INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Computer-assisted three-dimensional rotoscoping for realistic image composition

Roble, Douglas Raymond, Ph.D.

The Ohio State University, 1992

Copyright ©1992 by Roble, Douglas Raymond. All rights reserved.
COMPUTER ASSISTED THREE DIMENSIONAL
ROTOPSCOPING FOR REALISTIC IMAGE
COMPOSITION

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

Douglas Raymond Roble, B.S., M.S.

* * * * *

The Ohio State University

1992

Dissertation Committee:
Richard E. Parent
Wayne Carlson
Roni Yagel

Approved by

Adviser

Department of Computer
and Information Science
To my parents and my sister.
ACKNOWLEDGEMENTS

I’d like to start off by thanking my advisers, Ed Tripp and Rick Parent. Both of them gave me all the help I needed without forcing me to do something their way. This made the process of getting this degree much more enjoyable.

I’d also like to thank some friends who’ve helped me along the way. Scott Whitman was a great friend who got his degree before me. It was a great help to watch him do it and avoid his mistakes. (I can make my own mistakes, thank you very much!) Manas Mandal has always been there making sure that the fonts are all correct and the format of the thing fits his exacting standards.

Finally, Deborah Shands has been an amazing amount of help during the last part of this degree. While I might have finished a bit quicker without her, I wouldn’t have had nearly as much fun doing it. It’s great to have someone standing beside you offering encouragement. Thanks.
VITA

November 16, 1962 ......................................................... Born - Detroit, Michigan

1984 ........................................................................... B.S. Electrical Engineering and
Computer Science
University of Colorado
Boulder, Colorado

1987 ........................................................................... M.S. Computer Science
The Ohio State University
Columbus, Ohio

1985-present ................................................................. Graduate Teaching Associate
Computer Science Department
The Ohio State University
Columbus, Ohio

Publications


Fields of Study

Major Field: Computer and Information Science

Studies in:

- Computer Graphics: Prof. Richard Parent
- Computer Architecture: Prof. Ponnuswamy Sadayappan
- Traditional Animation: Prof. T. Collins
# Table of Contents

DEDICATION .......................................................... ii
ACKNOWLEDGEMENTS .......................................... iii
VITA ........................................................................ iv
LIST OF TABLES .................................................. viii
LIST OF FIGURES ............................................... ix

CHAPTER PAGE

I  INTRODUCTION .................................................. 1
   1.1 General Problem Description .............................. 1
       1.1.1 An Example of Traditional Image Composition .......... 2
       1.1.2 Using the Computer for Image Composition ............. 10
       1.1.3 Computer Extracted Three-Dimensional Information ... 12
   1.2 The Thesis Statement ....................................... 13
   1.3 Structure of Dissertation ................................... 13

II  RELATED WORK ............................................... 15
   2.1 Image Feature Extraction ................................. 15
       2.1.1 Edge Linking .................................................... 17
   2.2 Three-Dimensional Reconstruction ...................... 18
       2.2.1 Rotoscoping ..................................................... 18
       2.2.2 Range Information ........................................... 19
       2.2.3 Stereo Vision ................................................ 20
       2.2.4 Perspective and Scene Analysis ......................... 21
       2.2.5 Models to Understand a Scene ......................... 22
   2.3 Image Composition ......................................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Edge Detection</td>
<td>166</td>
</tr>
<tr>
<td>5.3</td>
<td>Edge Connection</td>
<td>167</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Removal of Lone Pixels from the Edge Detected Image</td>
<td>167</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Locating an Area in the Image</td>
<td>168</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Centering the Seed Point in the Area</td>
<td>170</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Radial Search from the Seed Point</td>
<td>172</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Creation of Ordered List of Boundary Pixels</td>
<td>172</td>
</tr>
<tr>
<td>5.3.6</td>
<td>Location and Filling of Gaps in the Boundary</td>
<td>174</td>
</tr>
<tr>
<td>5.3.7</td>
<td>Problems with Edge Connection</td>
<td>178</td>
</tr>
<tr>
<td>5.3.8</td>
<td>Results of Edge Connection on the Example Image</td>
<td>179</td>
</tr>
<tr>
<td>5.3.9</td>
<td>Parallel Implementation of Edge Connection</td>
<td>179</td>
</tr>
<tr>
<td>5.4</td>
<td>LIVE: An Object Positioning Program</td>
<td>182</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Explanation of HP 9000 Environment</td>
<td>185</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Display of Images</td>
<td>186</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Placing the Model Objects</td>
<td>188</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Matching Silhouette Edges of the Objects</td>
<td>190</td>
</tr>
<tr>
<td>5.4.5</td>
<td>The Rotation Problem</td>
<td>198</td>
</tr>
<tr>
<td>5.4.6</td>
<td>The Solution to the Rotation and Silhouette Edge Problems</td>
<td>199</td>
</tr>
<tr>
<td>5.4.7</td>
<td>The Results of This Process</td>
<td>204</td>
</tr>
<tr>
<td>5.5</td>
<td>The Composition Process</td>
<td>206</td>
</tr>
<tr>
<td>5.5.1</td>
<td>The Ray Tracer</td>
<td>207</td>
</tr>
<tr>
<td>5.5.2</td>
<td>The Compositor</td>
<td>214</td>
</tr>
<tr>
<td>5.6</td>
<td>Chapter Summary</td>
<td>219</td>
</tr>
<tr>
<td>VI</td>
<td>CONCLUSIONS AND FUTURE WORK</td>
<td>221</td>
</tr>
<tr>
<td>6.1</td>
<td>Future Work</td>
<td>221</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Three-Dimensional Analysis</td>
<td>222</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Image Composition</td>
<td>223</td>
</tr>
<tr>
<td>6.2</td>
<td>Conclusion</td>
<td>224</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIBLIOGRAPHY</td>
<td></td>
<td>225</td>
</tr>
</tbody>
</table>
LIST OF TABLES

TABLE          PAGE

1  Summary of symbols used in Equation 5.1  163
# List of Figures

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>An actor filmed in front of a Bluescreen.</td>
</tr>
<tr>
<td>2</td>
<td>The model of the landscape that the actor will be flying over.</td>
</tr>
<tr>
<td>3</td>
<td>A matte that allows the actor to be seen.</td>
</tr>
<tr>
<td>4</td>
<td>A matte that allows the area around the actor to be seen.</td>
</tr>
<tr>
<td>5</td>
<td>Transferring only the image of the actor to another piece of film.</td>
</tr>
<tr>
<td>6</td>
<td>The actor and the landscape combined.</td>
</tr>
<tr>
<td>7</td>
<td>The two mattes produced by rotoscoping the landscape image.</td>
</tr>
<tr>
<td>8</td>
<td>The actor and the landscape combined with the cone object in front of the actor.</td>
</tr>
<tr>
<td>9</td>
<td>The actor, landscape and shadows combined.</td>
</tr>
<tr>
<td>10</td>
<td>Sample Digitized Image</td>
</tr>
<tr>
<td>11</td>
<td>Sample Image with Edges highlighted.</td>
</tr>
<tr>
<td>12</td>
<td>The edge detection algorithm output is an image with just the edges.</td>
</tr>
<tr>
<td>13</td>
<td>Finding gaps in an area can be compared to inflating a balloon somewhere inside an area. Where the balloon “bulges” out, there is the possibility of a gap in the area.</td>
</tr>
</tbody>
</table>
An example of the types of edges possibly generated by an edge detection algorithm.

In order to close the gaps, the slopes of the area/edge boundaries are calculated so that an probe line can be extended.

The edge detection output of the sample image after the edge connection algorithm has been applied.

When filming a scene, all objects must be measured so that a simplified three-dimensional model of the objects can be created.

An example of a scene with the model objects placed close to the reference objects. The model objects are the objects outlined in heavy black.

A three-dimensional model (the dotted lines) of the cube has been placed near the edges of the reference cube.

Comparison of a possible model object and a reference object. The corners of the model object do not exist in the reference object.

The silhouette edges of the reference object are calculated.

Examples of how the endpoints of a silhouette edge can extend beyond the edge of the matching reference object.

Probes are sent out from the midpoints of the edges of the reference object. The labeled probe has identified the wrong reference edge.

Using the probes, the two-dimensional adjustments for the corner points of the model object are calculated.

Calculating the vertex adjustment vectors is a three step process: (A) probing, (B) calculating the adjustments to the edges, and (C) intersecting the vectors to find the corner point.
An infinite number of points render to a specific pixel.

If the model object has been misaligned with the reference object by rotation, the vertex adjustment vector will be miscalculated.

Because of errors in the adjustment vectors, it might be impossible to find a position of the model object (the dotted lines) so that the silhouette corners exactly match the positions they are supposed to map to (the black with white dots) while preserving the shape of the model object.

By fixing the point where a single vertex must render, the search space for the best fit is reduced.

Because of digitization of the original image, aliasing errors will occur when calculating the position of a point in three space.

Soft edges around a reference object result in large edge detection lines. The grey pixels represent the optimal results of an edge detection. The black pixels are extra pixels added because of the soft edges of the reference object.

An example of the three-dimensional positioning error that can occur due to extra pixels added during the edge detection process.

The three basic steps to the procedure of correctly orienting an model object with a reference object.

A vector is sent from the eye point, through the reference point, and intersected with the sphere of possible positions.

Possible cases for the intersection of the reference point vector with the sphere of possible positions.

The center of rotation and the reference point location create an axis around which the object can be rotated.
Additional auxiliary probes are sent from the silhouette edge of the model object. There provide a difference in slope between the edges — unless a corner or a bump is encountered.

Once the model object has been positioned in three-dimensional space (the starting state), the vertices on the silhouette edge are moved to align the model edges with the reference edges.

The polygonal representation of the model object is triangularized and then the model object silhouette edges are matched to the reference object edges with more accuracy.

The translation of the vertices on the deformed object (represented by solid lines) from the initial points on the original model object (dotted lines) are used to calculate adjustments to interior pixels.

When filming a scene, the camera position, light positions and object sizes must be recorded.

The addition of a new object in the scene requires careful calculation of new shadows.

Using the rendered image of the model objects to locate shadows might lead to errors.

A reference scene generated using a ray tracing package.

The results of inserting another cube into the reference image of Figure 44.

Wire frame approximations of the actual objects are translated to the same three dimensional position as the reference objects.

Any new objects that are added to the scene are positioned in reference to the rotoscoped reference objects.

Any new objects are completely rendered. Only the visible portions of the object are rendered.
Each rotoscoped reference object is given a number. For each pixel in this image, the rotoscoped reference object that can be seen is recorded.

The ray tracing program produces an image which indicates which pixels in the original image are illuminated by ambient light.

For each light in the scene, the rotoscoped reference objects are rendered to determine the approximate positions of shadows and highlights.

The shadow created by the introduction of a new object in the scene is calculated.

The reference image of Figure 44 modified so that the cylinder has three colors and there is no specular reflection.

A plot of the colors of all the pixels that make up the cylinder in Figure 53 in color space.

Figure 44 modified so that there is a specular highlight on the cylinder.

A plot of the colors of all the pixels that make up the cylinder in Figure 55 in color space.

A black and white image from a 24 bit color digitized photograph of a face in water.

The model scene generated for the image in Figure 57. The main light source is near the camera, off to the right.

A plot of the colors of all the pixels that make up the face in Figure 57 in color space.

In order to group the pixels of the same base color, the color space is divided into slices around the source light vector.

The saturation angle is defined as the angle between the light source vector and the color vector of an individual pixel.
A plot of the magnitude of the color vectors vs. the saturation angles of the pixels that comprise the top part of the cylinder from Figure 55.

