is small enough, such a system would make 3-D "snapshots" as accessible as 2-D photos are now.

This thesis describes a series of experiments into recovering the shape of certain 3-D curved objects from calibrated images. It presents the design, implementation, and use of a semi-automatic modeling system that can recover the 3-D shape of solids of rotation. The system takes advantage of high-level symmetry information specified apriori by the user. Also presented is a user-interface tool that allows a user to provide the information necessary for shape recovery from a set of uncalibrated images. The user interface was originally designed for use with the vision-assisted modeling process developed by Shawn Becker. This modeling process allows estimation of camera parameters from uncalibrated views and hence, is a good companion to the 3-D shape recovery methods presented here.

The remaining chapters are organized as follows:

Section 2 discusses some topics necessary for the understanding of this thesis. It describes the camera model used for the projection from 3-D into 2-D and summarizes the vision-assisted modeling process that generates the input for this system. It also goes over in moderate detail the ModelMaker program, which is a user-interface tool that has been extended for use with this system. Finally, the section presents some definitions of contours and generalized cylinders, and sites some previous works in the area of shape-from-contour that are of relevance to this thesis. Section 3.1 presents an algorithm for recovering the shape of a solid of revolution from two manually specified extremal contours and describes the process of annotating a 2-D/3-D scene for analysis. Section 3.2 describes a shape-from-contour algorithm for recovering the shape of a solid of revolution from detected edge pixels. Section 3.3 improves on the algorithm in 3.2 using the same input data. Section 3.4 describes a generalization of the algorithm in 3.4 to handle non-cylindrical objects. Sections 4 and 5 present some recovered models and discuss some of the limitations of the system, as well as speculate on future work.

2. Background

2.1 Camera parameters

To accurately extract 3D information from 2D images, information about the cameras that produced the images needs to be available. Measuring the position and orientation of the camera directly is a cumbersome process that requires special equipment and also precludes the use of images that were taken for reasons other than the experiment in question. The focal length of the camera, as well, needs to be known to invert the projection onto image space. This, too, places limits on the images that can be used for shape extraction. It is desirable to be able to obtain this camera information indirectly, by analyzing the 2D images alone. The 3-D geometry extraction system introduced in this paper requires pre-calibrated cameras. However, there are other computer vision systems that can estimate the required cameras from uncalibrated views. Becker and Bove's method [1] of semi-automatic model extraction, which is used extensively in this project, is one such system. It has the added capability of negating the affect of lens distortion, to produce distortion-free images regardless of the lenses used to create the original photographs. For the remainder of this paper, I will assume that the effects of lens distortion have been nullified in any images used as input.

2.2 Camera model

The camera model used in this project is the virtual framework used in a number of computer vision experiments. The camera is represented as a center of projection (COP) with a 3D world position, and a pyramidal view volume with its zenith at the

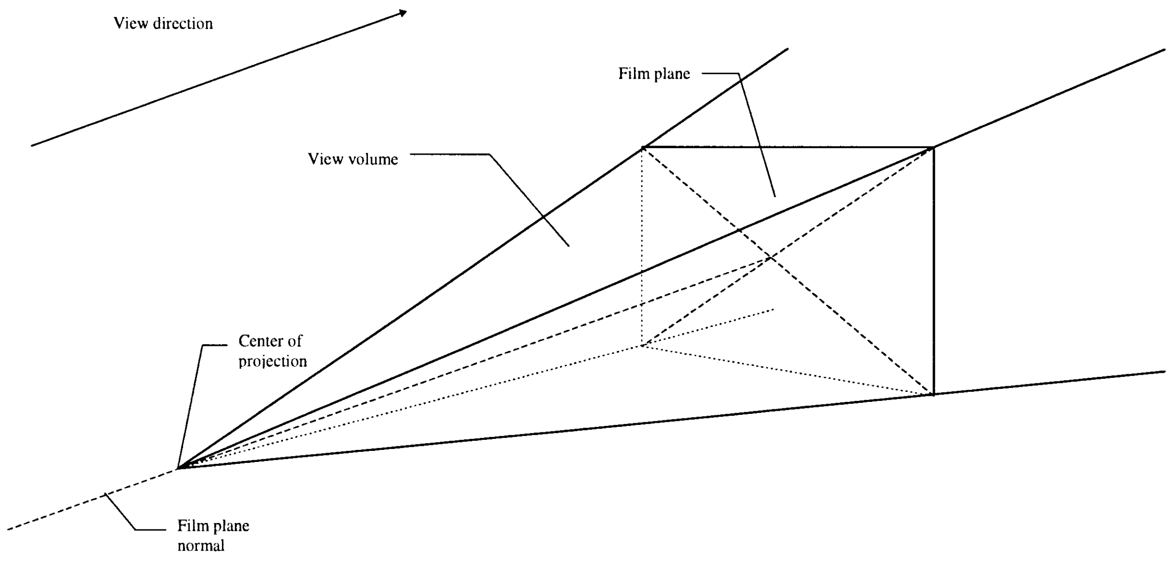


Figure 1: Camera model

COP. The film plane is located within the view volume, with a normal vector pointing toward the COP. The principal point is located at the center of the film plane. This latter restriction only approximates the behavior of physical cameras, but is accepted to allow the use of Open Inventor for 3D rendering[i].

2.3 Open Inventor

Open Inventor[2] is an object-oriented 3-D toolkit for the development of interactive 3-D graphics applications. This thesis makes heavy use of Open Inventor and the tools it provides. The Inventor toolkit is based on OpenGL, and currently runs on IRIX with X11/Motif and Microsoft Windows. It provides a set of classes for

[i] Open Inventor uses a camera model that fixes the principal point at the center of the film plane.

representing interactive 3-D scenes as a collection of objects. It also provides viewing widgets and camera abstractions for rendering 2-D views of Inventor scenes.

2.4 Vision-assisted modeling

To obtain the calibrated cameras necessary for shape extraction, I used a system developed by Shawn Becker and V. Michael Bove Jr. that can, given a set of uncalibrated 2-D images, estimate the internal and external parameters of the respective cameras that produced them. It introduces a shape-and-texture-from-wireframe algorithm to generate a textured 3-D model of the scene from a 2-D scene description. This description contains a set of digitized images and annotated lines and points that are bound to high-level geometric constraints, describing parallel constraints, an arbitrary world frame and scale, coplanar constraints, and optional additional line correspondences.

In the first phase of the model extraction process, these constraints are specified by the user through the use of an interface tool, two variations of which are discussed later in this paper. My choice to use this modeling process for camera calibration is based largely on the fact that it too, is a semi-automatic process, relying on the user to provide some of the high-level constraints in the 2-D scene description. This user-interface portion of the process has been extended by the author to provide the additional data needed for curved shape extraction. The second and third phases consist of structure-and-motion recovery, and multi-view texture recovery, respectively. These two final phases together make up the new shape-and-texture-from-wireframe algorithm. Analysis of a scene using this vision-assisted modeler produces a textured model consisting of planar surfaces, 3-D edges, and 3-D point features. The algorithms described in this paper continue the shape recovery process, extracting cylinders and other non-planar shapes from the combined 2-D and 3-D scene description.

The following subsections outline the 2-D information required by the vision-assisted modeler. This information will be augmented later to satisfy the input constraints of the curved-shape extraction process.

2.4.1 Images and lines

The first step in generating a 3-D model of a scene is to add to the 2-D description one or more digitized images of any size, orientation, focal length, distortion, or exposure. Next, lines are detected on each image using the Canny[3] edge detection algorithm and a line detection algorithm[4].

2.4.2 Parallel constraints

Lines which are projections of parallel 3-D edges in the scene need to be assigned to the same *direction* attribute. A direction is a class that describes the 3-D world-relative direction of a set of parallel edges in the scene after structure-and-motion recovery. In some cases, lines with a common vanishing point can be automatically be grouped into the same direction, but user intervention is required to correspond entire direction sets between views. To establish a coordinate frame for the scene, at least three non-coplanar directions must be assigned arbitrary world-relative vectors. To recover camera position for each image, one line of at least two known directions in each image must be given an arrow, indicating the heading of the line's direction vector as viewed in its respective image.

2.4.3 Coplanarity constraints

All 2-D lines which are views of coplanar 3-D edges must be manually assigned to the same *surface* attribute. A surface is an infinite 3-D plane which after structure-and-motion analysis has a fixed 3-D position and orientation. This user-specified assignment between images to the same surface provides some correspondence between lines in different images. Lines which are assigned to two surfaces mark the hinge, or intersection between two planes. Scenes such as this are sometimes called *origami* scenes. These hinges are used in the structure-and-motion recovery stage to fix surfaces in the 3-D scene.

2.4.4 SceneAnalyze

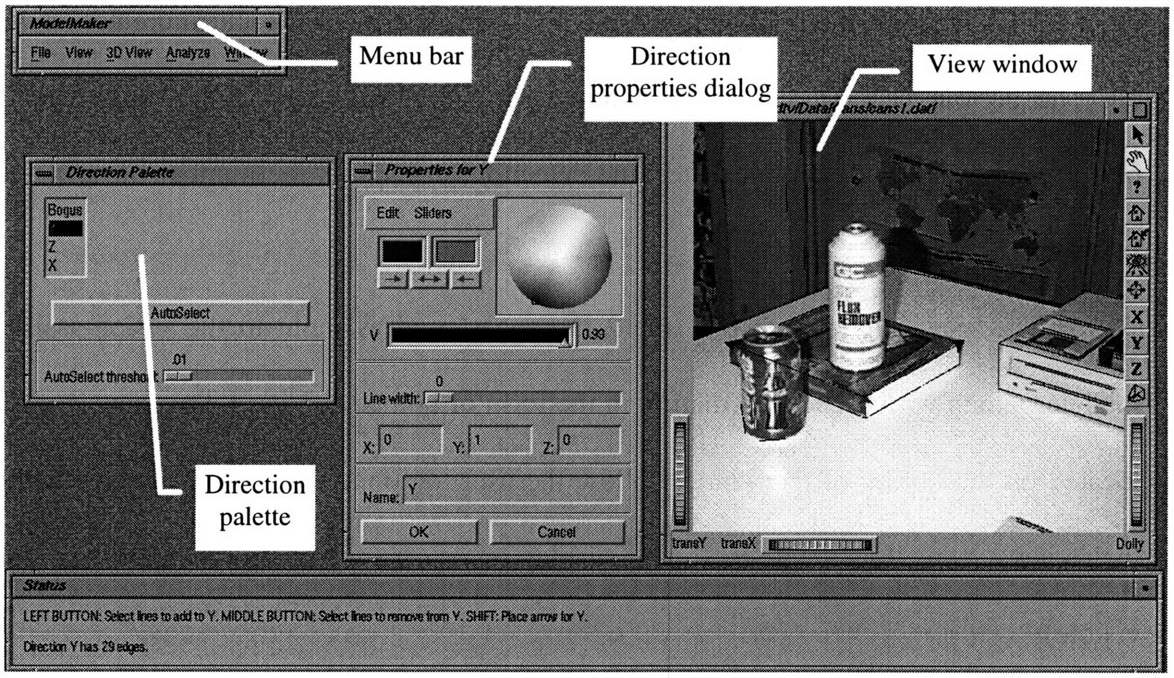
The batch analysis portion of the vision-assisted modeling algorithm has been implemented by Shawn Becker in the sceneAnalyze program. It takes as its input an *element file* containing 2-D scene information created from a user-interface tool. Its output is an element file containing the same 2-D information as well as 3-D descriptions of textured planar surfaces.

2.5 SceneBuild

Shawn Becker has developed an X11 interface for creating 2-D origami scenes from one or more digitized images. It uses a GNU-ReadLine[5] shell to receive and interpret commands, which are sent to a server portion of the program that maintains the model database as well as multiple view windows.

2.6 ModelMaker

ModelMaker is an interface tool developed by the author for the purpose of creating the origami scenes necessary for the vision-assisted modeling system described in section 2.3. It is based on Open Inventor and Motif, and makes heavy use of the 3-D object manipulation tools included with Inventor. For the purposes of this project, ModelMaker has been extended to allow the user to input the additional information needed (beyond that required for vision-assisted modeling) for curved shape extraction. The details of these extensions to ModelMaker are described in sections 3.3.1 and 3.4.1, each accompanying a description of the respective curved shape recovery algorithm that uses the new information. The following subsections describe the basics of the ModelMaker interface for vision-assisted modeling. Section 2.6.1 outlines the basic layout of the GUI presented to the user. Sections 2.6.2 through 2.6.6 outline the tools provided by ModelMaker for the major tasks involved in the vision-assisted modeling process: line detection, line editing, direction assignment, feature placement, and surface assignment.