A vector parallel to the light source is subtracted from the pixel color vector to move the color to the base saturation angle.

A plot of the intensity of the pixels in the rendered model scene against the intensity of the same pixels that make up the top portion of the cylinder in Figure 55. The dotted line is calculated using a least squares fitting algorithm.

The new shadow and object have been added to Figure 55.

A digitized photograph of an actual scene. This will be used as an example image throughout the chapter.

A complicated concave object can be easily modeled by breaking it down into convex parts and measuring them.

The results of finding the edges of the example image.

A section of the edge detection image showing lone pixels.

An example of a seed point picked in a large area of the image.

Example of a rectangular area chosen to center the randomly picked point.

After the center of an area has been discovered, an approximately radial pattern of probes is sent out.

The edges hit by the radial probe are given numbers to establish a direction around the area.

A closeup of Figure 73 showing the difference between the types of gaps that might occur.
A closeup of Figure 73. When a gap is encountered, a probe line is sent based on the slope of the pixels up to the gap. ........................................ 175

A line is traced between the endpoints of the gap and new pixels are added. ................................................................. 176

The original image with the results of a clockwise gap fill. .......... 177

The results of applying the edge connection algorithm on the edge detected image of the example image. ......................... 179

Parallel Edge Connection Algorithm ........................................... 181

The LIVE object positioning and alignment environment. .......... 184

In the LIVE environment it is possible to display the original image and the detected edges. ................................................. 187

A roto-scoping technician has placed a cylindrical model object near the reference object. ................................................. 189

The model object cylinder has probes extended from its silhouette edges. They stop when they encounter a edge detection pixel. .... 191

An example of two intersecting objects. The black outline edges are the edge used in the alignment process. ....................... 194

The cylindrical model object is translated to a point where the silhouette edges match based on the silhouette probes. ........... 197

The silhouette edges of cylinder are “scrunched” so that they more accurately align with the reference object’s silhouette edges. .... 200

Model object silhouette edges are subdivided and matched with the reference object silhouette edges. ............................... 202

The interior points of the model object are adjusted to reduce the discontinuities of the scrunching process. .......................... 204
The entire scene is modeled through the process of placement, probing, translating, scrunching and mangling. .............................................. 205

The scene description file created by LIVE for the example scene. ... 208

The scene description file created by LIVE for the example scene. ... 209

The preliminary rendering of the example scene. ......................... 210

A object to approximate the control object is added to the scene. It is rendered by the ray tracer using faceted shading. ......................... 211

All the pixels that are covered by a specific object are identified in this image. .............................................................. 212

The model scene of the example image is rendered illuminated by the light. .............................................................. 213

The areas of the image that have new shadows falling on them are represented by this image. ...................................................... 214

The results of the composition process on the example image. .... 216

A control image photographed with a spray can in approximately the same place as the inserted cylinder. ............................ 217
CHAPTER I

INTRODUCTION

1.1 General Problem Description

A large part of special visual effects film-making is the integration of one filmed image with another. In general, when an actor walks across an alien landscape or encounters a fantastic, mythical creature, the actor and the effect are filmed separately. Then, using techniques such as bluescreen, the important parts of both images are extracted and combined onto one piece of film. Thus, the actor really looks like he or she is walking on the alien world when, in fact, the actor was on a soundstage in front of a specially colored screen.

However, the methods of doing this procedure are fraught with problems. The actor, when filmed, really isn't on the alien landscape. This means that if the actor should cast a shadow on an object that exists in the landscape, he or she won't! In fact, to achieve such an effect animators must analyze each frame of film and hand draw an approximation of the shadows cast by the actors in the scene. The problem gets more complicated when the actor moves to such a position where the landscape would cast a shadow on him or her. Or what if the actor is carrying a flashlight that illuminates a part of the scene? If these effects are not approximated, one loses a
sense of realism. When the viewer expects a shadow and none is seen, the illusion is destroyed.

The advent of computer graphics in the area of special visual effects has made some previously difficult effects very easy. In computer graphics, the computer deals with three-dimensional representations of the creatures and objects that are inserted into a scene. Because of this, if a shadow is cast on the object, the computer can easily calculate the correct way the shadow should fall on the object. It doesn’t have to be approximated by an animator. With this knowledge about the effects added to the scene, all that is needed is a three-dimensional representation of the original scene and all the shadows and lighting effects that occur when combining the effects with the actual scene can be calculated by computer.

This thesis will show that the computer, with a small amount of help from a technician, can extract three-dimensional information from a single photograph. This information can be used to more accurately composite images. If the three-dimensional position of a set of objects is known, and a new object is introduced to that set, the lighting effects on and by the new object can be calculated. This will add a new sense of realism and open up new avenues in film-making.

1.1.1 An Example of Traditional Image Composition

Consider a more detailed example. This example will serve to show how such effects are currently done and what would have to be done if these effects were accomplished with the use of a computer. In this example, the final image desired is an actor flying above a landscape of cubes and cones. To do this photographically using traditional
techniques requires many steps.

First, the actor is filmed in a harness in front of a specially colored screen called a "bluescreen". The color of the screen is precisely controlled so that later, during the development of the film, the color can be found and removed from the image.

At the same time, in a different part of the studio, the model of the landscape is filmed. No bluescreen is used at this point - there is nothing to be removed from the scene.

After the filming is finished, there exist two pieces of film and somehow the images on those separate pieces must be combined. To do this, mattes are created. A matte is a photographic negative made up of completely black areas and completely white areas. (Remember, all these images are being stored on slide film, not prints. So a
Figure 2: The model of the landscape that the actor will be flying over.

Figure 3: A matte that allows the actor to be seen.
Figure 4: A matte that allows the area around the actor to be seen.

completely white area in the photograph means that it’s clear on the film. ) Using bluescreen processing techniques two mattes can be created that either stop the image of the actor or allow him to be seen. See Figures 3 and 4.

Now, using the mattes, the images of the actor and the landscape are transferred to another piece of film. First the photo of the actor and the matte that allows only him to be seen are sandwiched together and a light is shown through them to expose the new film. Since light can only penetrate where the matte is clear, only the image of the actor is imprinted on the new film. This process is shown in Figure 5. Then, the landscape photo and the matte that only shows what isn’t the actor are sandwiched together and a light is shown through it to expose the previous piece of
Figure 5: Transferring only the image of the actor to another piece of film.

film. This exposes all the non-actor area of the photograph with the landscape image. The resultant image is shown in Figure 6.

There are problems with this image though. What if what was desired was to have the cone-shaped object in front of the actor? One way of solving that problem involves more photography and more mattes. The landscape can be photographed in two parts. The landscape without the cone-shaped object could be photographed as it was before. Then, the cone-shaped object could be photographed in front of a blue screen so that another set of mattes could be produced from it. Now, combining all the images on another piece of film become more complicated, but it can be done.

The above method has problems too. What if the model was constructed in such
a way as to have the separation of the cone-shaped object from the rest of the model impossible? Somehow a matte would have to be generated for that object without resorting to bluescreen processing techniques. The most common way to do this is to draw, by hand, the matte for the object. This is done by projecting the frame of film onto a piece of paper and tracing the outline of the object to be matted. Then the outline could be filled in to produce one matte, and a negative photograph of that filled in image would be the other matte. (See Figure 7.) The technique of projecting an image on a piece of paper and tracing the image is called rotoscoping.

Now, using the mattes and the original photographs, a better image can be produced (Figure 8). The cone-shaped object is clearly in front of the actor and it
Figure 7: The two mattes produced by rotoscoping the landscape image.

looks like he is really flying through the scene. Or does it? From the way the scene is lit, one can tell that the light source is coming from the right side of the photograph. If this was a real photograph, shadows cast by the actor falling on the objects in the landscape would be reasonable to expect. But, none are in the image. And it is easy to see why. When the actor is composited into the image, it is done as if he is a two-dimensional object. The landscape lighting won't affect him and he won't affect the landscape lighting. It is impossible for the special effects technicians to accurately calculate where the shadows should occur from the information contained in the photograph.

The way the shadow problem is solved in practice is to resort to rotoscoping again.
Figure 8: The actor and the landscape combined with the cone object in front of the actor.
The image of the actor combined with the landscape is projected onto a piece of paper and artists make a "best guess" as to where a shadow should fall. By drawing the shadows on a piece of paper and photographing them, they can then be combined with the image. The final result is shown in Figure 9.

1.1.2 Using the Computer for Image Composition

By introducing a computer into the problem of image composition, a number of the problems inherent in photographic composition of images can be reduced. Unfortunately, the computer also introduces a number of problems as well. If the computer is to be used in the composition of the images, the images must first be converted.
into a form that the computer will understand. Special devices called "digitizers" or "scanners" can optically scan an image and encode each spot on the image as a set of numbers representing the color at that particular spot. Once this is done, the computer can manipulate the colors of the image in any way imaginable and then, using an optical printer, the image can be transferred to film again.

One of the main reasons not to use a computer to do the composition of images is that the resolution of the scanners is typically much less than the resolution of the film. In other words, the film contains more information per square inch than the computer can read. If you scan in an image and then print it directly to another piece of film, without manipulating the image in any way, the new piece of film will contain less information than the original. They will not be perfect copies. If you transfer the image using photographic techniques — shining a light through the original piece of film and exposing another — there will be a small amount of information lost, usually much less than occurs with a scanner. There will be some degradation though!

The degradation of the image through photographic transfer techniques highlights one of the advantages of computer composition: once the image is scanned in, it can be combined and recombined with other images without losing any more information. Doing this photographically will result in more and more image degradation each time an image is transferred from one piece of film to another.

So, if it is decided that the loss of information during the initial scanning of the image is acceptable, and the amount of image manipulation is large, computer composition is the probably the method to use. Also, the technology of image scanners
is increasing daily, and the amount of information loss is getting smaller and smaller.

Once the images are scanned into the computer, composition of images can proceed much the same way the photographic composition occurred. Instead of chemically locating the blue areas of the image to create a mask, it can be done mathematically. Then, with computer mattes, the images can be combined the same way they were in the previous section.

1.1.3 Computer Extracted Three-Dimensional Information

Using the computer, instead of photographic techniques, doesn't solve all the problems though. In Section 1.1.1 a rotoscoped matte had to be created to place the actor "behind" the cone shaped object. Also, any shadows that the actor cast on the surrounding objects had to be guessed at by artists and drawn in by hand. If it were somehow possible to calculate the three-dimensional positions of all the objects in the scene, standard computer graphics rendering techniques could be used to properly assemble the picture. The computer could decide at each point in the image what should be visible. In the example, if the position of the cone and the actor were known, it would be trivial to calculate that the cone was closer to the camera than the actor and that it should be seen, not the actor. Also, if the position of the light sources are known, the addition of new objects to a landscape during the matting process can cause appropriate shadows to fall on the landscape. Even new light sources could be added to the scene and their contributions could be calculated. This would give the cinematographer a great deal of flexibility in lighting the original shot — the lighting could always be fixed as a post-processing step.
Extracting three-dimensional information from a single photograph has long been a goal of researchers in the fields of computer vision and robotics. It is terribly difficult, however. It is difficult for the computer to recognize an object, much less locate its three-dimensional position. It was assumed at the beginning of this project that the computer could not, unaided, extract the necessary information from an image to accomplish the above goals. However, if, during the analysis of the image, an operator of the program gave the computer a “hint” or a starting point on where to start looking for three-dimensional information, the computer could achieve the goals set out for it. The exact details of this procedure will be discussed in the next chapter.

1.2 The Thesis Statement

The following is the thesis statement that guided my research and development of this project.