LEFT BUTTON: Select lines to add to Y. MIDDLE BUTTON: Select lines to remove from Y. SHIFT: Place arrow for Y.

Direction Y has 29 edges.

Figure 2: ModelMaker user interface

2.6.1 Basics

There are four main parts to the ModelMaker user interface: the menu bar, the status window, the view windows, and the choosers. All four are floating windows that can be individually placed and sized on the screen.

The menu bar makes available all commands relating to the model database as a whole, including loading/saving models, setting global options, setting vision-assisted modeling analysis parameters, and running phases two and three of the vision-assisted modeling process.

The status window provides information to the user about the current controls and feedback on actions performed. It is divided horizontally into two panes. The top pane (the help window) displays help on what commands are available at the time through the mouse and/or keyboard. As the user moves the mouse over scene structures or makes other GUI selections, the help window tells the user exactly what each future action will do. The bottom pane (the message window) provides feedback on the most recently performed action. For example, after line detection, it displays the number of new lines that were detected.

The view windows display the 2-D images used as input to the vision-assisted modeler, along with scene information in the form of points and lines. The view controls located on the border of the window allow the user to zoom and pan the images within their views, as well as toggle the image display on or off (making points and lines more visible).

The choosers consist of the *direction chooser*, the *feature chooser*, the *surface chooser*, and the *image chooser*. All four choosers display a list of element names. Clicking the left button on a name either makes the element of that name the current one. Clicking the right button brings up a menu of options for the current element. Individual element parameters can be adjusted in pop-up *properties dialogs*, accessible from the pop-up menus in the choosers. In the case of the image chooser, clicking on an image's name toggles its view visible/invisible.

2.6.2 Line detection

The only part of the line detection process that involves the user is the setting of parameters for the Canny edge detector and the line detector. These are adjusted through a set of sliders in the *line detection dialog*. Because the line detection process can be time-consuming. The user has the option of highlighting a rectangular region of the image for feedback during parameter adjustment. If a region is selected, the line detector will be executed on only the pixels in the region while parameters are adjusted. Once the parameters have been set to the user's satisfaction, the line detector can be executed on the entire image.

2.6.3 Line editing

Line detection may not always produce the desired results. Therefore, ModelMaker includes the ability to manually edit the resulting lines. By using the mouse, the user can add/move/delete lines in any image. While editing the endpoint of a line, ModelMaker will optionally "snap" the endpoint position to nearby lines, facilitating the creation of precise corners.

2.6.4 Direction assignment

In direction mode, ModelMaker displays the direction chooser, which allows the user to create/edit/delete directions and select the current direction. Directions are edited via a pop-up *direction properties* dialog, which gives access to parameters such as the direction's 3-D world vector and the color in which its lines are displayed. In the view windows, the user can highlight an arbitrarily shaped region to select lines to add/remove from the current direction.

2.6.5 Feature placement

In feature mode, ModelMaker allows the user to create/edit/delete views of a feature across images. Feature positions "snap" to line endpoints to facilitate clean corners.

2.6.6 Surface assignment

In surface mode, ModelMaker lets the user select lines to add to or remove from a surface. Surfaces are managed in the surface chooser in a manner consistent with directions and features. The program also provides a mechanism through which the user can manually indicate which images or portions of an image should be used for sampling textures for any given surface.

2.7 Contours

The recovery algorithms described in this paper use image contours as the basis for geometry extraction. A contour is a discontinuity in the image data, as defined by a particular detection mechanism (in this case, the Canny edge detection algorithm). The biggest advantage of using contours is that object silhouettes are usually easy to detect automatically, and that they are relatively independent of lighting conditions. A disadvantage is that contours of an object can originate from a number of physical properties. Following is a summary of the discontinuities that can create a contour and their usefulness in modeling the object(s) on which they appear:

- *Extremal contours* (also called *occluding contours* or *extreme boundaries*), where the viewing direction is tangential to the surface, provide an indication of the surface's extent within a certain view volume. At points along these contours, we know not only the position of the surface, but the surface normal as well.
- *Surface intersections* provide constraints on surface orientation. One example is the curved shared-boundary constraint[9]. It states that for every point along a curve C that

forms the boundary between two curved surfaces A and B, the tangent to C is orthogonal to the normal of A and orthogonal to the normal of B at that point.

- *Surface markings* can be useful with some apriori information about them. For example, stripes on a waving flag indicate the lines of maximum curvature for the surface.
- *Shading* can be used with shape-from-shading algorithms, given information about the light source and surface albedo.
- *Noise* is just not useful.

The algorithms presented in this paper use only extremal contours for geometry recovery. The primary reason is that it is difficult to classify any contours falling into the remaining categories without additional information about the shape or lighting. My goal in this project is to model objects encountered in the real world. In this regard, it is not reasonable to require the object to be evenly colored, or to be lit in a particular manner. Modeling objects of this type may be easier by using traditional CAD methods.

2.8 Generalized cylinders

Generalized Cylinders (GC) were first proposed by Binford[6] as a class of parametric shapes that can be used to represent a wide variety of objects in the real world. A GC is defined as the solid generated by a planar cross-section curve as it is moved and deformed along an axis. Various subclasses of GCs have been studied for purposes of modeling, since the GC class itself encompasses such a large domain. These subclasses are defined by placing constraints on one or more of the following properties:

- the axis

- the cross-section curve
- the deformation of the curve

One such subclass that has received much attention in the research community is the *Straight Homogeneous Generalized Cylinder* (SHGC). A SHGC is a GC in which:

- the axis is straight;
- the cross-section curve is a simple, smooth C^2 curve orthogonal to the axis;
- the cross-sections are scaled only
- the scaling factor is a C^2 function of position on the axis

A *surface of revolution* (SOR, also known as *solid of revolution*) is an SHGC with the additional restriction that the cross-section must be circular. SORs, although more restrictive than GCs or SHGCs, can nonetheless be used as a satisfactory representation for many man-made objects. In an average scene, the number of objects that can be modeled as SHGCs, but not as SORs, is usually quite small.

2.9 Previous work

Nevatia and Zerroug discuss some methods for extracting the shape of a curved object from a single image. Since the problem is underconstrained, these methods use various heuristics that attempt to mimic the human visual system to estimate the unsolved parameters. Two examples are shape compactness[7] and shape smoothness[8]. More effective on a global scale are constraint-based methods that attempt to capture regularity relationships between contours. These assume certain symmetries are present in the object that allow them to prefer one solution over another. As such, they are limited to specific classes of shapes. For the recovery of straight homogeneous generalized cylinders

(SHGCs), Ulupinar and Nevatia[9] use three such constraints: the curved shared-boundary constraint, the inner surface constraint, and the orthogonality constraint. All of these methods, however, assume that perfect curves have already been segmented in the image. In practical applications, this is rarely the case.

Sato and Binford[10] and Zerroug and Nevatia[11] have independently developed methods of extracting 3D shape of SHGCs from imperfect contours. The latter also addresses problems of occlusion and discontinuities in image features. All of the above methods, though, were presented within the framework of an orthogonal projection. Orthogonal projection simplifies the vision process, but it is not representative of images taken with physical cameras.

Gross[12] presents an algorithm for the recovery of 3-D shape of SHGCs from a single orthographic view using both contour and intensity information. As with any method involving intensity analysis, it is limited to objects with relatively homogenous surfaces.

Zheng[13] has explored methods of extracting 3-D shape using sequences of contours of a rotating object. He has successfully demonstrated the modeling of a human head by rotating the person in a swivel chair. In addition, his method allows the detection of regions in the final model that are unexposed to contour, thus providing a starting point for further analyses (e.g. shape-from-shading approaches). On the other hand, his approach places strong constraints on the image capture process. The rotation of the object must be precisely known, or a calibration object must be used for feature tracking when the rotation is not known. Since the number of images required is large, and the nature of the object/camera motion is so limited, one would be unlikely to find any preexisting source data outside of his lab. However, as a modeling method, it demonstrates the advantages of quickly and inexpensively obtaining 3-D data directly from 2-D images.

3. Experiments

This section describes in detail the progress I have made in modeling curved objects. It can best be divided into four subsections – three on modeling solids of revolution, and one on modeling more irregular shapes. All four portions of my project share one implementation. It may be better to view the separate parts as evolving from one to the next instead of as separate experiments altogether.

The user interface portion of the project is embodied in the ModelMaker program, described in Section 2.6. It has been extended in various ways to facilitate the different forms of user input required for each step.

The batch analysis portion of the project consists of the EugeAnalyze program, which is a stand-alone implementation of the algorithms described in this section. It takes as its input an *element file* output by Shawn Becker's sceneAnalyze containing 2-D images and annotation information, estimated camera parameters, and a 3-D scene consisting of textured planar surfaces. Its output is an element file containing, in addition to the elements produced by sceneAnalyze, textured models of cylinders and, in the case of the last experiment, textured non-cylindrical solids.

3.1 Experiment 1: SOR from two image contours

The first phase of this project involved extracting a 3D model of an SOR from user-selected image contours. The user would indicate the locations of the left and right extremal contours of the SOR in each image, along with an axis direction and feature. By forcing the user to identify the contours separately from the cross-section silhouettes, the system made the computer's task much simpler in exchange for demanding more work from the human.

3.1.1 User interface

For this flavor of user assistance, ModelMaker was extended to permit the user to identify left and right (image-relative) contours in each image for each SOR. The contour in this case was a series of connected line segments that indicated an extremal boundary of the cylinder. The user would click-and-drag the line segments to place them in each image. A contour had to be marked as being either the "left" or the "right" contour, relative to the image. The user would also need to bind the SOR to a fixed *direction* that represented the axis direction. Finally, a *feature* had to be bound to the SOR to indicate both the surface that held its base and a point at which it contacted the surface. The feature was required to lie on the end of one of the contours that contacted the base surface. The analysis would heuristically determine which contour to associate with it by examining its relative distance to the contour endpoints.

3.1.2 Batch analysis

The EugeAnalyze program started the geometry extraction process with the feature that was bound to the SOR. This feature, hereon referred to as the *base feature*, had already been fixed by sceneAnalyze. From this 3-D feature and two contours in the same image, the center of the base of the SOR could be found.

Since the feature was not explicitly assigned to a particular contour, the program would determine this by projecting the contour endpoints to the base surface and choosing the one whose projection fell closest to the base feature. The analogous endpoint on the other contour would then be projected onto the base surface as well. The center of the SOR was the midpoint of the line segment joining the two fixed contour endpoints.

With the center of the SOR base fixed, and the axis direction already specified, fixing the remaining contour points was a matter of projecting them to the plane formed by the SOR center and the axis that faced their respective cameras. The 3-D geometry of the cylinder was formed by converting each 3-D contour point to a radius from the axis and a height off the base surface. This became the cross-section curve of the resulting model.

3.2 Experiment 2: SOR from image contours

The next portion of my thesis involved recovering the shape of an SOR from only image edge pixels and a base surface. Requiring the user to manually mark image contours was a step that I wanted to eliminate from the modeling procedure. In lieu of user interaction, an automatic edge detection process was used to create contour points, and the user was only required to assign a surface to the SOR before batch analysis could begin. Given a set of images containing SOR points and a fixed surface, EugeAnalyze would then generate a textured 3D model of the SOR.