By modeling objects in an image with simple shapes, it is possible to extract three-dimensional information from a photograph with limited support from a user. Using this information additional images can be incorporated into the original image in a realistic fashion.

1.3 Structure of Dissertation

The First Chapter introduces the problem and explains the overall concept of the thesis. Previous work related to this thesis is discussed in the Second Chapter. In
the Third Chapter, the algorithms and mathematics of extracting three-dimensional information from an image and using that information to add new objects to the scene are explained. The Fourth Chapter covers the concepts and mathematics behind the realistic composition of a new object into the digitized photograph. The Fifth Chapter discusses the implementation of all the programs developed for this thesis and presents the results of a demonstration experiment. Finally, in the Sixth Chapter, conclusions are drawn and future research is discussed.
CHAPTER II

RELATED WORK

This thesis deals in topics of image processing, image understanding, three-dimensional
reconstruction using models, image composition and lighting analysis. Many of these
topics have extensive backgrounds in computer vision and computer graphics. This
chapter presents a review of the previous work in these fields.

2.1 Image Feature Extraction

Before any matching between model objects and the reference objects in the digitized
image can begin, the reference image must be analyzed and the silhouette edges of
the different objects must be extracted. Much work on this problem has been done
in the field of computer vision and image processing.

The search for the silhouette edges of objects begins with the search for points (or
pixels in the image) that fall on the edge of an object. Many edge detection algorithms
have been developed to find edge points. The basics behind edge detection can be
found in the paper by Levialdi [24] and in the books by Rosenfeld [32] and Shirai [36].
The basic concept behind most edge extraction procedures is to apply a filter to the
image that enhances the high frequency portion of the image and minimizes the low
frequency portion of the image. It is assumed that the edges between objects will be
portions of the image that are changing rapidly and hence have a large high frequency component. The algorithms differ in the manner of the filter and how effective it is.

Edge detection algorithms generally calculate the gradient of change in intensity (or color) over a local area. Because the image is composed of discrete pixel elements, a continuous gradient cannot be used and a discrete gradient must be developed. A typical operator that calculates the gradient (developed by Prewitt [26]) might look like this:

\[
H_1 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad H_2 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}
\] (2.1)

By convolving the first operator, then the second with the image, edges in all directions will have been enhanced. All that is left is to apply a thresholding algorithm to the resultant image to separate the edge pixels from the non-edge pixels. All edge detection algorithms are vulnerable to noise in the image. Before any filter is applied to the image, the image should be smoothed first.

More advanced algorithms have been developed. In Fleck's papers [15] and [14] and Jain's paper [20] the advantages and disadvantages of the many attempts at finding edges in an image are discussed.

In this project the contributions made came after the edge detection step. Because of this, an edge detection algorithm was chosen that was easy to implement and gave good results. While most edge detection algorithms assume a black and white image, this project was geared towards color images. Thus an edge detection algorithm that made use of the extra information available in a color image was chosen. We used the algorithm by Shiozaki [35]. It was simple to implement and provided acceptable
results. Any other edge detection method (such as the one presented by Cumani [9]) that provides reasonable results would work for the purposes of this paper.

2.1.1 Edge Linking

Once the pixels that fall on an edge in the image have been found the task is not complete. For this project it was necessary that the edges found must be whole and complete. There can be no gaps in the edges. Unfortunately, in edge detection algorithms this is a common occurrence. The intensity of edges varies in an image and one section of an image might result in an edge while another section of the image won't result in an edge, even though both sections contain parts of the same edge. Roberts [28] developed an edge linking technique that examined the local direction of a group of edge points and matched them with a nearby group of edge points that had, to within a threshold, the same direction. Zucker, Hummel and Rosenfeld [33] proposed a method where edges could be linked using a relaxation technique. In this technique, each pixel — based on the locations of neighboring pixels — is assigned a direction. This labeling defines a measure of compatibility between pixels. By recursively relaxing the acceptable compatibility between pixels, a linkage between pixels can be found. The main problem with these methods is that gaps are still left in the edges found in the image. For this project some way to guarantee that no gaps would be left in an edge had to be found. Such a procedure is presented in Chapter III, Section 3.1.2.
2.2 Three-Dimensional Reconstruction

Extracting the three-dimensional shape and position of objects in an image has been a problem that has received a large amount of attention from researchers in the field of computer vision. Being able to identify what an object is and where that object is located is very important to many diverse fields. In the following sections we will discuss the various approaches have been taken: three-dimensional range data accurately locates objects and results in more information to identify them; stereo vision can provide three-dimensional information that can be used in a similar way; the shape of the objects can be inferred from the lighting of the objects in some cases; and finally, the tack taken in this research, three-dimensional models can be used to guide the recognition of objects.

2.2.1 Rotoscopying

Long before the computer was applied to the concept of image understanding film-makers were doing a very simple form of image analysis. For King Kong to pick up Fray Wray either a life sized monster had to be created (very difficult) or different sections of two separate images had to be composited together. To do this, each frame of the film would be projected onto a registered piece of paper. A technician (who was also part artist) would carefully trace around the interesting parts of the image. When each frame of the scene had been analyzed this way, they had a complete record of where the important objects were in each scene. This record could be used to composite images in various ways. This is referred to as rotoscoping and a
large library of techniques associated with it are possible. (See the book by Imes [19] for an excellent explanation.)

Unfortunately, this technique does not provide any information about the three-dimensional information in the scene. In the film *Who Framed Roger Rabbit* [13], actors and animation were seamlessly integrated using this technique. Unfortunately, because there was no three-dimensional information in the rotoscoping of the scenes, all the shadows that the animated characters cast on the actors had to be approximated by hand by the animation artists.

### 2.2.2 Range Information

Range images are created using devices that not only take a standard photograph of a scene but also calculate the camera-to-object distance at each pixel. The distance a point is from the camera can be measured in different ways: a laser can illuminate the object and the time the laser takes to travel from the camera and back can be calculated; a similar method uses an ultrasonic pulse (like the ones in some autofocus cameras) bounced off the object; alternately, a spot of light can be projected and detected from another angle — the position of the measured spot can be used to triangulate the distance to the object. (The last technique can be extended to a light stripe method that uses a band of light instead of a single spot.) All of these methods are explained in Shirai's book [36].

In this thesis, we present a method to extract three-dimensional information from a photograph. With a range image, this information is measured automatically. For the purposes of this thesis, it would solve many problems. Unfortunately, all the methods
of measuring range data are prohibitively slow and most of them are intrusive to the normal photograph. (Only ultrasound does not illuminate the image in some visible way. Any such illumination will ruin any photograph being taken at the same time.) The range data is also too slow to be used in any situation where the object is moving.

2.2.3 Stereo Vision

By observing the same object from two different positions, its three-dimensional location can be calculated. If two cameras are set up and synchronized and the distance between the two of them is carefully measured, the three-dimensional location of a point in the image can be calculated in a relatively straightforward way. (See the book by Shirai [36].) Once the same point on both images is found and identified, the calculation of its position is simple. Unfortunately, finding of the same points on the two stereo images is quite difficult. The two images are necessarily taken from different vantage points. A feature in one image might not be visible in the other image and the different perspectives on the same object yields features that are not the same size and shape. It is difficult to find matching points in this case.

Shirai [36] and Gagalowicz [17] propose different methods of finding correspondence between two features. Shirai proposes a method which considers a feature in one image, and creates a window where the corresponding feature may be found in the other image. Inside the window, various calculations are performed to see if an unambiguous correspondence between features can be found. If none can be found, the size of the window is decreased until an unambiguous match is found or a limit is reached. If the limit is reached, the feature is not visible in both images. Gagalowicz
takes a different approach by looking for regions in both images that have similar homogeneous properties. Through a variety of techniques these regions are matched and from them, the three-dimensional location of the regions can be found.

In this thesis, the added difficulty of filming everything in stereo was considered unacceptable. If the image was filmed in stereo, the techniques developed for stereo vision could aid the process described here.

2.2.4 Perspective and Scene Analysis

Research done in computer understanding of scenes has usually made a simplifying assumption: because it is easier to work with, orthographic projection is considered in most cases. However, perspective projection contains a large amount of information and this can be used in the analysis of scenes. In the paper by Barnard [3], methods of determining the 3-D spatial orientation of curves and surfaces are presented. In this work, vanishing points are found and from them, the geometrical properties of objects with parallel lines can be calculated. Other techniques for calculating the orientation of planes are discussed.

The paper by Kanade [22] presents methods for using Origami theory and mapping image regularities into shape constraints to recover the shape of objects in the image. Augusteijn discusses similar techniques in his paper [2].

Using the shading of an object to discover the shape of the object was discussed in the paper by Ikeuchi and Horn [18]. By assuming that a surface is smooth and using an initial assumption of a surface normal, the normal at all points along the surface can be built by measuring the change in lighting at neighboring points.
All of these papers (and others in the field of scene understanding, see [31, 36]) attempt to discover facts about the scene with little or no a priori knowledge of the scene. While they are able to make many strides in the understanding of the scene, a complete model of the scene is not created.

2.2.5 Models to Understand a Scene

In computer vision, just being able to list the objects that are visible in a scene is a difficult task. Two-dimensional models of a scene can be matched against the scene itself and, by using relationships between regions, an understanding of the scene is created (Tenenbaum and Barrow [37]). Experiments along these lines are not very useful for the purposes of this thesis, they provide no information as to where the object is located, only what objects are in the scene.

Three-dimensional models have been used to understand an image. The system ACRONYM developed by Brooks [5] is one such system. In this system, three-dimensional models of possible objects in a scene were developed. The models were based on volumetric cones. Cones were simple enough to approximate most objects and easy to analyze. A graph is built of the relationship of objects with sub-objects so that more complicated objects can be represented. The image is then analyzed for "ribbons" which will be matched to the possible cone models. It was shown that in an overhead view of an airport, airplanes could be recognized. The system is limited in that the objects are constrained on a plane and three-dimensional rotation is not allowed. The objects may not change their shape and must maintain a certain orientation for the system to work.
Other model-guided image understanding systems are presented in the book by Shirai [36] and the papers by Gagalowicz [17] and Baur [4]. The main goal of all these systems is to recognize objects in a scene. In this project the human technician locates the objects. We are more concerned with finding the exact position of the object without having to resort to intrusive or slow ranging techniques.

2.3 Image Composition

Compositing two images together has been attacked in various ways. As described in Chapter I, the bluescreen technique is a useful, purely photographic way of compositing two or more images together. Imes [19] describes the possibilities of the bluescreen.

Porter and Duff [25, 10] have developed a system of compositing digital images that have been rendered separately from each other. By storing a “coverage” channel with the normal color channels for each pixel, two images can be composited together without any aliasing artifacts. This system was intended for images produced by different renderers and was done in a purely two-dimensional manner — no interaction between the images is allowed at this stage.

Burt and Adelson [6] decompose images into a set of band pass filtered component images. By matching the component images with similar frequency characteristics and smoothly interpolating between them, the modified image components can be summed to create a seamless integration of two images. This provides a excellent way of putting images together as mosaics — the seams between the images are completely removed.
In 1986, Nakame [12] developed a system for creating architectural montages. In this system, a new architectural object is inserted into a photograph in a realistic fashion. The advantages of this system are that the building is accurately rendered with the light source (the sun) at the correct position; by adding fog effects in the rendering process, the building appears to be at the appropriate distance and anti-aliasing was done to integrate the building with the backgrounds. This system does not accurately model the three-dimensional nature of the objects and terrain in the photograph. The image must be divided by hand, in much the same way traditional rotoscoping is done, to separate the foreground from the background. When the new building is laid into the new photograph it does not affect the surrounding terrain in terms of lighting.