3.2.1 User interface

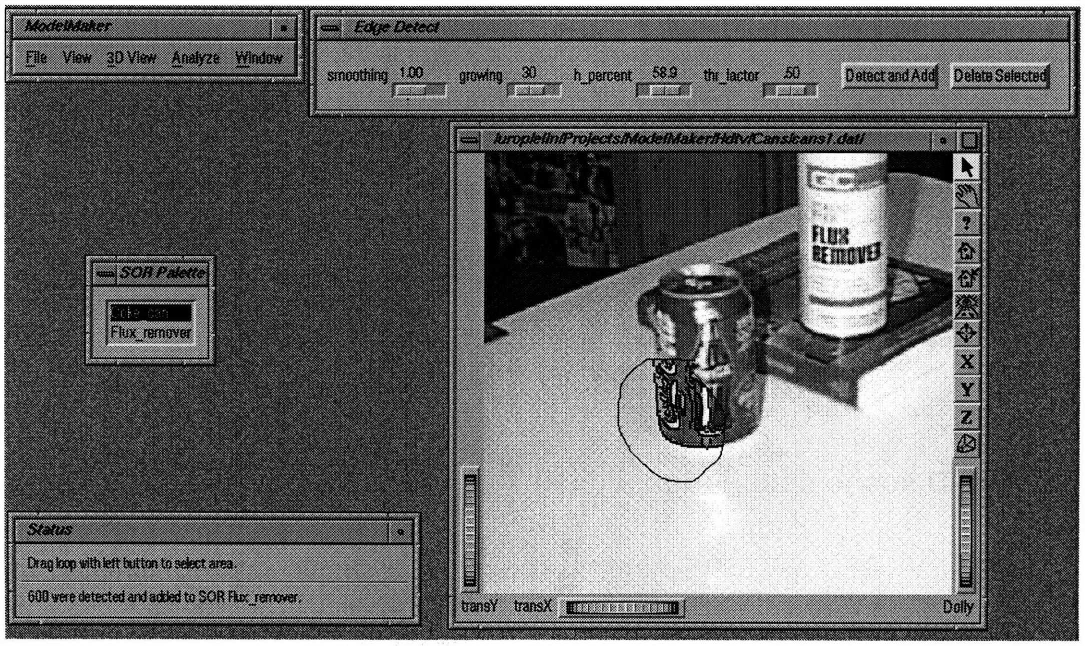
3.2.1.1 2-D scene description

ModelMaker required the user to perform two simple steps to annotate an SOR for 3D geometry extraction:

1. Highlight the region in which the projected SOR image appears and run the edge detector.
2. Assign the SOR to surface.

3.2.1.1.1 Detect edges

ModelMaker uses a Canny edge detector to create image points representing SOR contours. For each SOR, the user selects an arbitrarily-shaped region on the image that contains the projected image of the shape by clicking and dragging the mouse. He then interactively adjusts parameters for the edge detector in a floating palette containing a slider for each parameter (Figure 3). ModelMaker invokes the edge detector on the sub-image defined by the bounding box of the selection region. The program then adds all detected points inside the selected region to the model database. Detected points can be added and removed at any time from any defined SOR by selecting an image region and either detecting more points or deleting all points in the region.



smoothing 1.00 growing 30 h_percent 58.9 thr_factor .50

Drag loop with left button to select area.

600 were detected and added to SOR Flux_remover.

Figure 3: ModelMaker edge detector interface

This screen capture illustrates the parts of the ModelMaker edge detection interface. The *Edge Detect* palette provides access to parameters for the Canny edge detector. In the view window, the user has selected a portion of the image for edge detection. The resulting points can be seen inside the selection region.

3.2.1.1.2 Bind surface

Each SOR must have a surface associated with it, which contains a cross-section of the SOR.

3.2.1.2 3-D SOR display

The ModelMaker program was also modified in order to display 3D SORs in fully analyzed models.

3.2.1.2.1 Geometry representation

SOR geometry is represented as a quadrilateral mesh. The element file describes the SOR as a 3-D center point in world coordinates, a 3-D axis direction vector in world coordinates, a fixed 3-D surface, and a list of 2D profile points. The profile points are given in the coordinate system of the plane formed by the center, the axis direction, and the (1, 0) vector in surface space. ModelMaker, at load time, generates the quadrilateral mesh by rotating the 2D profile about the direction axis. The number of revolutions created is a parameter that can be changed by the user.

3.2.1.2.2 Texture mapping

Textures that are generated by EugeAnalyze can be displayed in ModelMaker. ModelMaker uses Open Inventor to map the texture onto the quadrilateral mesh. For the side of the cylinder, the s axis on the texture maps to its circumference, and the t axis maps to its height. For the top and bottom, the texture is square in *s-t* space. An inscribed circle on the texture maps onto the cylinder geometry.

3.2.2 Batch analysis

In this experiment, EugeAnalyze has a more difficult task to perform than in the previous one. It has to separate the extremal contours on the sides of the cylinder from the contours created by the cross-sections.

3.2.2.1 Bind axis direction

After analysis, each SOR has a fixed direction associated with it. This direction represents the world-relative direction vector of the axis of the SOR. EugeAnalyze uses the normal of the base surface as the axis direction for the SOR.

3.2.2.2 Find image outlines

Detected edges can represent extremal contours, surface intersections, texture, lighting changes, or noise. For this form of geometry extraction, only points that fall into the first category are used. Although contours formed by surface intersections can be used to assist in finding the cross-section of the SOR in cases in which the cross-section intersection and surface are visible, they are too difficult to separate from other detected contours formed

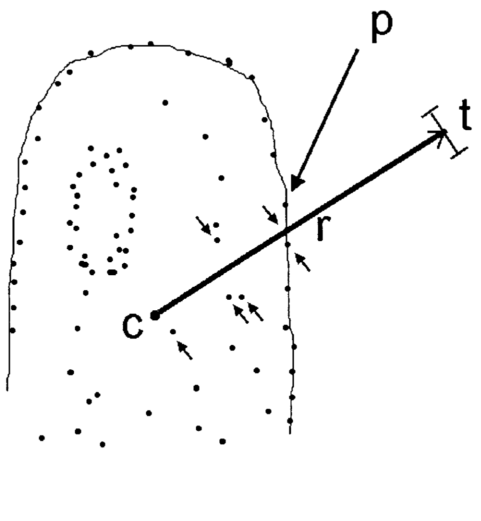


Figure 4: Outline recovery

by shading and texture. Thus, for this project, those contours are not considered. It is important, then, that the edge-detector-generated points located on the interior of the SOR not be assumed to be extremal contour points. We are only interested in the outline of the shape, as viewed from a particular camera. To find the image outline of the detected points, the center *c* of the image points is first found by taking the center of the bounding box enclosing all the points. Then, for each ray *r* from *c* to any SOR image point, all points along the ray except for the one *p* at the greatest distance from *c* are removed, as illustrated in Figure 4. The figure depicts one of these rays. All points (marked with small arrows) within distance *t* of ray *r* will be removed from the set of points on the outline. Point *p* will be kept in the outline set because it is farthest from *c*, the image point set center. The rays are generated for image points in decreasing order by distance from the estimated center. This prevents already-found outline points from being pruned by rays passing through neighboring points.

Define *t* as the maximum distance from *r* at which a point may lie and still be considered "on" *r*. By adjusting *t*, we can control the number of outline points found. Large *t* may prune too many points for accurate geometry extraction. Small *t* may leave interior points, which creates noise in the resulting 3D form. Given a "detectable" edge, an edge detector will find some set *s* of points along that edge. The distance *d* between adjacent points in *s* is dependent on image quality and edge detector parameters. We want to use for *t* a value no smaller than the largest *d* in the image. In order to determine a satisfactory *t*, EugeAnalyze bases its choice of *t* on the average of the distance between each point and its nearest neighbor. This is grounded on the heuristic that a point's nearest neighbor is derived from the same detectable edge.

Note that this approach to recovering the outline from a set of detected points requires that the extents of the object are visible in each image that is used for geometry extraction. When an object is occluded in an image, that image cannot be used for geometry extraction. However, it will still be used in the texture sampling stage. Figure 5 and Figure 6 show photographs of the “Cans” scene, taken with a Kodak digital camera. Figure 7 and Figure 8 show the results of running a Canny edge detector on the portions of the images containing the flux remover canister. Note the large number of interior points generated by the texture on its surface. Figure 9 and Figure 10 illustrate the outlines recovered by EugeAnalyze. These are the points which will be used for 3-D geometry extraction.

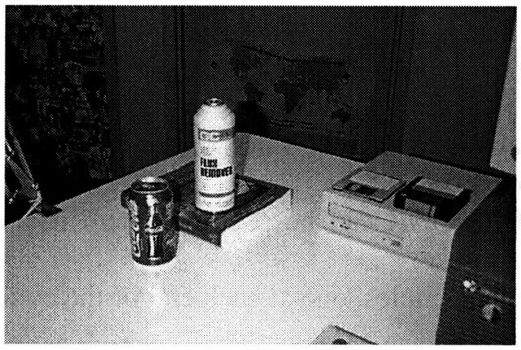


Figure 5: "Cans" original image 1

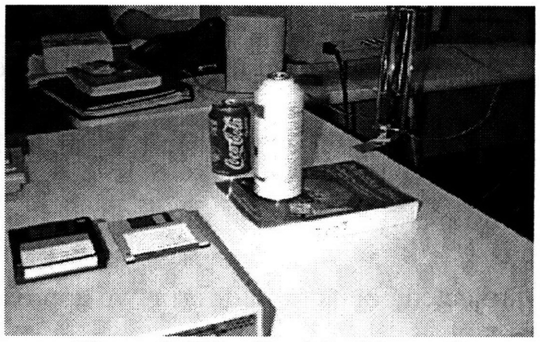


Figure 6: "Cans" original image 2

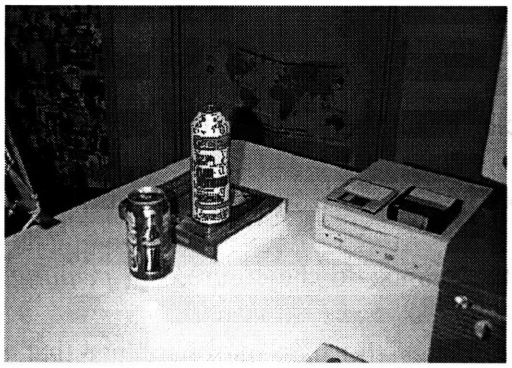


Figure 7: "Cans" detected edge points 1 for flux canister

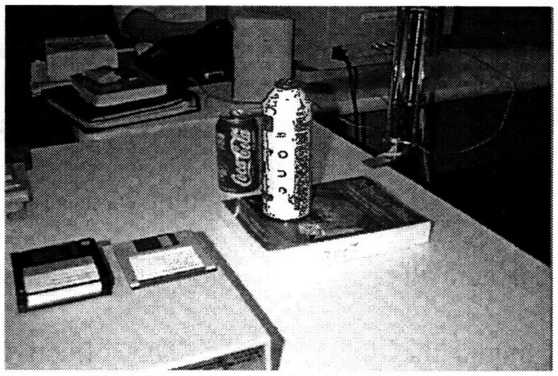


Figure 8: "Cans" detected edge points 2 for flux canister

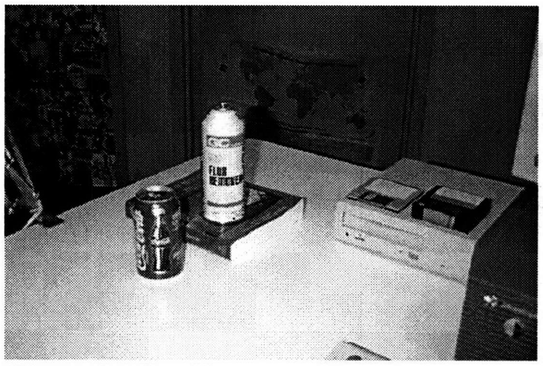


Figure 9: "Cans" recovered outline 1 for flux canister

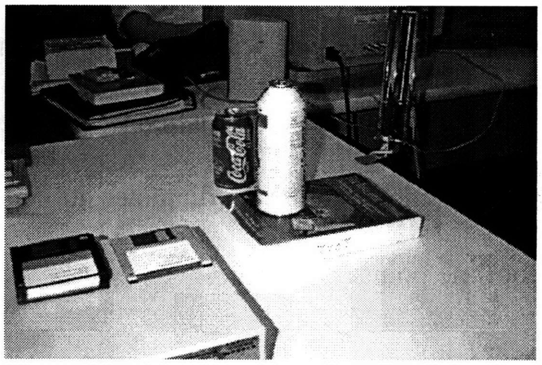


Figure 10: "Cans" recovered outline 2 for flux canister

3.2.2.3 Find base

EugeAnalyze uses the knowledge that the cross-section of the cylinder is circular to find the base, even when only a single image is available. Since the base surface has presumably been fixed, the outline points of the SOR can be projected onto it. These projected points should form two approximate half-circles, one at each end of the cylinder (see Figure 11). Assuming a non-transparent base surface, only one of these circles can represent its base. In the figure, cylinder *A* results from using the correct circle as the base. Cylinder *B* results from the incorrect circle. To ensure that the correct base is found, the program ignores any point whose projection onto the base surface lies farther from the camera than the projection of the center of the points. In the figure, the *pruning line* marks the this boundary. This effectively eliminates the half of the points that were generated by the half of the SOR that is farther from the surface. Then, EugeAnalyze finds the circle that best fits the remaining outline points. To prevent it from fitting a too-large circle to the extremal contours of the SOR, EugeAnalyze limits the circles that can be considered to have a diameter no larger than the bounding box of the pruned projected outline points.