In a demonstration video tape produced by Alias Research Co. highlighting their software package "Upfront" [7], the presentation demonstrates a method for adding new images to a digitized background. By using lighting cues that already exist in the scene and by a trial and error approach to calculating the viewing parameters, they are able to add new objects to the scene so that they are lit in a similar manner. There are no facilities to extract three-dimensional information from the digitized image, the "Upfront" package is used merely to composite new objects on a two-dimensional background.

Anderson, in his Master's Thesis [1], proposes a system similar to the one developed in this thesis. By carefully measuring everything in the scene, Anderson built a complete three-dimensional model of a photographed scene. Then, using techniques
to model a camera lens, he showed that a replica of the photographed scene could be rendered using ray tracing techniques. He then proposed a technique where the ray tracing program could generate a matte similar to the ones used in the blue screen process and new objects could be photographically inserted into the scene. His system had drawbacks in that every detail of the scene must be measured and that none of the objects could be moving. He proposed using intrusive three-dimensional measuring techniques or stereo vision to extract the three-dimensional position from the objects in the scene. Also, the shadows added to the scene were done using standard photographic techniques. It wasn't possible to accurately match the new shadows with the shadows already in the scene.

Finally, in a recent paper, Thirion [38], develops a system to view aerial photographs from different perspectives. The system, like Anderson's, requires an exact model of the photographed scene. The exact shapes, sizes and positions of the objects in the scene are recorded. (As this system was intended for photography from altitude of buildings, this is reasonable.) The image is mapped onto the model of the scene and a texture map is created for the object. The scene can then be viewed from different angles. Because there is no alignment between the model object and the scene, the model object must be very accurate or part of the scene will be mapped onto an object when it shouldn't be. Also, Thirion develops a simple way of removing shadows from an image created this way. His method is again vulnerable to misalignment of objects and does not take into account the reflective properties of the objects being illuminated. For the high altitude photography these problems were considered
acceptable.

In the last couple of years, computer effects have been used in films to create realistic special effects. Notably, in the *The Abyss* and *Terminator II*, computer effects were composited with live action film in such a way that the lighting of the environment was affected by and had an effect on the computer graphics element (the water tentacle in *The Abyss* and the shiny second terminator in *T2*.) These were done using a combination of traditional and new methods. The reflections on both objects were created by making a reflectance map (see Foley [21]) of the scene filmed from different angles and using it during the rendering of the computer graphics object. The shadows and lighting cast on the actors already in the scene were created using traditional photographic techniques or simple shadowing similar to the one used in Thirion's paper. The effects are explained in articles in Cinefex [34, 11].

### 2.4 Chapter Summary

In this chapter, previous work in the areas of computer vision, image understanding and image composition is discussed. The understanding of the size, shape and position of objects in an image has received considerable attention from the field of Artificial Intelligence, and Robotics. By simplifying the problem with the addition of a human technician, as is done in this thesis, the problem becomes more tractable. Also, while some work has been done to add new shadows and lighting effects to an image, the necessary analysis of the lighting in the scene has not been done. This thesis shows methods of three-dimensionally locating all objects in a scene without the use of ranging devices and stereo vision. It also shows ways of adding new lighting effects
that take into account the nature of the material and lighting that is already in the scene.
CHAPTER III

EXTRACTING THREE-DIMENSIONAL INFORMATION

This chapter discusses the mathematics and algorithms of calculating the three-dimensional positions of objects in a photographed scene. The analysis of the image, the interaction with the user and the computer calculations involved are discussed.

In order to place a three-dimensional object into a scene, three-dimensional knowledge of the scene itself must be known. In the real world, human beings have many methods of determining the position of objects in three space. Two of these are usable in terms of computer graphics: a stereoscopic vision system and a knowledge of the size of the objects. The method of determining position with stereoscopic vision is well known; if each eye sees the same point at a slightly different position, basic triangulation can be used to calculate the three-dimensional position of that point from the eye. However, since the eyes are relatively close together, this technique is useful only if the point or object is close to the viewer. Objects farther away cause no discernible difference in orientation of each eye and the triangulation method results in an answer of "far away." That's when the second distancing technique comes into play. Through experience, most humans learn the apparent size of various common objects at many distances. Thus, they can, by comparing the observed ap-
parent size and the remembered apparent size of an object at many distances, make a decent guess as to the actual distance of the object from themselves.

There are other visual cues that humans use to determine the positions of objects: if an object covers or obscures another object, the first object must be in front of the second object; as objects get farther away from the eye, the detail on the objects is harder to pick out; atmospheric and lighting effects also give clues as to how far away an object is. All these visual cues are used to build up an understanding of the environment that a person is currently standing in, however, they are not tremendously accurate measurements of the distance of the object from the viewer. Except for the case of one object obscuring another object, these cues are also hard to quantify. The detail visible on an object depends on the atmosphere, the quality of the lens of the camera and the quality of the film (or eyesight of the person). All of these are very changeable parameters and very difficult to measure. Therefore we are brought back to the two options: stereoscopic photography and size comparison.

While stereoscopic photography does exists, for the purposes of this problem, it is considered a luxury. In special effects film-making, stereo filming uses twice the film and, because of the complexities of the stereo cameras creates additional problems of filming. (Consider a lens with a long focal length, when filming with such a lens only certain objects are in focus. Objects farther away and closer than the focused objects are fuzzy. Because computation of three-dimensional position from stereo photographs requires visible detail in the images, only the positions of the focused objects can be calculated.) So, we limit the problem to calculating three-
dimensional information based on non-stereo images. With this limitation, the only method available to calculate the three-dimensional positions of the objects is the apparent size of the object.

In this chapter we will explain how, if the size of an object in an image is known, an approximate three-dimensional computer graphics object can be created to model the image object. Also, if the camera position is known, the model of the object can be manually placed “near” the object in the image. Once that is done, the computer can automatically align the model to the image object and then calculate the distance from the object to the camera.

There are various tasks to be completed in the solution of this problem. The first of these is to extract usable information from the digitized image. Since the size and shape of the object is to be used to calculate the distance the object is from the viewer, the outline of the object is computed. This is done using an edge detection algorithm. However, edge detection algorithms tend to miss faint edges. Since the complete outline of the object is important, an additional step of edge connection is done. Edge connection closes all the gaps in the outline of the objects.

Once the image has been analyzed and the models of all the objects in the image have been created, a human operator places the objects near their digitized counterparts in the image. Then, the silhouette edges of the computer model objects are matched, through an elaborate probing process, with the edges of the silhouettes of the objects in the image.

After the objects have been positioned as closely as possible to the actual position
of the objects in the scene, their silhouette edges should line up. However, since the computer models of the objects are only approximations of the real thing, there will be some error. To alleviate this in the compositing of images, the computer model is "shrink-wrapped" to the silhouette of the original objects. This deformation of the object is then applied to the surface of the computer model object to give it an approximation of the contours that exist in the actual object.

Once these tasks have been finished, the computer has a model of the actual scene where the positions of each object are known (to a certain amount of error), the modeling objects match the silhouette of the original objects exactly, and the modeling objects have been deformed to approximate the complexity of the surface of the actual objects.

3.1 Analysis of Image

To align the modeling object to the objects in the image, usable information must be extracted from the original image. The most useful information, for the purposes of this task, that exists in an image are the edges that objects create. Much research has already been done on the best way to detect edges in an image. (See Chapter II, Section 2.1.)

For this task, the edge detection algorithm must satisfy the following criteria: it should find all the major edges in the image, find a minimum of false edges, and all the edges that are found should be complete, connected and without gaps in them. After an extensive search of the literature in the field, an algorithm that satisfied all these constraints could not be found. So, an algorithm that examines the existing
edges in an image and connects any gaps that might exist was developed. There are some problems and limitations with this new algorithm, but in general it does a very acceptable job. Since the algorithm was expected to be slow, (It turned out to take about 15 minutes on a single Butterfly node to processes a 256 x 256 image.) it was implemented in parallel on a Butterfly Multiprocessor computer.

For the following discussion, a sample image will be used. For pedagogical purposes the image is a simple one, consisting of a cube and a cylinder. (See Figure 10.)

![Figure 10: Sample Digitized Image](image)

For this discussion, the objects in the image will be referred to as *Reference Objects*. The purpose of this task is to attempt to calculate the three-dimensional positions of these reference objects in the image.
3.1.1 Edge Detection

Of the many edge detection operators, the one proposed by Shiozaki [35] seemed to do the best at finding the major boundary edges of objects. The entropy operator that he developed has the advantage that its sensitivity changes according to the average brightness in a local region. With this, it is able to find edges in areas where other operators couldn’t. Also, Shioaki has specifically addressed color images in his algorithm. This project was designed exclusively for color images and his algorithm seemed very suited to the problem.

Basically, the entropy operator is a nonlinear spatial filter and it calculates the entropy of brightness in the neighborhood of a pixel. The entropy is small when the change of brightness in the region is severe and is large when the change is smooth. Once the operator has been applied to each pixel in the image, the edges can be brought out using a thresholding technique. It is noted in the paper that the algorithm is subject to noise, so it is important to remove noise beforehand. For a more complete description of the algorithm see Shiozaki’s paper.

After the initial edge detection is executed on an image, an image with just the detected edges of the image is produced. In general, this image will have some standard problems associated with edge detection: false edges will have been found, some edges will have been missed and some edges will be incomplete. In order to align the model objects, the silhouette of each object in the image be surrounded by a complete edge. An algorithm is needed that examines the output of the edge detection of an image and “fixes” the edges that have been found. For this, an edge
connection algorithm was developed.

3.1.2 Edge Connection

The basic concept behind the edge connection algorithm was to examine the results of an edge detection algorithm and "connect up" any holes that are found in the image. A hole would be considered a gap in a line. But, unfortunately, the lines of generated by the edge detection algorithm are not perfect — they are oftentimes many pixels thick and bumpy — so a gap is defined as a point where a set of pixels ends without changing directions. If a line of pixels turns a corner, from certain perspectives it looks as if that line just ended. In this case it does not constitute a gap.

To discover these holes, the image must first be filtered of all extraneous lines.
Edge detection algorithms are notorious for leaving small specks and points on image where some small deviation in texture caused it to think a line was there. It is trivial to make a filter to remove these. Once the image is cleaned up, the process of fixing the holes in the image begins. First, the holes must be found. Unfortunately, the algorithm has no knowledge of the locations of the objects in the scene and hence can not make decisions on the existence of holes based on the physical properties of the objects in the image. In other words, the algorithm must work without the knowledge of the shape of the objects. This makes the location and correction of holes much more difficult and problematical. After the holes are found, connecting them consists of finding the endpoints of the
gap and drawing a line between them.

An algorithm that performs well on various test images is presented. The exact details of the implementation of the algorithm will be described in Chapter V; at this point an algorithmic outline of the process is discussed.

3.1.3 Finding the Holes in the Outline of an Object

The computer, when looking for holes in objects, does not know that a set of pixels makes up the silhouette of an object. The algorithm is unaware of the concept of an object. For example, consider a random mass of lines on a piece of paper. Given the task of making sure that there were no holes in a potential silhouette of an object described by the lines, a person might connect the end points of each line to every other line. More likely, the person would make a guess and connect lines to other lines that are close to the original line, or traveling in the same direction. This might introduce extra lines in the image, but at least all possible gaps would be filled.

In terms of the algorithm, the best that can be achieved is to make all areas described by the edge detection convex (or at least close to convex.) To better illustrate this, imagine picking a random starting point in the middle of some area. (The details of choosing the best starting point will be described in Chapter V, Section 5.3.3.) From this point, a two-dimensional balloon is inflated. If, at some point, the balloon stops getting bigger, the area is completely connected and gap free. If, however, the balloon keeps getting larger and "bulges" out of gaps in the area, these bulging areas are a good indication of a gap. (See Figure 13.)