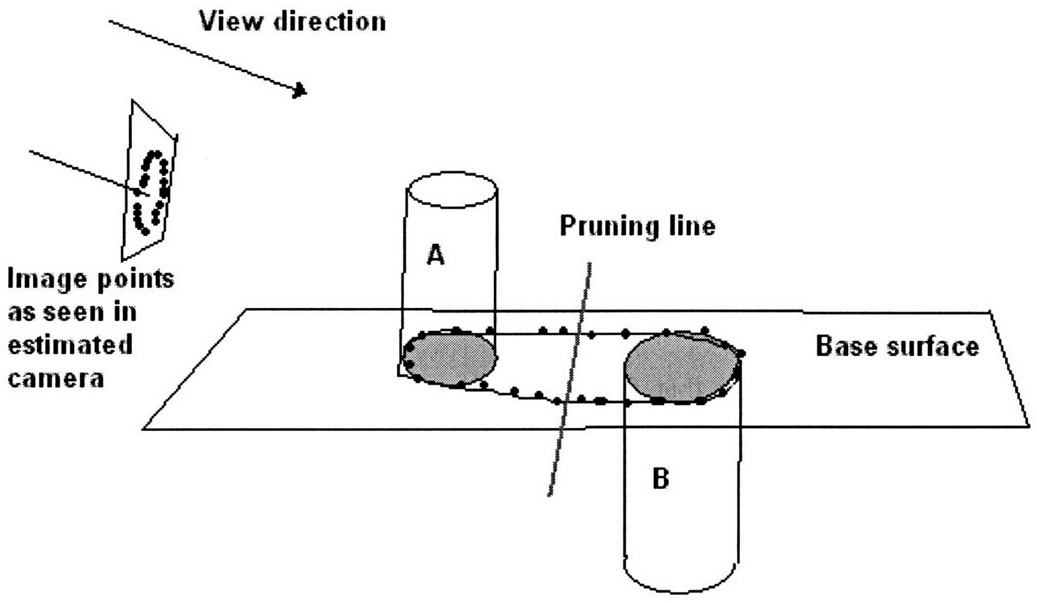


Figure 11: Fitting circles to projected outline points

To further prune candidate points for finding the base, we clip them in we project them onto the base surface. They are then clipped to the extent of the texture that has been mapped onto the surface. This heuristic makes the assumption that the base of the SOR does not overhang the edges of the surface it lies on. Finally, for each image, the bounding box of its projected outline points in surface space is generated. The intersection of all these boxes is then used to further prune the projected hull points. This heuristic can fail in cases where the base of the SOR is occluded in one or more images containing SOR image points.

Now that the size of the outline point set has been significantly reduced, EugeAnalyze fits the best circle to the points as projected onto the base surface. Given a set of projected outline points, the program iterates through all subsets of three. For each combination, a circle is generated by using the intersection of the perpendicular bisectors of any two line segments connecting the three lines as its center. To determine how well the circle "fits" the points, we use as an error measure the number of points whose distance from the estimated center falls outside of a certain threshold of the estimated radius. This provides us with a good starting point for further refinement.

To speed the process of checking all combinations when the number of points is large, we can mark each point that fits some estimated circle. Once a point has been marked, it will not be used again to generate another circle. This method is based on the observation that any points that fit an already estimated circle will generate a similar estimate.

3.2.2.4 Find other cross section

The next step in extracting the 3D geometry of the cylinder is to find the elevation of the other cross section from the base surface. This is done by projecting image points to candidate planes that may contain the other cross section and fitting circles to them. The plane containing the matched circle whose center best matches the estimated cylinder axis is chosen as the plane for the second cross section.

3.2.2.4.1 Prune points on other half of SOR

With a first approximation of the base already determined, the image points can be pruned before entering the cross-section finding stage. Each point is projected onto the axis of the SOR as determined by the base center and axis direction. The range of projected positions along the axis is found for all image points, and the midpoint is taken. Any points projecting on the side of the midpoint containing the projected center of the

SOR are removed from the cross-section candidate set. This reduces the size of the search space and prevents false matches from the image projections of the SOR base.

3.2.2.4.2 Project to offset planes and fit circle

After pruning the SOR points, we project them to candidate planes and fit circles to them. Each candidate plane is parallel to the base surface. Each plane is generated from an image point's projection to the SOR axis. For each plane, we project the set of image points that project to the axis on the opposite side of the plane from the SOR base. We then find the circle that best fits the projected points and note the distance of the circle's center from the axis. The plane containing the circle with the closest center is chosen as the other cross-section plane.

3.2.2.5 Refine center

Once the center and cross-section have been found, the remaining contour points are used to re-estimate a new center. The contour points are the image points representing the extremal contours of the cylinder side (i.e. the visible edges of the SOR that are not formed by either cross-section). To find the new center, the contour points are projected to the base surface and their bounding box is found. The center of the bounding box is used as the new center of the SOR. From this new center and the axis direction, a new axis is computed.

The 3D SOR geometry is stored as a 2D profile to be rotated about an axis. The point coordinates in the profile are defined as radius-height pairs. The radius is computed by projecting the image contour points to the plane p containing the axis and facing the camera. The distance from a contour point's projection onto p and its projection onto the axis is the point's radius. The height is the position of the point's projection onto the axis in axis coordinates.

3.2.2.6 Sample textures

Once the geometry has been fixed, texture information from all images is used to generate textures for each cylinder. Three merged texture maps are created for each SOR: one each for the top, bottom, and side. The texture sampling method is described in the following paragraphs.

3.2.2.6.1 Texture coordinate system

Throughout this paper, textures will be described in *s-t* coordinates, with *s* being the horizontal axis and *t* being the vertical axis in texture space. For the side of a cylinder, the *s* axis spans the circumference of the cross-sections, and the *t* axis runs from the top to the bottom of the SOR.

3.2.2.6.2 Texture resolution determination

Since texture data is obtained from the original images, the amount of useful data is limited by the number of cylinder pixels that appear in all images. It is important that the texture resolution be fine enough to capture as much of this image information as possible without wasting resources with interpolated pixels.

The *s* axis of the texture runs around the circumference of the SOR. We find the radius of the cylinder as seen in the film plane and compute the *s* length from that. To find the radius, in each image we find the maximum distance on the film plane from a contour point to the SOR axis. The maximum of these distances is the radius r of the SOR, and the s length is $2\pi r$.

To determine the length of the texture in the t direction, we compute for each camera the distance between the cross-section centers as projected onto the film plane. The maximum of these distances is used as an estimate of the s length of the texture. It is multiplied by a constant factor to further ensure the capture of all image data.

The textures for the top and bottom of the SOR are squares of identical size. The length of the edges of the squares are determined from the estimated maximum diameter of the cylinder as viewed from the original cameras. For this diameter, we simply double the radius r found in the previous section for the s dimension of the side texture.

3.2.2.6.3 Multi-view texture sampling (partial implementation)

The geometry of an object modeled using one of the algorithms in this thesis is obtained by using contour information alone. In order to create a model that faithfully resembles the object in the original images, we need to generate the texture to be applied to the recovered geometry. Since the 2-D scene description contains only a few images of the original object, the computer only has information about the object's appearance as seen from the original cameras. We wish to use this information to generate the "best" texturing for the shape. The recovered geometry with the "best" texturing will, when rendered from the estimated cameras, match the original images as closely as possible.

The texture is generated by mapping x-y image coordinates into s-t texture space and sampling image pixels directly into the texture map of the object. This mapping is achieved by rendering from the original cameras the recovered geometry of the object with a special calibration texture mapped onto its surface. Each pixel *p* of the calibration texture contains:

- the *s* coordinate of *p* in texture space,
- the *t* coordinate of *p* in texture space, and
- an identifier that is unique to object or object part.

Once the rendered *calibration image* has been produced, we can read *s-t* texture coordinates directly from *x-y* image coordinates. Likewise, the identifier tells us which texture to project the image pixel toward. Figure 12 illustrates the pixel-to-pixel relationship between the original image and the calibration image. A pixel on the original image tells us what color to map onto the texture. The pixel on the calibration image at the same location tells us the ID of the texture that is visible at that location as well as the *s-t* coordinates of the exact texture element.

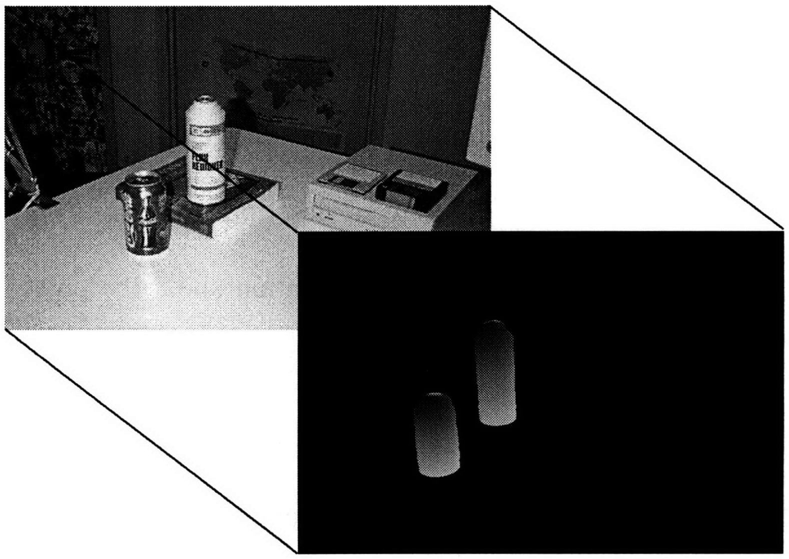


Figure 12: "Cans" original image with calibration image

In this partial implementation, EugeAnalyze maps each image pixel onto one texture pixel. This allowed me to see how well the geometry matched the original images without unnecessarily slowing execution of the program. In the next experiment, I completed the texture sampling system to deal with one-to-many mapping of image pixels. For this particular test, I instead had the program perform what I thought would be a faster, but less correct texture interpolation process.

3.2.2.6.4 Sparse-texture interpolation

Due to the fact that the actual transformation from image pixels to texture pixels is usually not a one-to-one mapping, the resulting sampled textures will contain some pixels without color values assigned from any image. To rectify this, the color values of the empty pixels in the sparsely filled texture are interpolated from filled pixels. The algorithm is simple:

1. Define a neighbor of pixel at (s, t) as any pixel with coordinates ([s-1, s+1], [t-1, t+1]). For every empty pixel in a sampled texture with at least one filled neighbor, sets its value to the average of the filled values.
2. Repeat from step 1 until no new pixels are filled.

The only caveat is that the *s* axis needs to "wrap around" from one end of the texture to the other to eliminate any synthetic seam from appearing in the texture. This is necessary because the rectangular side texture of an SOR must be wrapped onto a cylindrical surface. To accomplish wraparound, I define (0, *n*) and (s_{max}-1, *n*) to be neighbors, where s_{max} is the horizontal resolution of the texture and n is any integer in the range [0, t_{max}), where t_{max} is the vertical resolution.

3.2.2.6.5 Advantages

This method of texture sampling offers a number of benefits[ii]:

- It is independent of the intricacies of the rendering pipeline. Any unusual treatment of the projection onto the film plane of 3D geometry will affect both the calibration images and the rendered images of the extracted model.
- It is independent of the texture-mapping implementation (i.e. no matter how the texture is applied to the geometry, it will perform correctly).
- It handles occlusion without any extra computation or user intervention.
- It ensures that the final model, when rendered from the original cameras, will most closely match the original images.

[ii] This assumes that the same rendering equipment will be used for texture sampling and viewing the extracted model.

3.3 Experiment 3: SOR from image contours

The most complete portion of my thesis involved a revised approach to recovering the shape of an SOR from image edge pixels and a base surface. The algorithm has been completely implemented in *EugeAnalyze*. The program takes as its input the output of the vision-assisted modeler described in Section 2.3. This input contains a partially recovered 3-D scene (planar surfaces and textures only) as well as 2-D annotation information, including cylinder definitions and detected cylinder points, generated by the user through ModelMaker.

3.3.1 User interface

The user interface to this experiment is identical to that of Experiment 2. The changes I made were in the analysis portion.

3.3.2 Batch analysis

The EugeAnalyze program takes as its input an element file containing a recovered model with properly annotated SOR images. It will take the image points of defined SORs and generate 3D geometry and textures for them whenever possible.

3.3.2.1 Fix axis

The third experiment is similar to the second in many respects. It begins to diverge starting with the method used to fix a 3-D position for the axis of symmetry. An initial estimate of the axis is found first using triangulation where possible, then it is refined based on the image contours. Depending on the number of images in which the

SOR appears, EugeAnalyze takes one of two different approaches to finding an initial estimate.