What constitutes a "bulge" and hence a gap? When inflating the balloon, there
Figure 13: Finding gaps in an area can be compared to inflating a balloon somewhere inside an area. Where the balloon "bulges" out, there is the possibility of a gap in the area.
is a definite direction of inflation. When an edge is encountered, the balloon flows along the edge in the direction of inflation. If, at some point, the balloon must flow in a direction contrary to the general flow to maintain contact with the edge, it starts bulging. This occurs when a line terminates and the balloon must flow around the tip of the line to maintain contact (Point A in Figure 13). It also occurs at convex corners of objects—which is unfortunate—because they will also cause a bulge in the balloon and an edge gap which will then be filled (Point B in Figure 13).

There is an addition problem in that the edges generated by edge detection algorithms are not always exactly one pixel wide. Also, the lines that are generated do not necessarily follow Bresenham's line generation algorithm for producing lines that are well behaved and optimal. Figure 14 demonstrates the various types of lines producible from an edge detection algorithm.

In Figure 14, a section of an image has been edge detected and enlarged. Four possible results of the edge detection are shown. The first edge, Edge A, is the most perfect edge one could hope for—it is one pixel thick and four-connected. Edge B is still acceptable, it is one pixel thick but only eight-connected. Unfortunately, Edge A and Edge B are hard to come by in edge detection algorithms. This isn't the fault of the algorithm as much as it is the fault of the image being analyzed. Typical photographs don't have sharp edges around the objects in the image. Either the photograph is slightly out of focus or the object itself is slightly fuzzy. Therefore, Edges C and D are more likely encountered in a typical image. Edge C is an example of a complete edge, but more than one pixel thick. It causes problems in that it
Figure 14: An example of the types of edges possibly generated by an edge detection algorithm.

is very hard to tell the direction or slope of the edge. If the edge is complete and connected, as in Edge C, this doesn't cause much of a problem. However, as shown in Edge D, if the same edge has a hole or gap in it, it becomes difficult to decide how the gap should be sealed.

If we return to the balloon analogy, it can be seen that the edge isn't as important as the boundary between the edge and the area. When inflating the balloon, if doesn't matter if the edge is one pixel or three pixels thick. It will still stop the balloon from expansion. Also, the area/edge line can be used to calculate a more specific direction or slope than the edge itself. If the edge is thick, it's very difficult to compute the direction or slope of the edge. This is not the case with the boundary between edge and area, it is always infinitely thin and the slope of it is easier to calculate.
3.1.4 Filling the Holes in the Outline of an Object

Figure 15: In order to close the gaps, the slopes of the area/edge boundaries are calculated so that an probe line can be extended.

Using an algorithm that approximates the expansion of a balloon, the gaps in the edges that surround an area are marked. To close the gap, if a naive approach is taken, one could just extend the edge until it encounters another edge. This would only work in the case of an object with no curves or corners. If the outline of the object curves away at any point, the extended line may never encounter its appropriate endpoint. Instead, using the calculated slope, an imaginary line is projected from the edge. Then, traveling around the area in the same direction as the line, the other pixels that have been marked as part of the boundary of the area are examined. The first pixel that is closer to the center of the area (determined at the beginning as the point
to start inflating the balloon) than the line is chosen as the second endpoint for that 
gap. The line is drawn and the next gap is analyzed. This assumes that objects are 
composed of basically convex areas. A concave object will have its largest convex 
area closed and then the remaining areas will be examined anew. This introduces 
new edges to the interior of the object but this is acceptable as long as the silhouette 
of the object is analyzed and closed correctly.

It turns out that the larger the area one starts with, the better this algorithm 
performs. If small areas are connected at the beginning, they generate more small 
areas and the image gets filled with "lace". So, each possible starting point for an 
area is examined and a list of the area sizes is made. Then, the list is traversed from 
largest to smallest. The list is also updated from time to time since filling one area 
may change the size of areas around it.

3.2 Generation of Model Object

There is still not enough information in the image to calculate the three-dimensional 
positions of the objects in the scene. If various areas of the image are identified as part 
of a single object, the area of the image that the object takes up can be calculated. 
This information alone does not give any three-dimensional positional information 
about the reference object. If a photograph of a large object far away from the 
camera is compared with a photograph of a small object very close to the camera, the 
area both objects take up on each photograph might be the same. It is only when the 
actual sizes of the original objects are known that the three-dimensional positions of 
the objects can be calculated. To do this, a model of the scene must be created and
Figure 16: The edge detection output of the sample image after the edge connection algorithm has been applied.
aligned to the actual image. This task requires that when a photograph is taken or film is shot, all the important objects in the shot must be measured. The knowledge of the approximate size and shape of each object must be known to achieve acceptable results in the image.

In Figure 17 a simple drawing of a scene with a mailbox, a street light and a wall is shown on the right. To calculate the positions of the objects, their size must be known. The dimensions and shape of the mailbox, street light and wall must be measured and recorded. The more exact the better, but an approximation is acceptable—the algorithm will "shrink-wrap" any approximate model around the actual image to make sure that the outline of the image and the outline of the modeling object match. But, the more detailed the measurement, the less approximating the algorithm must do to calculate the position of the object in space.

This also applies to people in the image. Any actor or actress in the image must be measured. Once again, the more accurate, the better. The actor's height and width, and the circumference of his or her limbs should be recorded. There exist devices that, by scanning the subject with a laser, can create a three-dimensional model of the subject. These would be ideal to produce the kind of models the algorithm is looking for but this accuracy is not essential for acceptable results.

Once the object, be it a mailbox or a person, has been measured, a three-dimensional representation of it must be produced. Using simple objects such as spheres, cylinders, boxes and cones, a data generation program can produce a model of any object. Once again, the model generated doesn't have to be exact. If it isn't
Figure 17: When filming a scene, all objects must be measured so that a simplified three-dimensional model of the objects can be created.

possible to completely model the object (wrinkles in clothing would be particularly challenging) it won’t matter that much. The model of the object will be adjusted to fit the actual outlines of the object. Figure 17 shows an example of this. By combining these objects, scaled correctly for the object they are to model, a simple model of the object in the scene can be developed. For the rest of this discussion, these objects will be referred to as Model Objects.

If the reference object being modeled is flexible and movable — as an example, the limbs of a person — the model object can be created so that the simple objects that compose the model object are attached together in a hierarchical manner. In this way, not only can the human operator place the model object near the reference object, but he can also adjust the movable parts of the model object so that the two objects are bent and flexed in the same way. This becomes important and necessary when matching the outlines of the reference object and the model object. If the arms
of a person in the image are raised up, the model object should have its arms raised to the same position.

3.3 Aligning the Model Objects to the Reference Objects

Once the edges of the image have been detected and connected and the model objects have been created, each model object must be aligned to each reference object in the image. In this way, three-dimensional positions of the objects in the scene can be discovered. This section will describe all the details of that process.

3.3.1 Modeling the Camera

During filming the position and orientation of the camera must be recorded for each frame of film shot. Without this knowledge, the size of the objects in the image can not be used to calculate their distances from the camera. If both the object and the camera’s position are unknown, the equations for distance calculation will have infinitely many solutions and can not be solved. For single frame photography, recording the camera information is an additional inconvenience, but not too debilitating. In motion picture photography, if the camera is going to move during the shot, a computerized motion control device must be used to record the position of the camera at each frame. A reference point should be picked in the scene and at each frame, the three-dimensional location of the focal point of the camera and the orientation of the camera (pitch, roll and yaw) should be recorded. The camera lens parameters (zoom, f-stop) should also be recorded.

Knowing the location of the focal point of the camera is not enough information.
The lens parameters and setting must be known and modeled too. The properties of the camera lens are modeled by shooting some test images with the camera and some reference objects placed at a fixed distance from the camera. The focal point of the camera is known. The size of the objects are known. Three-dimensional computer models of the objects in the scene are created and positioned at the exact same place in the scene as the reference objects. The viewpoint of the renderer is positioned at the same place as the focal point of the camera. The scene is rendered and the view angle and other camera parameters are adjusted until the reference photo and the rendered image match exactly.

In his Master's thesis, Steve Anderson [1] developed the mathematics to fully model a lens based on its focal length. He showed that an image rendered using his model could be created so that an exact match between the rendered image and a test photograph could be created. His model could be used here instead of the ad hoc solution presented above.

### 3.3.2 Placement of the Model Objects

Now, with the reference photograph, the edge detected image of that photograph, the three-dimensional model objects of the reference objects in the scene and the parameters of the camera, it is possible to create a complete three-dimensional model of the entire original scene. Unfortunately, it can not be done completely automatically. It has long been a problem in the area of artificial intelligence and image processing to have a computer understand an image as well as a human can. Unfortunately, only small advances have been made in this area. It is now possible for a computer to
recognize a shape from a limited subset of shapes that it knows about. A computer, attached to a video camera monitoring a conveyor belt with tools moving by on it, can recognize the various tools that pass by. But only if the computer has been instructed as to the shape of the tool and if the tool is oriented correctly can this be done reliably. In this thesis, no limitation on the type of things one could photograph was imposed. As long as one knew the shape and dimensions of the objects in the scene, the computer could calculate the distance. Without limitations on the recognizable object set, the problem of having a computer comprehend an image is beyond the capacity of current technology. However, human beings are wonderful at image recognition. They are relatively bad at distance calculations. So, relying on the strengths of the computer to calculate exact distances and of humans to recognize objects, this problem can be solved. An operator is used to give the program a “hint” as to where the reference objects are in the scene.

Using an interactive object positioning program, the operator loads the reference image and the model objects into the program. Then, for each object in the scene, the operator moves each model object so that it’s “close” to each reference object that it is supposed to represent. To do this, the operator is given a display where the reference image is displayed and each model object is displayed on top of the reference image in wireframe. The operator may only rotate and translate the object. As he or she does this, the object is shown as it would appear in perspective, at that orientation and that position in the scene. Scaling of the object is not allowed since the size of the model object is used to determine the distance the reference object
is from the camera. If the object is scaled to make it fit better, it will throw the distance calculations off. If the operator wants to make the object smaller, he or she must move the object farther away from the camera.

The rotation of the object is the most important part of this operation. As will be explained in section 3.3.4, calculating the rotation of an object — even when the object location is known in the image — is very difficult. Precise points, say the corners of a cube, must be known and matched with the corners of the model object. However, with edge detection it's hard to tell if there is a straight line in the image, much less a corner. With more complex objects it is well nigh impossible for the computer to match up a specific point on the model object with the same point on the reference object. So, this algorithm is very dependent on the operator to rotate the object so that is oriented the same way as the reference object. Naturally, there will be errors in the rotation that the operator performs, but these can be tolerated and, in the case of motion picture photography, actually reduced over time as more information about the scene is collected.

This operator intervention is inconvenient but necessary. It is made easier by a couple of features. If multiple images are shot of the same scene, whether it is traditional photography or motion picture photography, the scene will only have to be modeled once. As the camera moves around the scene, any non-moving objects in the scene should not change their position. In fact, if the camera does move from one point to another while filming approximately the same scene, the positions of the non-moving objects in the scene can be calculated to a greater accuracy. The
different perspectives on the scene give the program more information in calculating the positions and orientations of the objects. Also, in motion picture photography, even moving objects don’t change their position by very much from frame to frame, unless they are moving very fast. The operator would be responsible for adjusting the orientation of the model object for each frame, but the program should be able to translate the object so that it moves along with the reference object.