3.3.2.1.1 Estimate center from multiple images

In the case where multiple images are available with detected SOR contour points, triangulation is used to fix an approximation of the SOR's center. In each image, the center of the points in the SOR is found and assumed to be a 2-D view of the SOR center. The average triangulated 3-D position from all pairs of images is used as the initial estimate of the center.

3.3.2.1.2 Estimate center from single image

When only one view of a scene is available, triangulation cannot be used to fix the center of the cylinder. Instead, the circle-fitting method of Section 3.2.2.3 is used to find the center of the SOR base. Then with the center of the base fixed, EugeAnalyze finds the center of the SOR shape by projecting the center of the points in the image onto the plane that contains the axis and faces the camera. Figure 13 displays the 3-D planar surfaces for the "Cans" scene as generated by vision-assisted modeling. This scene is part of the input to EugeAnalyze. The recovered camera positions are also shown.

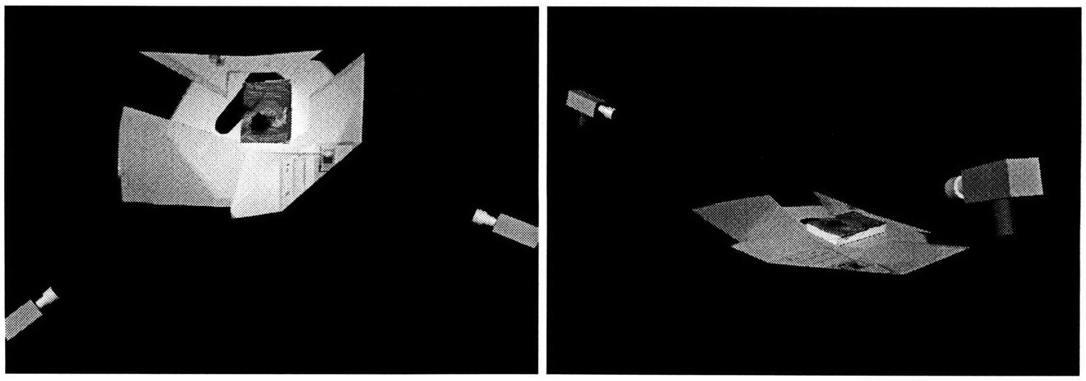


Figure 13: Planar surfaces in "Cans" scene with estimated cameras

3.3.2.1.3 Refine axis position

Once an estimate of the center of the SOR has been fixed, EugeAnalyze refines the axis position based on the image contour points. The image points are projected onto the plane passing through the center that faces the camera. Using the lines that connect the camera COP with the projected points, EugeAnalyze generates a 3-D cone for each camera, representing the possible enclosing volume of the object as seen from that camera. Rays are fired along the positive and negative axis direction from the SOR center until they hit a part of the rendered 3-D cones. These points are used as the upper and lower bounds of the SOR along the axis. At each axis position, then, rays are fired in a disc oriented orthogonal to the axis. The shortest diameter is used as the diameter for that cross-section. Once diameters have been found along the length of the cylinder, the center of the SOR is set to be the average center of all the diameters. This process is repeated until the change in the center (in the plane parallel to the base surface) is below a pre-set threshold.

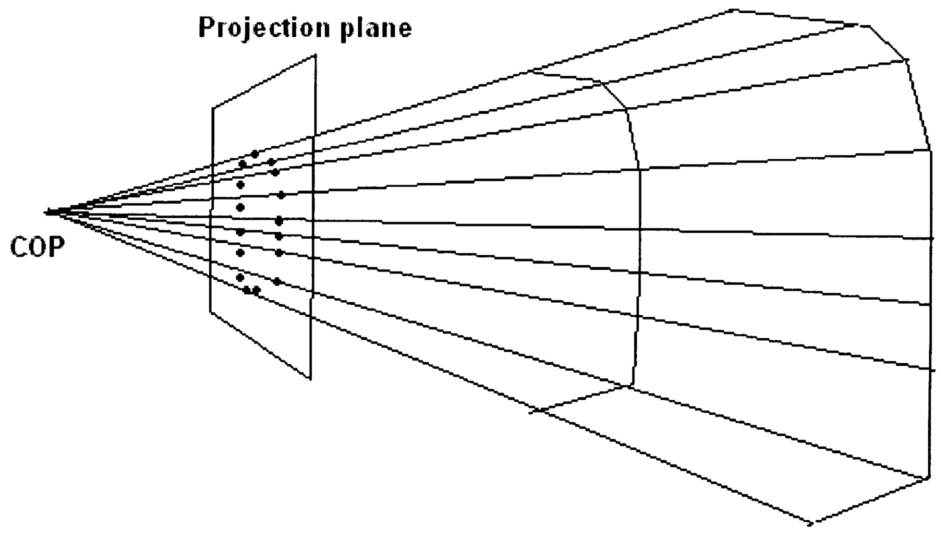


Figure 14: Projected contour points and object volume

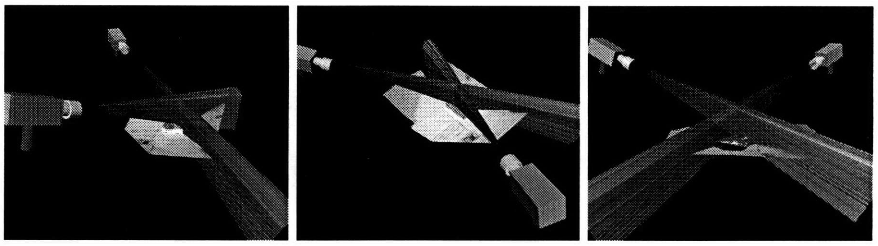


Figure 15: "Cans" -- projected contour points for flux canister

3.3.2.2 Generate mesh geometry

After the axis has been fixed to within the desired tolerance, the geometry of the SOR is "sampled" using the same ray-firing technique in the previous subsection. This results in 3-D points on the surface of the cylinder arranged in a grid, allowing a quadrilateral mesh to be wrapped around the shape.

3.3.2.3 Multi-view texture sampling

In this experiment, I completed the implementation of the texture-sampling method used in Section 3.2.2.6.3. EugeAnalyze still uses special textures to render calibration images to use for image→ texture space mapping. It is the mapping itself that was improved.

Since a single pixel in image space doesn't necessarily map onto a single pixel in texture space, each image pixel is projected as a quadrilateral onto texture space and filled with a single color. To find the *s-t* coordinates of the corners of the pixel, the calibration image is rendered at a larger scale than the original image. Each pixel in the original image maps onto a square of pixels in the calibration image. The *s-t* coordinates of the corners of that square represent the corners of the quadrilateral in texture space.

The textures in Figure 16 were sampled from two images. The overlap between the two original images in each texture can clearly be seen. The differences in color result from a number of factors, including specular lighting, different exposure levels in the images, and errors in camera parameters. Correction for exposure can be done on the images before shape recovery. The error in the camera parameters can be a result of the restrictions of Open Inventor cameras. Since the principal point of a real camera is usually not in the center of the film plane, the estimated cameras used by EugeAnalyze will invariably contain some error, no matter how accurate the camera calibration algorithm.

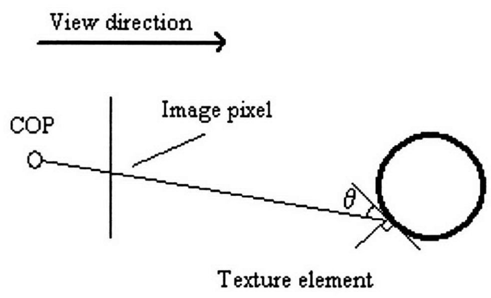


Figure 17: Mapping image pixels to texture elements

To account for small errors in geometry recovery, the texture mapping portion of EugeAnalyze prunes pixels whose projection is larger in area than a threshold size. The threshold is based on the median size of all texture elements derived from the same image for the same object. This prevents inaccurate texture elements from being created when the projection angle θ is small (Figure 17).

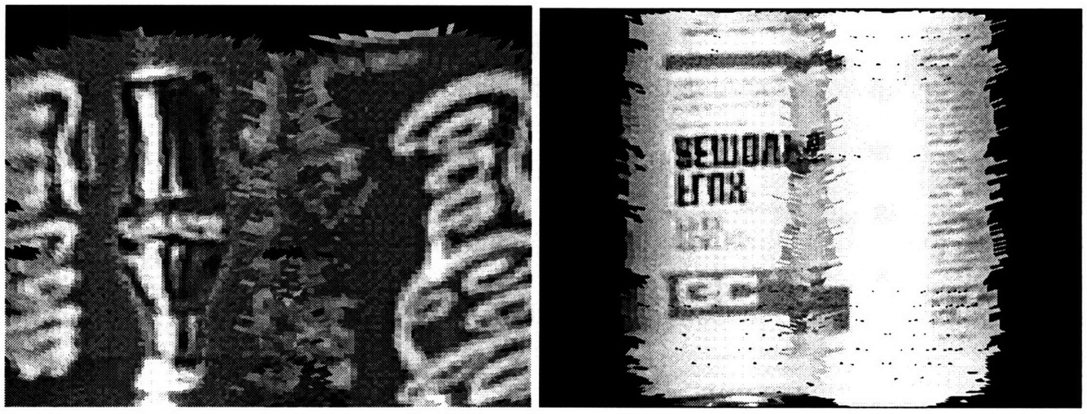


Figure 16: Sampled textures for "Cans" scene: coke can and flux canister

3.4 Experiment 4: Curved objects from multiple edge images

The final phase of my work was to explore the possibility of recovering the shape of arbitrarily shaped 3-D objects from multiple edge images. Since the projected contours in Section 3.3.2.1.3 represented the largest possible volume of the object, I thought it would make a good starting point for modeling non-cylinders. Using this method, of course, meant I had to make several assumptions about the object being modeled and the views used. Because the center of the image points were assumed to be near the object center, the object must be somewhat symmetric. The outline recovery algorithm, too, limits the outline of the object to being star-convex in each view. While not all objects fit these parameters, the class of objects that can be modeled by this method encompasses a broader range than generalized cylinders.

3.4.1 User interface

For this experiment, I used the same user interface as in the previous two sections. However, the object does not need to be assigned to a surface. Only image edge points are needed, and at least two images are required.

3.4.2 Batch analysis

The analysis portion of this experiment is very much a generalization of the technique of projecting image contour points used in Experiment 3. First, a 3-D object center position is estimated using the triangulation method of Section 3.3.2.1.1. Then, contour points are projected from the COP through the film plane to form enclosing cones that constrain the volume of the object. The intersection of the volumes created by cones from multiple images is the volume of the object being modeled. To generate the mesh necessary for viewing, the interior of the volume created by the intersection of the cones is sampled using a similar method to that of Experiment 2 and 3. The difference here is that instead of taking the smallest diameter only for each cross-section, all points are added to the mesh. Since the solid is not assumed to be cylindrical, the axis used for geometry sampling is set to be the longest axis in the 3-D volume.

4. Results

4.1 Solids of revolution

Presented in this section are rendered images of the recovered geometry of some cylinders, using the system developed in Experiment 3.

4.1.1 Cans (one view)

Figure 18 is a single digital image of the "Cans" scene. The original camera was estimated using the vision-assisted modeling process, and the book and table top were modeled as planar surfaces in 3-D. The partially completed scene was then passed to EugeAnalyze for curved shape recovery. Figure 19 shows three views of a wireframe model of the recovered cylinder geometry, generated using the system implemented in Experiment 3.