Figure 18: An example of a scene with the model objects placed close to the reference objects. The model objects are the objects outlined in heavy black.

In Figure 18 the scene with the mailbox and the street light is shown with the model objects placed “close” to the reference objects. What is close enough? The answer to that question depends on how complicated your image is. If the image is rather simple, as Figure 18 is, the model objects can be positioned fairly sloppily
around the reference objects that they are to model. However, if the image being modeled is a complicated one, with many edges and overlapping objects, the model objects must be placed nearly on top of the reference objects. Remember, there is really no way for the computer to tell which edge belongs to which reference object in the image. So, if the reference image is dense with edges, the model objects must be placed close to the edges with which they are to align. The alignment process is a iterative one, and if the model object is aligning itself with improper edges, the alignment process can be stopped and the model object can be repositioned.

### 3.3.3 Computer Assisted Positioning

Returning to the original sample image, Figure 19 shows the status of the image and what is known after the operator has positioned a model object on top and near a reference object (in this case the cube). The only information that is available about the reference object is its edges, and even that information is not well organized. In a typical image there may be edges caused by the pattern on an object, extraneous edges generated by the edge connection program and, of course, edges caused by other objects in the scene. Since the model object is typically much simpler than the reference object, relying on exact matching between the details of each of the objects is unrealistic. The only information that is present, detectable and usable in both the model object and the reference object is the exterior outline or silhouette edges of the objects. This section will describe how the outline of the model object is aligned with the outline of the reference object.
Figure 19: A three-dimensional model (the dotted lines) of the cube has been placed near the edges of the reference cube.
Problems in aligning the objects

The problem, as it stands now, is to somehow align the reference image, a photograph of, most likely, real-world, fuzzy, bumpy objects, and the model objects, approximations of the reference objects that don't have the same amount of detail and may not even be aligned perfectly. Since the interior details of the reference objects (patterns, textures and the like) are not modeled by the model objects, the information contained there cannot be used to align the two. In fact, as mentioned in the previous paragraph, the only correspondence between the reference object and the model object is the outlines or silhouettes of the objects. But even these are different from one another. What information is contained in both the reference object and the model object and how can that be used to align the objects? These questions will be discussed in this section.

One of the problems that must be dealt with when aligning the objects is that the model object is a polygonal representation of a three-dimensional thing. This means that the object will have straight edges between points. If the object is to have curved surfaces in it, the surfaces will be approximated with many polygons. Because of this, the silhouette edges of the model object will be composed of mathematically straight lines.

Worse though, is the fact that, because of object imperfections, photography errors, digitization artifacts and edge detection inaccuracies, a section of the silhouette of the reference object that is to represent a straight line will not be perfectly straight. Naturally, when anything is digitized, aliasing will occur and distort the edge. An
example is shown in Figure 14.

If a human being was presented with the task of aligning two images, one of the first methods he or she would use would be to pick out various reference points on each object and try to match these points. When done, some fine tuning would be done on the rest of the parts of the objects to get an exact match.

Reference points are nearly impossible for the computer program to locate on the images given the information it has available. Perhaps the easiest reference points on an image of an average object would be the objects corners. Finding the corners of the model object is trivial. The points of the polygons that make up the silhouette edge of the object would be the corners of the object. However, finding the matching corners on the reference image is not nearly as easy. As stated before, straight lines are not always straight. Given this, it is very difficult to detect an actual corner of the object as opposed to a corner generated by a digitizing artifact.

One solution to this is to have the human operator identify the reference points on the reference object. This adds a considerable amount of work for each object in the scene, but makes some of the math easier. For this thesis, it was assumed that this identification of points would be used as only a last resort. In Section 3.3.4 the usefulness of such a procedure becomes evident when rotating the object is done by the computer.

Another problem inherent in reference point location is that while the model object might have corners, the reference object may not. Consider the case of a photograph of a cylinder (See Figure 20). The reference cylinder would be modeled
with a polygonal representation of a cylinder. When analyzing the model object, many points along the rim of the cylinder would be flagged as "corners". Obviously, there would be no corresponding corners on the model object. In fact, if a human operator was asked to identify a "corner" on the reference object to match a corner found on the model object, he or she would have a difficult time doing so!

**Sending out probes from each model object edge**

The above section shows that the only easily obtainable information about the positions of the reference objects is the silhouette edges of the objects. By using the silhouette edges of the model objects as a starting point, it is possible to find the reference silhouette edges from the extraneous interior and spurious edges present in the image. This section explains the procedures that must be followed to do this.
Figure 21: The silhouette edges of the reference object are calculated.

The silhouette edge of the model object must first be found. For a convex object this is trivial. First, the normals are calculated for each polygon. These are then compared to the eye vector. Then, for each edge between two polygons, if the dot product of the normals of each polygon with the eye vector have the same sign, the edge is discarded. Only edges with polygons that have different signed dot products are kept. For a convex object, the complete silhouette of the object is composed of this set of edges. See Figure 21.

Model objects may be built by hierarchically linking simple convex objects, thus making more complex, concave objects. The silhouette of a hierarchical object is simple to determine. The silhouettes for each of the sub-objects are calculated. Then,
for each convex object, clip the silhouette of it with the rest of the objects. The resultant edges of all the objects can be sorted and linked and this will result in a complete silhouette of a more complex object.

If the reference object is modeled by a true concave object — not one composed of convex objects — the silhouette can still be determined. It is slightly more complicated though. Typically, the edges that are shared by back and front facing polygons are calculated and then the interior edge segments of these edge are eliminated. It can also be done by simply rendering the object and remembering the edges that make up the boundaries of the object. This is slightly slower.

Since perspective transforms and clipping have been done on the model object, it maps onto the digitized image in the same manner as an actual object of the model object’s size and three-dimensional position. The silhouette of the model object provides a starting point for the search for the corresponding edges in the reference object.

Before the search for the edges is explained, the constraints should be examined. The model object is placed “close” to the reference object. That means that the model object is near enough to the reference object so that other edges in the image will not be mistaken for the edges that are being searched for. In a controlled environment, it would be reasonable to expect that if a reference edge is found and it isn’t the correct edge, it could be thrown out because the slope of the edge doesn’t fall within some error limit of the slope of the model edge that the search was started from. However, in the case of most images the slope of a digitized line is very difficult to obtain. Once
again, refer to Figure 14. That figure shows typical edges encountered after an image has been digitized. The slope of the edge can be approximated, but only with a large degree of error. Also, in typical images, straight lines are the exception, not the rule. Bumpy, curved silhouettes are much more typical of the types of things in everyday life. So, using a heuristic to eliminate edges from consideration may be possible in some cases, but in general it cannot be used.

Given that there might be a considerable amount of noise in the image in terms of extraneous edges, a method was developed to search for the reference edges that corresponded to the model edges. The slope of the model edge is used to calculate the perpendicular of the edge. It is considered reasonable to assume that the model edge and the reference edge have approximately the same slope. Starting from the edge, in the direction of the perpendicular, the pixels of the reference image are searched. The first edge pixel (and the edge that it's attached to) that is found will be considered the matching reference edge to the model edge. This process is called an edge probe.

Since there is no constraint on the human operator to place the model object "inside" the reference object, two probes must be sent. One from each side of the edge. The first one to encounter an edge pixel will take priority.

The starting point for the edge probes is not obvious. Initially, the endpoints of the edges look attractive in that their three-dimensional position is known and ultimately they will be used in the calculation of how far to move the model object to fit the reference point. Unfortunately, if a probe is started from the one or both of the endpoints of a silhouette edge it is most likely going to miss finding its appropriate
Figure 22: Examples of how the endpoints of a silhouette edge can extend beyond the edge of the matching reference object.
reference edge. It is assumed that the human operator who placed the model object did a good job in that the model and reference silhouette edges are close to one another. However, it is also assumed that some errors will occur. (Actually, this is guaranteed since the model object and the reference object will rarely be an exact match.) The most common errors to make when positioning the model object are either to place the model object closer to the camera than the reference object, having it appear bigger than the reference object or to place the model object slightly offset from the reference object. In both of these cases at least one and maybe both of the endpoints of certain silhouette edges will extend beyond the edges of the corresponding reference edge. (See Figure 22.) Any perpendicular edge probes starting at the endpoints of such edges would completely miss the intended reference edge.

To alleviate this problem, the edge probes for each model silhouette edge are started in the middle of the edge. By doing this, small errors in placement of the model object will not affect the chances that the edge probe will encounter the appropriate reference edge.

Two perpendiculars are sent out from the middle of each model object silhouette edge. Eventually, one of these edge probes will encounter a pixel that contains an edge color or the boundary of the image. If both probes encounter the boundary of the image, the edge is not included in the calculation of the exact position of the object. If one of the probes does hit an edge pixel, that pixel and the edge that it's attached to are considered the reference edge match to the probing model edge. It may be that both edge probes sent in opposite directions from the model edge will
find edges. In that case the closest reference edge is chosen to be the matching edge. Figure 23 shows the results of this process.

Figure 23: Probes are sent out from the midpoints of the edges of the reference object. The labeled probe has identified the wrong reference edge.

Errors will occur. Because of the relative coarseness of the model object and the inaccuracies of the human operator, the model object will most likely be placed slightly off. And even with the edge probes extending from the middle of the edges, they still might miss the correct edge completely or, possibly, erroneously encounter another edge before the correct reference edge. As long as the majority of the edge probes find the correct reference edges, the fitting algorithm will work correctly. However, it should be mentioned at this point that this entire algorithm is an iterative one. The first positioning of the model object by the computer will not be the most ac-
accurate. It should get the model object much closer to the reference object than the human operator placed it. Then, if the entire probe and adjust procedure is repeated, the probability for error decreases. If the procedure is repeated multiple times, the initial errors of probing can be eliminated.

Another solution would be to send out more than one pair of probes from each model edge. Instead of searching from just the center, a search could be started from each pixel that the model edge intersects. The reference edge that is selected by the majority of edge probes would then be chosen as the matching edge. This has problems in that the same things can happen as did with the edge probe starting at the endpoint of the edge. It is possible that a majority of the edge probes could identify the wrong edge. Also, it is difficult to tell if one reference pixel is connected to another reference pixel along a single edge. With the edge connection algorithm, most pixels in the image will have been connected to one another in some fashion.

Calculating the distance to move each vertex of the model object

The probes sent from each edge show how far and in what direction a single model object edge must be moved so that it aligns with an edge on the reference object. However, a single probe vector doesn’t result in enough information to get the endpoints of a model object edge to match with the reference object corners. To align both corners of a model object edge, one must use the probe vectors of the edges adjacent to the model object edge in question. For each vertex in the list of silhouette edges of the model object, the probe vectors associated with the two edges that
contain that vertex are used to calculate how far the vertex must move to be aligned with a corner on the reference object.

Figure 24: Using the probes, the two-dimensional adjustments for the corner points of the model object are calculated.

Figure 24 shows an example of the vectors that need to be calculated. Since the three-dimensional position of the vertices of the silhouette edge of the model object are known, knowing how far they must be moved on the screen is essential for calculating the amount the entire object is to be moved in three space. Using a process described below, the probe vectors from adjacent edges are combined to produce the vertex vectors. Notice that the erroneous probe vector at the top left corner of the cube will cause two vertex adjustment vectors to be in error.

Figure 25 will be used to explain the calculation of the vertex vectors more fully.
In each part of this figure, a section of the reference object's outline is shown as the dotted line. The solid lines are two model object edges after the initial placement of the object by the human operator.

![Figure 25: Calculating the vertex adjustment vectors is a three step process: (A) probing, (B) calculating the adjustments to the edges, and (C) intersecting the vectors to find the corner point.]