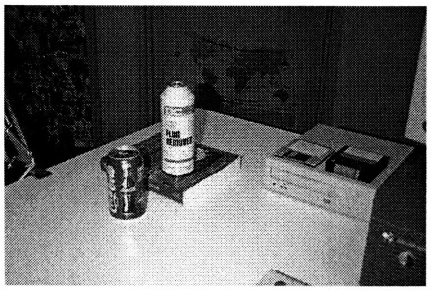


Figure 18: Original image

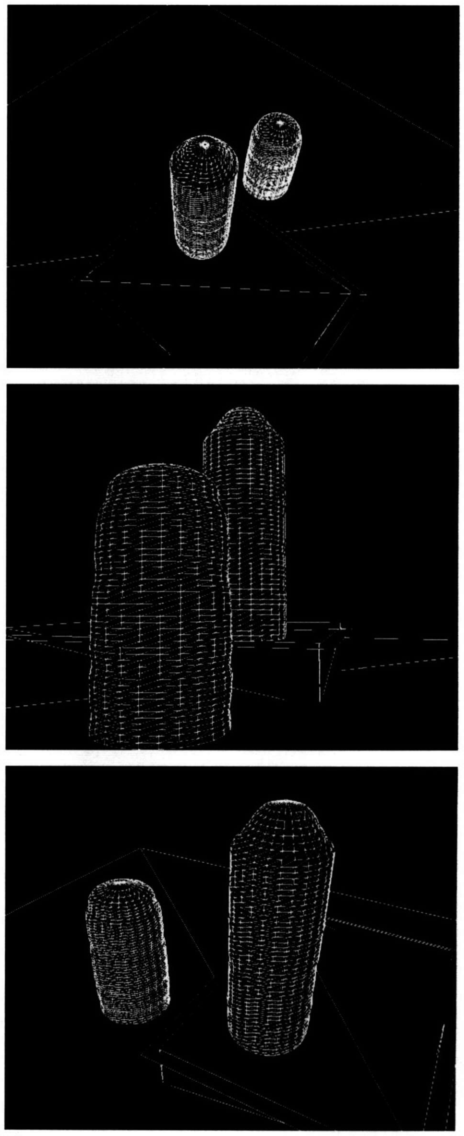


Figure 19: Recovered geometry

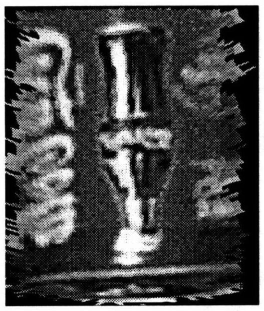


Figure 20: Texture for Coke can

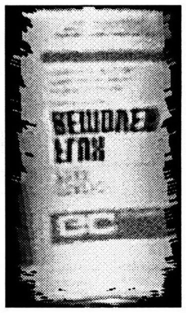


Figure 21: Texture for flux canister

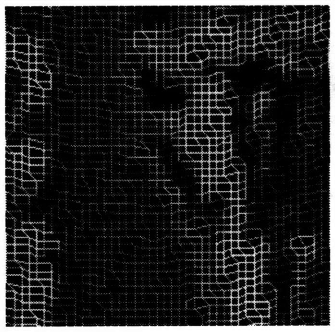


Figure 22: Texture for Coke can pixel boundaries (zoomed)

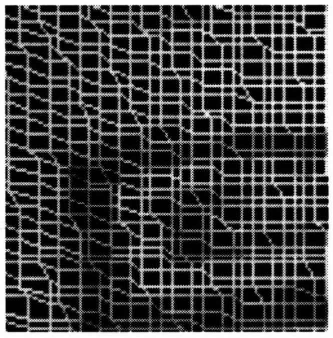


Figure 23: Texture for Flux canister pixel boundaries (zoomed)

The unwarped textures for the Coke can and Flux canister objects are shown in Figure 20 and Figure 21. Closer inspection of each texture reveals the individual projected image pixels, which appear as quadrilaterals in texture space (Figure 22 and Figure 23). Each quadrilateral represents the area of one pixel from the original photograph as projected on the recovered geometry through the estimated camera.

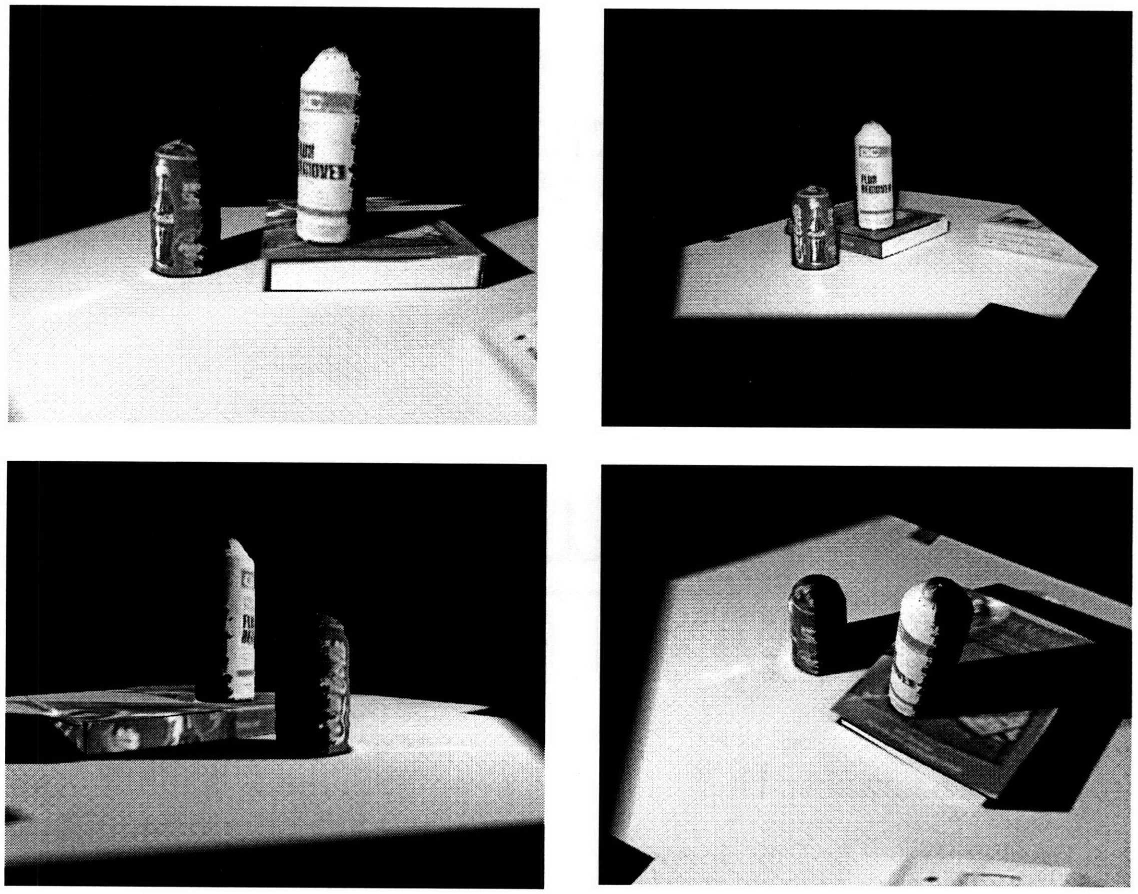


Figure 24: Recovered scene from one image

The illustrations above show rendered images of the recovered 3-D "Cans" scene from one original photograph. The positioning of the cylinders can be verified by the "shadows" left on the planar surfaces of the table and the book face in the vision-assisted modeling process. The "shadows", or dark areas, on the table surface and book cover are not due to lighting. They result from a lack of texture information in the vision-assisted modeling stage of shape recovery. Likewise, the black areas on the two cylinders represent areas of known geometry but unknown texture.

4.1.2 Cans (two views)

This model was extracted from the images shown in Figure 5 and Figure 6. It demonstrates the effect of merging textures from multiple images. The error in camera estimation can best be seen in the upper orange stripe on the flux container. Corresponding more point features between images during vision-assisted modeling helps alleviate this problem.

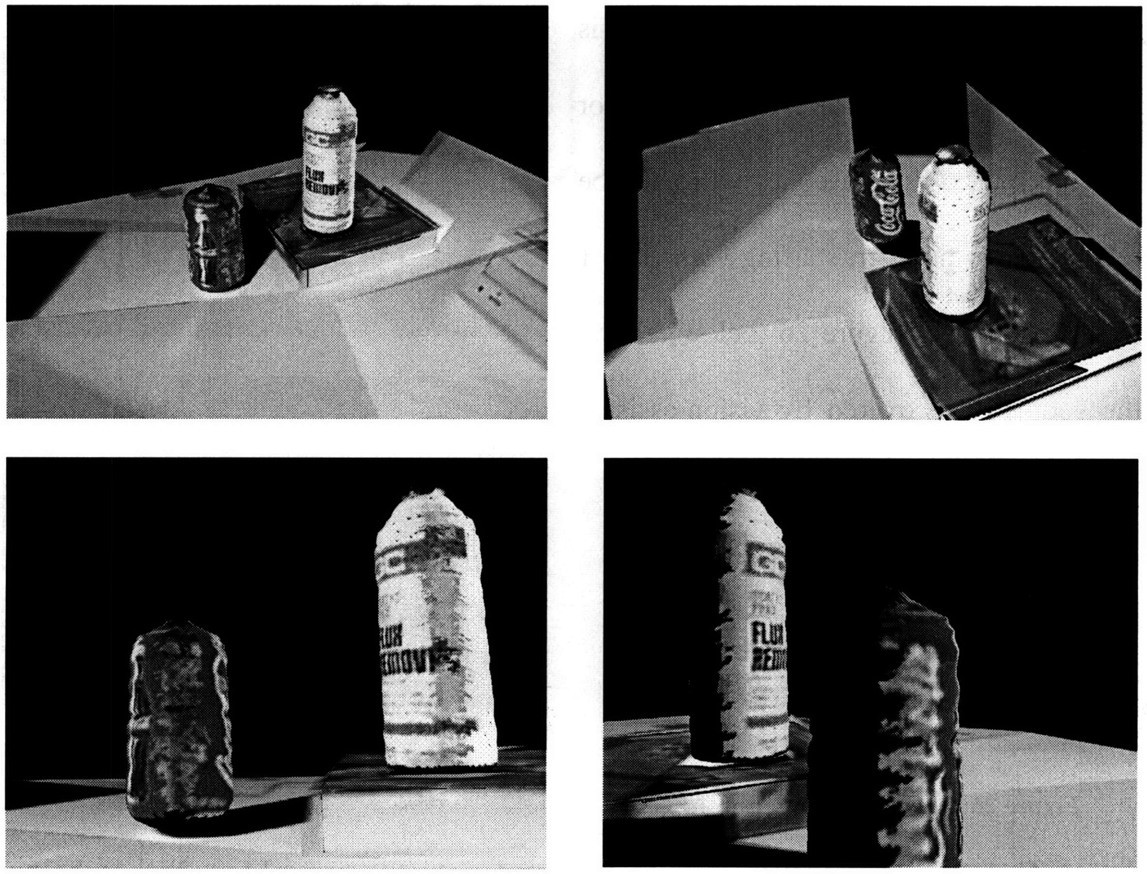


Figure 25: Recovered scene from two images

4.1.3 Kendall (one view)

Figure 27 depicts the original image for the "Kendall" scene. The three images in the figure show the same photograph, with outlines for the three traffic cones to be modeled. The farthest cone cannot be modeled from this image because it is partially obscured by the striped cone. For labelling purposes, the cones will each be referred to by number, starting with cone 1 in the top image. Figure 26 (below) shows the geometry recovered by vision-assisted modeling from the single image.

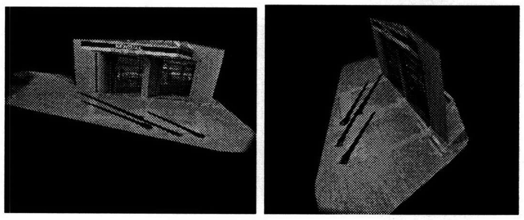


Figure 26: Recovered planar geometry

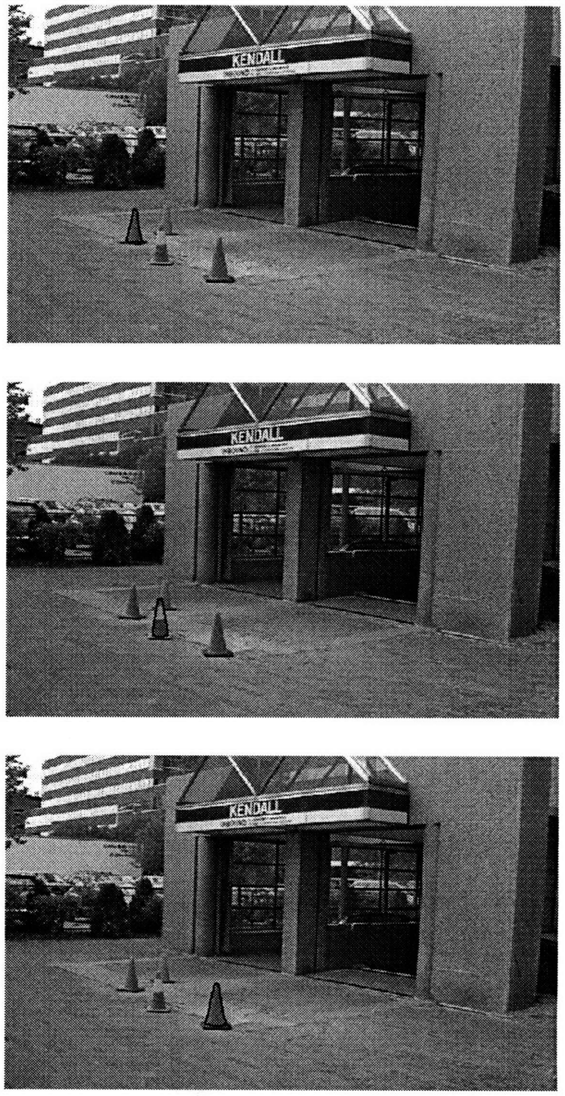


Figure 27: Original image with cone outlines

The outlines in Figure 27 were generated manually using ModelMaker, because the edge detector was not able to extract a satisfactory edge image from the original photograph. Images in which the objects being modeled appear at low resolution all had this problem. The geometry recovery system, however, can deal with outlines at a higher resolution than the source image.