In Part A of Figure 25 the perpendicular probe is sent from each model object silhouette edge. It is sent from the middle of the edge so that it has the greatest chance of hitting the appropriate reference object edge. However, a single probe does not give enough information to fully align the model object edge to the reference object edge. As one can see, if the edge $\vec{L}_1$ is moved in the direction and distance specified by vector $\vec{P}_1$, the edge will be parallel and on top of the reference object edge, but the corners will not match up.
By using silhouette edges adjacent to a vertex, the distance and the direction the vertex must be moved can be calculated. In Part B of Figure 25, the probe vectors $\vec{P}_1$ and $\vec{P}_2$ are added to the points $V_1$, $V_2$ and $V_3$. This results in two new edges, $\vec{L}_1'$ and $\vec{L}_2'$ shown in Part B of the figure. These new edges should be on top of the reference model edges, with their corners slightly off. To calculate the position of the corner, the intersection between the two new vectors must be found. That point is represented by the letter $S$ in Part C of Figure 25.

The intersection, $S$, of the two two-dimensional lines, $\vec{L}_1'$ and $\vec{L}_2'$ is trivial. When $S$ is found, the vector $\vec{A} = S - V_2$ can be calculated. This is the distance in screen space that the vertex $V_2$ should be moved.

This procedure continues for all vertices in the silhouette edge of the model object. Eventually, the distance each one should be moved will have been calculated.

**Calculation of the three-dimensional position of a single point**

At this point the position and adjustment vector of each vertex on the silhouette edge of the model object are known. The task now is to calculate the distance the vertex must be moved in three space so that it will be rendered at the correct position on the image. The main problem with this is that there are an infinite number of $[x,y,z]$ triplets that render to a specific pixel on an image. ( See Figure 26. ) How should the vertex be moved?

Before we calculate the overall adjustment of the model object to fit the reference object, the simpler problem of moving a single vertex will be examined. The current
Figure 26: An infinite number of points render to a specific pixel.
position of the vertex in world space, the pixel that the vertex currently renders to and the pixel the vertex should render to are known. This section derives the equation that calculates the position of the point that is closest to the original vertex and still renders to the desired pixel.

We are given a point on the object model \( P_e = [x_e, y_e, z_e, 1] \), the eye space position of a vertex and \( S = [x_s, y_s] \), the screen coordinates of a pixel on the image matched to \( P_e \). We want to find \( P'_e = [x'_e, y'_e, z'_e, 1] \), the eye space position of a point to which \( P_e \) will be moved that is closest to \( P_e \) and renders at point \( S \).

The first step in calculating the point \( P'_e \) is to use the standard perspective transformation matrix and calculate the coordinates of the pixel that an eye space point will render.

\[
x_s = \frac{x'_e}{z'_e \tan \theta} \quad y_s = \frac{y'_e}{z'_e \tan \theta}
\]  

(3.1)

where \( \alpha \) is equal to the aspect ratio of the screen and \( \theta \) is equal to the viewing angle.

Solving Equation 3.1 for \( x'_e \) and \( y'_e \) gives

\[
x'_e = x_s \tan \theta z'_e \quad y'_e = y_s \frac{\tan \theta}{\alpha} z'_e
\]  

(3.2)

The distance between the original point \( P_e \) and the new point \( P'_e \) is given by the standard distance equation.

\[
D = \sqrt{(x_e - x'_e)^2 + (y_e - y'_e)^2 + (z_e - z'_e)^2}
\]  

(3.3)

Using Equations 3.2 and 3.3

\[
D = \sqrt{(x_e - x_s \tan \theta z'_e)^2 + (y_e - y_s \frac{\tan \theta}{\alpha} z'_e)^2 + (z_e - z'_e)^2}
\]  

(3.4)
Since the object of this derivation is to find the \( z'_e \) that results in the minimum distance, the square root is not needed. Taking the derivative with respect to \( z'_e \) of the square of Equation 3.4 gives

\[
\frac{dD^2}{d\dot{z}_e} = -x_ex_s \tan \theta + x_s^2 \tan^2 \theta \dot{z}'_e - y_ey_s \frac{\tan \theta}{\alpha} + y_s^2 \left( \frac{\tan \theta}{\alpha} \right)^2 \dot{z}'_e - \dot{z}_e + \ddot{z}_e
\] (3.5)

Setting Equation 3.5 to zero and solving for \( \dot{z}'_e \) results in this final equation,

\[
\dot{z}'_e = \frac{x_ex_s \tan \theta + y_ey_s \frac{\tan \theta}{\alpha} + \ddot{z}_e}{x_s^2 \tan^2 \theta + y_s^2 \left( \frac{\tan \theta}{\alpha} \right)^2 + 1}
\] (3.6)

With Equations 3.6 and 3.2 we now have a complete description of the eye position of the point \( P'_e \).

**Errors in the vertex adjustment vectors due to rotation of model object**

It is assumed that the human operator who placed the model object near the reference object did a good job. In particular, it is assumed that the model object and reference object are oriented the same way in three space. If this is the case, the model object edges and the reference object edges would be parallel to one another. With the edges parallel, the method of calculating the vertex adjustment vector will always work. However, it is quite possible that either the human operator will make an error in the positioning of the model object or the model object will be a simplification of the reference object and the edges will not always be parallel. This being the case, the amount of error produced by the algorithm presented in the previous sections, based on the angle the edges were misrotated, can be calculated.

Figure 27 is a rather complicated figure which explains the geometry involved in calculating the error in aligning a misrotated model object. In the figure, there
The reference edges are $Re_1$ and $Re_2$

The model object edges are $Me_1$ and $Me_2$

The calculated corner point is $S$

The actual corner point is $I$

Figure 27: If the model object has been misaligned with the reference object by rotation, the vertex adjustment vector will be miscalculated.
are two heavy solid lines, labeled $Re_1$ and $Re_2$, that represent the reference object edges. For ease of explanation, the edges are at right angles to one another. The results of this section can be extended to deal with edges that intersect at an angle other than 90 degrees. The heavy dashed lines, $Me_1$, $Me_2$, $P_1$ and $P_2$ represent the two misaligned model object edges and the probe vectors sent from their midpoints, respectively.

The error will be defined as the distance from the actual position of the intersection of $Re_1$ and $Re_2$, labeled $I$, and the calculated position of intersection, point $S$. The amount of misrotation is given by the value of $\phi$ and will be the angle defined by the intersection of a reference object edge with the corresponding model object edge.

The easiest way to calculate the amount of error is to use the model object edges as axes of a coordinate system. The task is to find the location of $S$ and $I$, from which the distance between them can be found. Since the model object edges are at a right angle to each other, they provide a nice coordinate system. The location of point $S = \begin{bmatrix} P_2 \\ P_1 \end{bmatrix}$ in that system is given by the algorithm described on page 61.

The location of the actual intersection of the two reference edges, $I$, is a little more complicated. The following is known:

- $\phi$ is the angle defined by a reference object edge and a model object edge.
- $L_1$ and $L_2$ are distances to the midpoints of the model object edges.
- $P_1$ and $P_2$ are the lengths of the probes sent from each of the model objects.

Using this information it is possible to develop equations for the lines $Re_1$ and
$Re_2$ in the coordinate space defined by the model object edges.

For $Re_1$ the line equation is:

$$y = P_1 + L_1 \tan \phi - x \tan \phi$$  \hspace{1cm} (3.7)

For $Re_2$ the line equation is:

$$y = L_2 - \frac{P_2}{\tan \phi} + \frac{x}{\tan \phi}$$  \hspace{1cm} (3.8)

The intersection of $Re_1$ and $Re_2$ result in the position of $I$. Equations 3.7 and 3.7 give us two equations and two unknowns. Solving for $x$ and $y$ results in:

$$x = \frac{P_1 \tan \phi + L_1 \tan^2 \phi - L_2 \tan \phi + P_2}{1 + \tan^2 \phi}$$

$$y = \frac{-P_2 \tan \phi + L_2 \tan^2 \phi + L_1 \tan \phi + P_1}{1 + \tan^2 \phi}$$  \hspace{1cm} (3.9)

As $\phi$ approaches zero, the values of $x$ and $y$ in Equation 3.9 approach $P_2$ and $P_1$, as one would expect. More importantly, to find the minimum error for a given misrotation of the object, $P_1$ and $P_2$ are set to zero. This means that the midpoints of the model object edges are right on top of the reference object edges and hence, the alignment of the objects are as good as it can possibly get. This results in the best possible position of the calculated corner point for a given misrotation:

$$x = \frac{L_1 \tan^2 \phi - L_2 \tan \phi}{1 + \tan^2 \phi}$$

$$y = \frac{L_2 \tan^2 \phi + L_1 \tan \phi}{1 + \tan^2 \phi}$$  \hspace{1cm} (3.10)

The amount of error generated by the misrotation of a model object is the distance between the actual corner of the reference object, $I$ and the calculated corner, $S$. 
Using the values in either Equation 3.9 or Equation 3.10, the error becomes:

\[
\text{Error} = \sqrt{(P_2 - x)^2 + (P_1 - y)^2}
\]  

Substituting Equation 3.10 in Equation 3.11 and using the fact that \( P_1 \) and \( P_2 \) are zero, renders an equation for the minimum amount of error distance for a given misrotation.

\[
\text{Error}_{\text{min}} = \sqrt{(L_1^2 + L_2^2) \sin \phi}
\]  

From this equation, one can see that the longer the model object edges are, the greater the error factor. Also, since the equation is based on a \( \sin \) function of \( \theta \), the error will start out small and remain linear for a while, as long as \( \phi \) remains small.

Equation 3.12 calculates the smallest amount of error on the screen one can expect with a certain misrotation of the model object. However, that also has an effect on the three space location that is eventually computed for the model object. In the previous section, the method of calculating the three-dimensional position of single vertex on a model object is discussed. We will use these concepts to derive the three space distance the screen space error will create.

After the iterative application of the object alignment procedures described in this chapter, the model object will be translated so that its silhouette edges are as close as possible to those of the reference object. However, as shown above, if the model object is misrotated, getting the model object closely positioned to the reference object does not remove all error. The distance that a calculated corner pixel will be off from the actual corner pixel has already been calculated in this section. Now, for a given \( z_e \) value, the pixel distance error can be interpolated to three space.
For a given intersection point of two reference edges, \( I = \begin{bmatrix} x_I & y_I \end{bmatrix} \), the lengths of the two model edges, \( 2L_1 \) and \( 2L_2 \) and the angle of misrotation, \( \phi \), the minimum error distance is \( \text{Error}_{\text{min}} = \epsilon \). (Given in Equation 3.12) Using this value, a possible location of the calculated intersection pixel, \( S \), can be calculated:

\[
\begin{bmatrix} x_S & y_S \end{bmatrix} = \begin{bmatrix} x_I + \epsilon & y_I \end{bmatrix}
\]

(3.13)

Using Equation 3.1, for a given \( z_e \) value, the eyespace location of a point that renders to pixel \( I \) on the screen is:

\[
x_e = x_I z_e \tan \theta \\
y_e = y_I z_e \frac{\tan \theta}{\alpha} \\
z_e = z_e
\]

(3.14)

At the end of the previous section, the \( z \) location of a three space point, \( z_e' \), that is closest to a reference point, \( z_e \) and renders to a point \( \begin{bmatrix} x_S & y_S \end{bmatrix} \) is calculated. (Equation 3.6) Using this equation, the \( z \) location of the point that renders to the error position on the screen is calculated by substituting the values from Equation 3.14 into Equation 3.6. (\( \theta \) is the viewing angle and \( \alpha \) is the aspect ratio of the screen.)