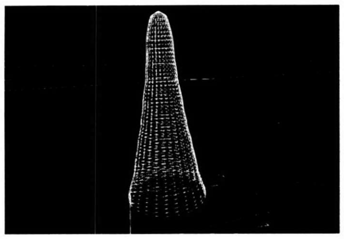


Figure 28: Cone 1 geometry

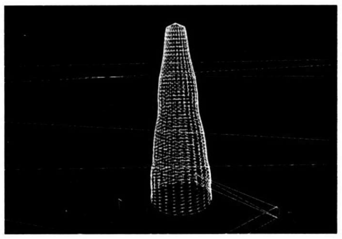


Figure 29: Cone 2 geometry

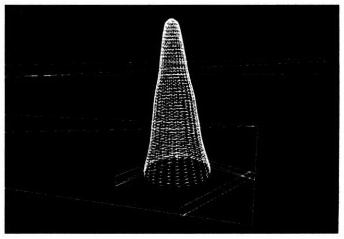


Figure 30: Cone 3 geometry

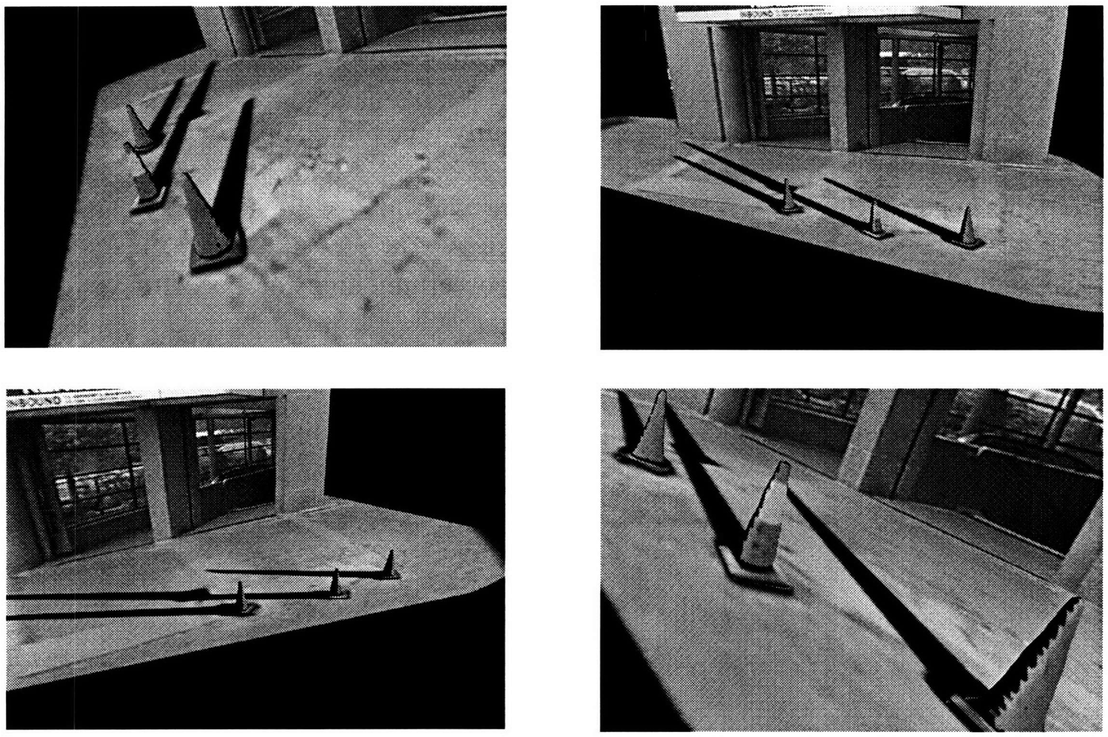


Figure 31: Recovered textured "Kendall" scene

5. Conclusions

This thesis presented some experiments on the recovery of 3-D shape of curved objects from calibrated 2-D images. The first three experiments concentrated on solids of revolution, while the fourth attempted to generalize it to other forms.

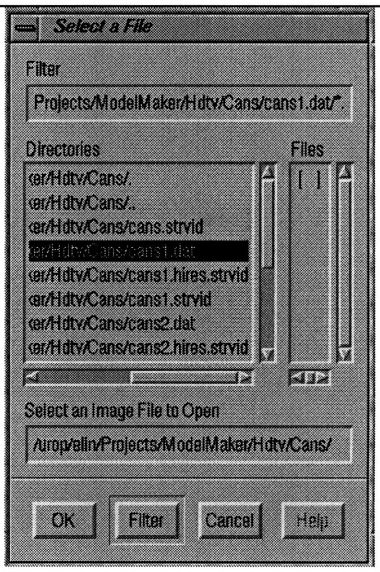
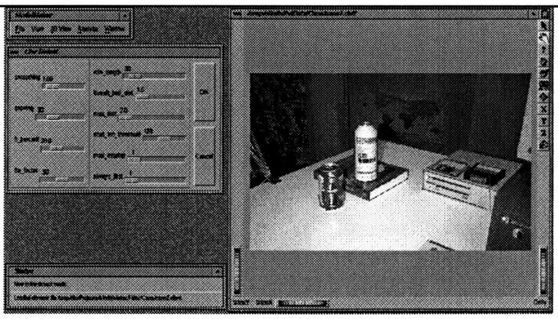
The recovery algorithm in Experiment 1 required too much user input to claim to be a quick and effective method of recovering shape. However, it was a good starting point for further work. Experiment 2 successfully recovered some cylinders, but caused problems when the cross-section or base were viewed from a direction parallel to the cross-section. Experiment 3 improved on this by eliminating the need for visible cross-sectional outlines except in the case of one image. Some recovered scenes were presented, both using multiple views and a single views. Additionally, a texture-sampling algorithm for preserving appearance from the original cameras was described and demonstrated, with visible success.

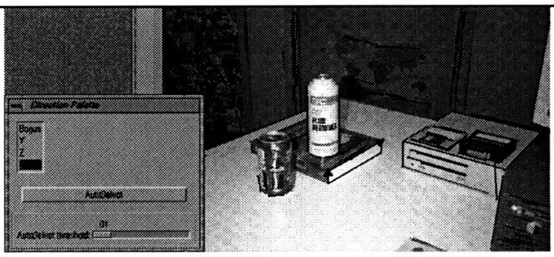
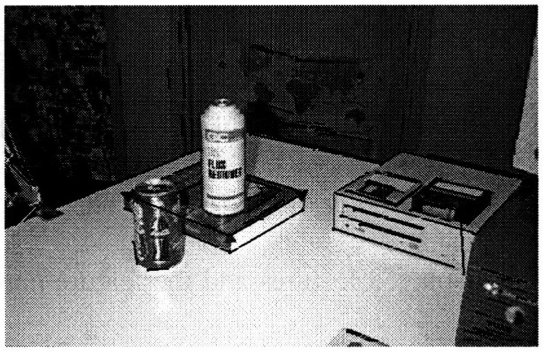
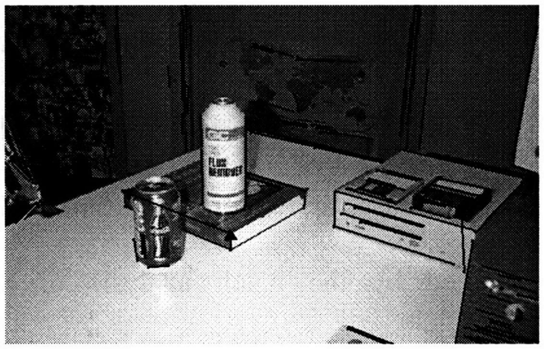
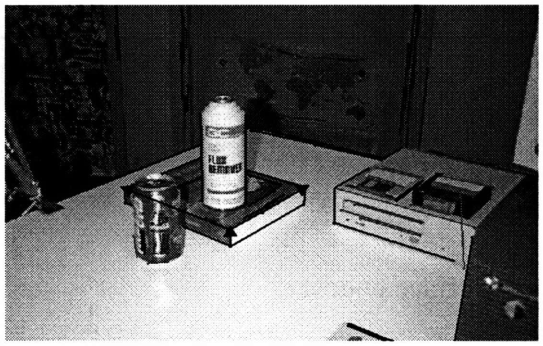
The final experiment presented a more general scheme for modeling 3-D shapes from outlines alone. Its results were less than spectacular, but show some promise. A suggestion for future work is to allow 2-D feature correspondences between images to be used to augment the object volumes when contour alone does not constrain the shape satisfactorily. Since planar surfaces are also used to constrain the shape, 3-D surfaces (i.e. bounded plane segments) could be placed within the object volume by other means, such as vision-assisted modeling, to compress the volume further. If we limit ourselves to smooth objects, we can use surface curvature or other means to interpolate values between contour boundaries and known interior features.

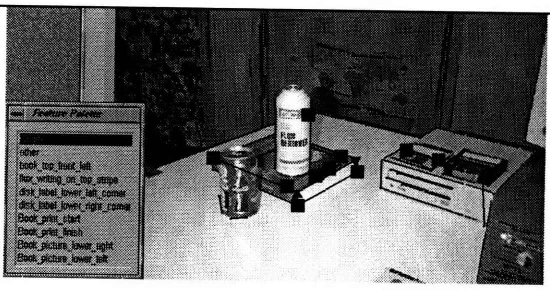
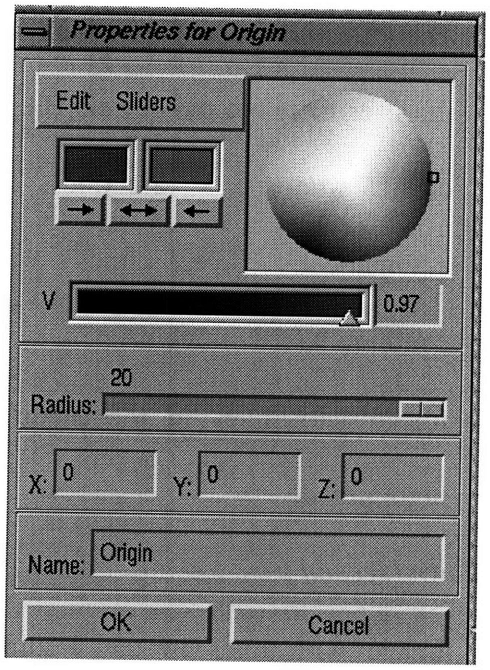
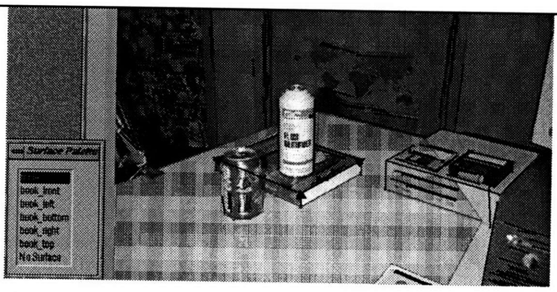
As a whole, the algorithms presented in this thesis are an effective way of modeling specific shape classes directly from 2-D images. With a modicum of user input, one can quickly recover the 3-D shape of solids of revolution and create models for use in computer graphics.

Appendix A: Example scene

This section presents a step-by-step walkthrough of the model recovery process, from digitized images to recovered textured model. The screenshots are of the ModelMaker program running on a Silicon Graphics workstation.

Step 1: Open image(s) *Shown:* • The File Open dialog, used to select an image file to load.	 Filter Projects/ModelMaker/Hdtv/Cans/cans1.dat/*. Directories Files \<er/Hdtv/Cans/. \<er/Hdtv/Cans/.. \<er/Hdtv/Cans/cans.strvid er/Hdtv/Cans/cans1.dat \<er/Hdtv/Cans/cans1.hires.strvid \<er/Hdtv/Cans/cans1.strvid \<er/Hdtv/Cans/cans2.dat \<er/Hdtv/Cans/cans2.hires.strvid [ ] Select an Image File to Open /urop/elin/Projects/ModelMaker/Hdtv/Cans/
Step 2: Detect lines *Shown*: • The menu bar, which is the top level command interface. • The line detect dialog, which provides access to edge detection and line detection parameters. • The status bar, which notifies the user of actions performed, and commands • Image 1 from the "Cans" scene, with	

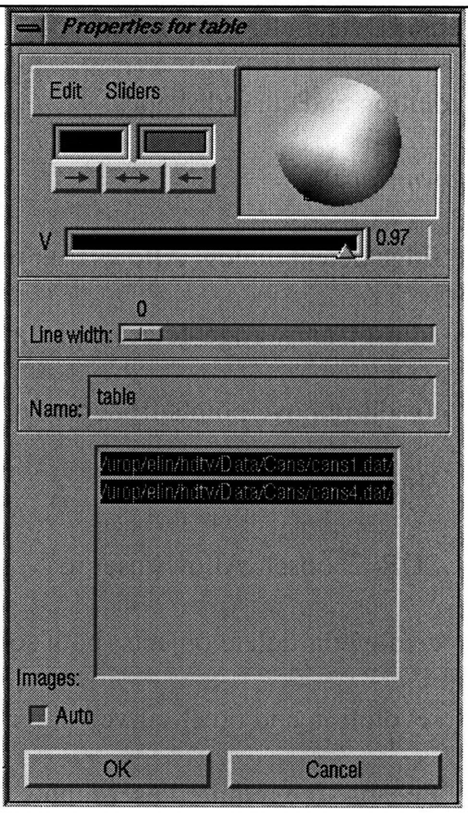
detected lines. • available in the current context.	
Step 3: Group lines into directions. *Shown (top):* • The directions chooser, which allows the user to select the current direction, create/edit/delete directions, and automatically select lines with a common vanishing point. • Image with lines in the *X* direction highlighted. *Shown (second from top thru bottom):* • Image with lines in the *Y* direction highlighted. • Image with lines in the *Z* direction highlighted. • Image with lines in the *Bogus* direction highlighted. Lines in the bogus direction are not used for camera calibration in the vision-assisted modeling process.	 Direction Palette Bogus Y Z AutoSelect AutoSelect threshold:   

Step 4: Place features in images *Shown (top):* • Feature chooser. • Image with features marked as large squares. *Shown (bottom):* • Feature properties dialog. The color editor allows the user to change the display color of the feature. The size of displayed features and the feature name can be changed in this dialog. The X, Y, Z fields can be filled with numerical values, or with question marks. In the latter case, the feature position will be fixed by the batch analysis of the vision-assisted modeling program.	 Feature Palette other book_top_front_left fax_writing_on_top_stripe disk_label_lower_left_corner disk_label_lower_right_corner Book_print_start Book_print_finish Book_picture_lower_right Book_picture_lower_left  Properties for Origin V 0.97 Radius: 20 X: 0 Y: 0 Z: 0 Name: Origin
Step 5: Classify lines and features into coplanar sets (surfaces). *Shown (top):* • Surface chooser. • Image with surface *table* highlighted. The polygon was placed by the user and	 Surface Palette book_front book_left book_bottom book_right book_top No Surface

represents the area from which to sample the texture for the surface. The current vision-assisted modeling implementation does not automatically detect areas of occlusion on a surface.

Shown (bottom):

- Surface properties dialog. The top color editor sets the display color of the surface. Line width and surface name can be changed with the middle fields. The images field list all images in the model. Textures for this surface will be sampled from selected images only.

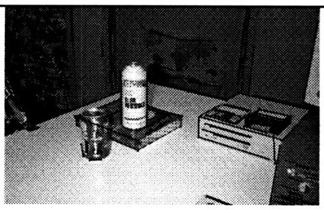
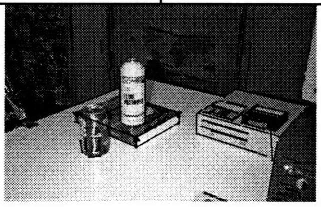
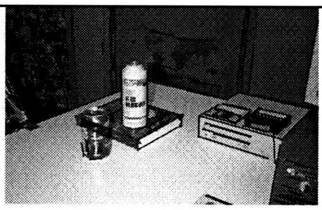
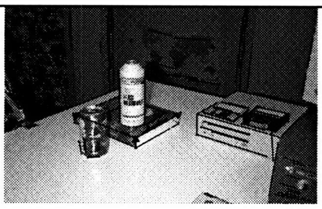
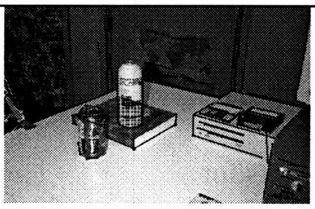
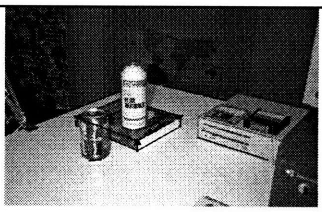


V 0.97

Line width: 0

Name: table

Images:
/urop/elin/hdtv/Data/Cans/cans1.dat
/urop/elin/hdtv/Data/Cans/cans4.dat

 Book bottom surface	 Book top surface	 Book left surface
 Book right surface	 Book front surface	 Unclassified lines

Step 6: Detect edges for solids of revolution and curved shapes.

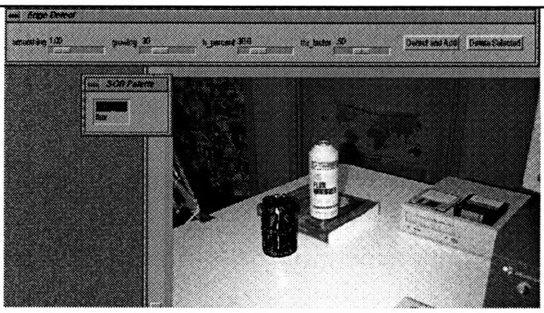
Shown (top):

- Edge detect dialog. Allows user to adjust Canny edge detector parameters to add/remove points from the current object.
- SOR Chooser. Allows user to create/edit/delete objects (both solids of revolution and other curved objects). Because the required user inputs for Experiment 4 have not been solidified, the interface remains incomplete. Therefore, for now, both solids of revolution and arbitrarily shaped curved objects share one interface.
- Image with detected edge points for Coke can shown.

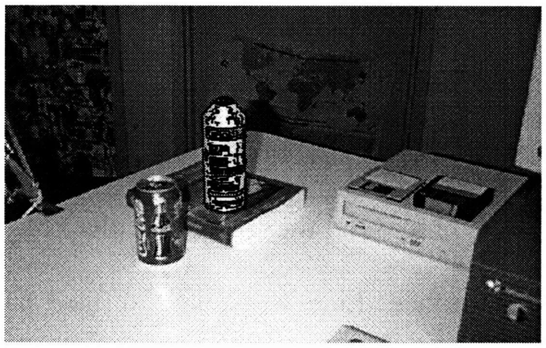
Shown (middle):

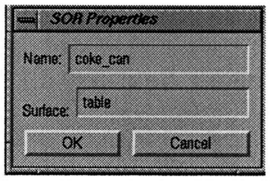
- Image with detected edge points for Flux canister shown.

Shown (bottom):



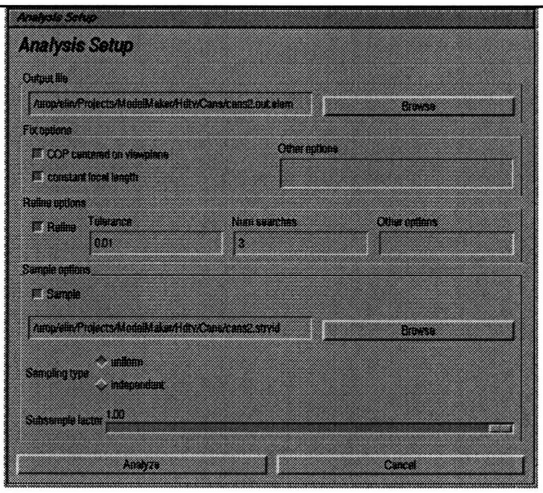
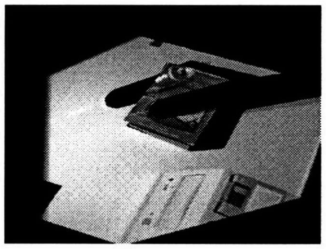
smoothing 1.00 growing 30 h_percent 30.0 thr_factor .50

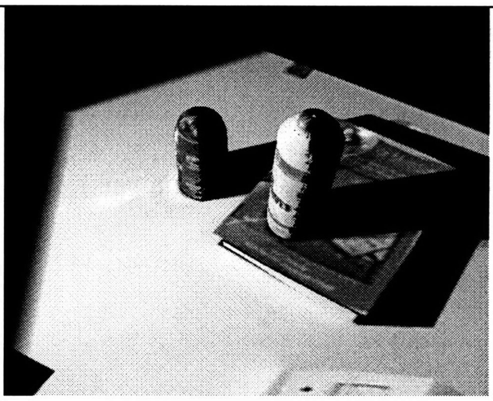




Name: coke_can

Surface: table

• SOR Properties dialog. Allows user to change name of object and assign a surface (needed for analysis). If a surface is supplied, the object will be analyzed as a solid of revolution, otherwise, the object will be analyzed as an arbitrarily shaped curved object (see Experiment 4). This interface needs to be improved.	
Step 7: Vision-assisted modeling batch analysis. *Shown (top):* • Analysis setup dialog. Provides access to batch analysis parameters for vision-assisted modeling. *Shown (bottom):* • Recovered scene with planar surfaces only (from vision-assisted modeling).	 Analysis Setup Output file: /urop/elin/Projects/ModelMaker/Hdtv/Cans/cans2.out.elem Fix options: COP centered on viewplane; constant focal length Refine options: Refine; Tolerance 0.01; Num searches 3 Sample options: Sample; /urop/elin/Projects/ModelMaker/Hdtv/Cans/cans2.stryid Sampling type: uniform / independant Subsample factor 1.00 

Step 8: Recovery of curved objects. Because the requirements for curved object analysis changed over time during this project, ModelMaker does not yet provide access to EugeAnalyze. *Shown:* • Recovered scene, with textured cylinders.	

Bibliography

[1]Shawn Becker and V. Michael Bove, Jr. Semiautomatic 3-D model extraction from uncalibrated 2-D camera views. *SPIE Image Synthesis* (February 1995).

[2] Josie Wernecke, *The Inventor Mentor.* Addison-Wesley Publishing Company, Reading, MA, 1994.

[3] J. Canny, A Computational Approach to Edge Detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 6 Nov. (1986), pp. 679-698.

[4] Stefan Agamanolis, "Line-finder man pages", *MIT Media Lab TVOT man pages*, December, 1995.

[5] Http://garden-docs.www.media.mit.edu/garden_docs/gnu/readline/rlman_toc.html

[6] T. O. Binford, "Visual perception by computer," *Proc. IEEE Conf. Systems and Control*, Miami, Dec. 1971.

[7]M. Brady and A. Yuille. An extremum principle for shape from contour. *IEEE Transactions on Pattern Analysis and. Machine Intelligence..* **PAMI-6**, 288-301 (1984).

[8]H. G. Barrow and J. M. Tenenbaum. "Interpreting line drawings as three dimensional surfaces. *Artificial Intelligence.* **17**, 75-116 (1981).

[9]F. Ulupinar and R. Nevatia. Shape from contours: SHGCs. *Proc.. IEEE Int. Conf. Comput. Vision*, Osaka, Japan, 1990, pp. 582-586 (1990).

---

[10]H. Sato and T.O. Binford, BUILDER-I: A system for the extraction of SHGC objects in an edge image. *Proc. DARPA Image Understanding Workshop*, San Diego, CA, pp. 779-791, 1992.

[11]M. Zerroug and R. Nevatia, Quasi-invariant properties and 3-D shape recovery of non-straight, non-constant generalized cylinders. *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, New York, 1993, pp. 96-103 (1993).

[12] Ari D. Gross. Recovery of SHGCs From a Single Intensity View. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 2, February 1996, pp. 161-180.

[13] J. Y. Zheng. Acquiring 3-D models from sequences of contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 2, Feb. 1994, pp. 163-178.