\[
z_e' = z_e \left( 1 - \frac{\epsilon^2 + x_I \epsilon}{(x_I + \epsilon)^2 + \frac{y_I^2}{\alpha^2} + \frac{1}{\tan^2 \theta}} \right)
\]

(3.15)

Using this equation, the values of \( x_e' \) and \( y_e' \) can be calculated. (Equation 3.2) The distance between the point \( \begin{bmatrix} x_e & y_e & z_e \end{bmatrix} \) and \( \begin{bmatrix} x_e' & y_e' & z_e' \end{bmatrix} \) is given by:

\[
D = \sqrt{(x_e - x_e')^2 + (y_e - y_e')^2 + (z_e - z_e')^2}
\]

(3.16)
Substituting:

\[
\begin{align*}
    x_e - x'_e &= z_e \epsilon \tan \theta \left( \frac{(x_I + \epsilon)^2}{(x_I + \epsilon)^2 + \frac{y_I^2}{\alpha^2} + \frac{1}{\tan^2 \theta}} - 1 \right) \\
    y_e - y'_e &= -z_e y_I \frac{\tan \theta}{\alpha} \frac{\epsilon^2 + x_I \epsilon}{(x_I + \epsilon)^2 + \frac{y_I^2}{\alpha^2} + \frac{1}{\tan^2 \theta}} \\
    z_e - z'_e &= \frac{z_e \epsilon^2 + x_I \epsilon}{(x_I + \epsilon)^2 + \frac{y_I^2}{\alpha^2} + \frac{1}{\tan^2 \theta}}
\end{align*}
\]  

(3.17)

Now, to simplify the equation, assume:

\[
K = \frac{1}{(x_I + \epsilon)^2 + \frac{y_I^2}{\alpha^2} + \frac{1}{\tan^2 \theta}}
\]  

(3.18)

Finally, the eye space error distance that an error in the positioning of the model object creates, based on values of \(z_e, x_I, y_I\) and \(\epsilon\) can be calculated:

\[
D_{\text{min}} = z_e \sqrt{\epsilon^2 \tan^2 \theta \left( (x_I + \epsilon)^2 K - 1 \right)^2 + \left( y_I \frac{\tan^2 \theta}{\alpha^2} + 1 \right) \left( \epsilon^2 + x_I \epsilon \right)^2 K^2} 
\]  

(3.19)

The value of the error distance is based on not only the error term \(\epsilon\) but also the pixel position \([x_I \ y_I]\). This means that miscalculation of the object position increases as the object strays from the center of the image.

In summary, a few things can be seen from this equation. First, as the object gets farther away from the eye the magnitude of the error increases. Also, to a lesser extent, the farther the objects is removed from the center of view, the larger the error.

The size of the two-dimensional error term is the main determiner of the magnitude of the three-dimensional error. Luckily, because it is based on the \sin\ function, it does not increase quickly for small amounts of misrotation.
Calculation of model object position for two-dimensional alignment

In the discussion starting on page 64, the three-dimensional position of a single point is calculated. The problem becomes more complex when dealing with the network of vertices that comprise the silhouette edge of the object. It isn't possible to calculate the individual vertex position adjustments and apply them to the vertices. That would result in the deformation of the model object and it assumes that all the vertex adjustment vectors were correct. It also makes an incorrect assumption that for all vertices on the silhouette edge, the minimum amount of translation is the correct amount of translation. This is not the case. Consider an operator who aligned the model object of a box very closely to the left edge of the reference object box. The right edge of the model object box was a considerable distance in error. If each vertex was moved without regard to the other vertices of the box, the left side vertices would be moved only slightly ( they are already close to where they should be ) but the right side of the model object would be translated considerably. A gross deformation of a model object like this is considered unacceptable. The model object was created to approximate the reference object in three dimensions. If the model object is deformed by a large amount, new lighting effects added to the object would appear un-natural and visually disturbing. Additionally, since the three-dimensional position of the model object would not be calculated correctly and would be subject to large changes over subsequent frames in a sequence of images, the lighting effects would jump around on the surface of the object.

It is desired to find a \[ \begin{bmatrix} \Delta x & \Delta y & \Delta z & 1 \end{bmatrix} \] that can be uniformly applied to all
vertices so that the vertices are placed on top of the corners of the reference object. This section derives the equations necessary for the calculation of this adjustment to the position of the model object. In this section, rotation of the model object by the computer will not be considered an option. In Section 3.3.4 the difficulties in automatically rotationally aligning the model object to the reference object is discussed. This section only considers the possibility of translation of the model object.

The following things are known at this point:

- The position of the model object in eye space, \( P_o = \begin{bmatrix} x_o & y_o & z_o & 1 \end{bmatrix} \)

- The position, in eye space, of each vertex on the silhouette of the model object, \( P_i = \begin{bmatrix} x_i & y_i & z_i & 1 \end{bmatrix} \) for all \( i = 1 \ldots n \). The non-silhouette vertices are not interesting at this point.

- For each \( P_i \) there exists an \( S_i = \begin{bmatrix} x_{si} & y_{si} \end{bmatrix} \) which is the screen space coordinates of the point at which we want \( P_i \) to be displayed.

If we move the object in eye space by an amount \( \Delta = \begin{bmatrix} \Delta x & \Delta y & \Delta z & 1 \end{bmatrix} \), each point \( P_i \) will be displayed at a new location in screen space, \( S'_i = \begin{bmatrix} x'_{si} & y'_{si} \end{bmatrix} \). Using Equation 3.1 and the amount moved \( \Delta \), we get

\[
x'_{si} = \frac{x_i + \Delta x}{(x_i + \Delta z) \tan \theta} \quad \quad \quad y'_{si} = \frac{(y_i + \Delta y) \alpha}{(x_i + \Delta z) \tan \theta}
\] (3.20)

The distance between to the two points on the image, \( S_i \) and \( S'_i \) is given by

\[
D_i = \sqrt{(x_{si} - x'_{si})^2 + (y_{si} - y'_{si})^2}
\] (3.21)
The previous two equations yield

\[
D_i^2 = \left( x_i - \frac{x_i + \Delta x}{(z_i + \Delta z) \tan \theta} \right)^2 + \left( y_i - \frac{(y_i + \Delta y) \alpha}{(z_i + \Delta z) \tan \theta} \right)^2
\]  

(3.22)

So, for each vertex on the silhouette edge of an object we have an equation,

\[
f_i(\Delta x, \Delta y, \Delta z) = D_i^2 \text{ for all } i = 1 \ldots n
\]  

(3.23)

We want to find the values of \( \Delta x \), \( \Delta y \) and \( \Delta z \) such that all \( f_i(\Delta x, \Delta y, \Delta z) \) equal zero. Unfortunately, in the general case, such a solution will not exist. The normal probes sent from each silhouette edge are not always accurate. They may strike an edge of another object or a errant pixel before they hit the edge they were supposed to hit. Other errors dependent on the edge detection algorithm might occur as well. What this means is that the system of equations, \( f_i(\Delta x, \Delta y, \Delta z) = 0 \) for all \( i = 1 \ldots n \) probably won't have a solution. ( See Figure 28. )

Understanding this limitation, we want to find the values of \( \Delta x \), \( \Delta y \) and \( \Delta z \) such that the model object would match the reference object position the best. But, what is the best solution for this problem? Should the task be to find the smallest total of all the \( \Delta s \)? How do we search the space of possible values for \( \Delta x \), \( \Delta y \) and \( \Delta z \)?

Because of the rarity of an actual solution to the set of equations, an algorithmic method of finding a \( \Delta x \), \( \Delta y \) and \( \Delta z \) that causes the model object to render "closest" to the adjusted vertex points was developed. ( Because of the occurrence of erroneous vertex adjustment vectors, it may be the case that one or two of the calculated vertex positions are incorrect and all the rest are correct. In this case, if the position of the model object is calculated so that it minimizes the distance of all the silhouette
Figure 28: Because of errors in the adjustment vectors, it might be impossible to find a position of the model object (the dotted lines) so that the silhouette corners exactly match the positions they are supposed to map to (the black with white dots) while preserving the shape of the model object.
vertices from where they are supposed to be, that solution would be less correct than one that disregarded the couple of erroneous vertices altogether. So, "closest" means a point where the majority of vertices are in their correct positions or, failing that, a point where the distance between the vertices and their calculated positions is minimized. A search through the possible values of $\Delta x$, $\Delta y$ and $\Delta z$ could find the proper values, but the search space is terribly large. So, a method to reduce the search space was also developed.

![Diagram showing possible positions for the model object with fixed vertex rendering to specified point.](image)

Figure 29: By fixing the point where a single vertex must render, the search space for the best fit is reduced.

Initially, one of the vertices of the silhouette edge of the model object is chosen randomly ($P_p = \begin{bmatrix} x_i & y_i & z_i & 1 \end{bmatrix}$). This point should render to the point $S_p = \begin{bmatrix} x_s \ y_s \ z_s \ 1 \end{bmatrix}$. Using Equation 3.20, knowing that $x'_{si} = x_s$ and $y'_{si} = y_s$, an equation
that calculates the $\Delta x$ and $\Delta y$ based on a value for $\Delta z$ can be derived. (Equation 3.24.)

$$
\Delta x = (z_p + \Delta z) x_p \tan \theta - x_p \\
\Delta y = (z_p + \Delta z) y_p \frac{\tan \theta}{\alpha} - y_p
$$

(3.24)

This can be thought of as drawing a line through the eye point and the pixel $S_p = \left[ x_{sp}, y_{sp} \right]$. If a point falls anywhere on that line, it will render to $S_p$. Equation 3.24 calculates the $\left[ \Delta x \quad \Delta y \quad \Delta z \quad 1 \right]$ that must be added to $P_p$ to get it to a point on the line, for a given $\Delta z$ value.

Now, for a given $\Delta x$, $\Delta y$ and $\Delta z$, Equation 3.20 can also be used to calculate the position that any other silhouette vertex, $P_i = \left[ x_i \quad y_i \quad z_i \quad 1 \right]$, on the model object will render to on the screen.

However, that point should, according to the probe vectors, render to a pixel $\left[ x_{r_i}, y_{r_i} \right]$. The distance between the two pixels is given by:

$$
D_i^2 = (x_{r_i} - x_{s_i})^2 + (y_{r_i} - y_{s_i})^2
$$

(3.25)

Now, substituting Equation 3.20 into the previous equation, one gets:

$$
D_i^2 = \left( x_{r_i} - \frac{x_i + \Delta x}{(z_i + \Delta z) \tan \theta} \right)^2 + \left( y_{r_i} - \frac{(y_i + \Delta y) \alpha}{(z_i + \Delta z) \tan \theta} \right)^2
$$

(3.26)

Equation 3.26 can be combined with Equation 3.24 to create the next equation.

$$
D_i^2 = \left( x_{r_i} - \frac{x_i + (z_p + \Delta z) x_p \tan \theta - x_p}{(z_i + \Delta z) \tan \theta} \right)^2 + \\
\left( y_{r_i} - \frac{(y_i + (z_p + \Delta z) y_p \frac{\tan \theta}{\alpha} - y_p) \alpha}{(z_i + \Delta z) \tan \theta} \right)^2
$$

(3.27)
By letting $\beta = \frac{1}{\tan \theta}$ and $\gamma = \frac{\alpha}{\tan \theta}$ the previous equation (Equation 3.27) can be simplified to:

$$D_i^2 = \left( x_{ri} - \frac{x_i\beta + (z_p + \Delta z) x_{sp} - x_p\beta}{z_i + \Delta z} \right)^2 + \left( y_{ri} - \frac{y_i\gamma + (z_p + \Delta z) y_{sp} - y_p\gamma}{z_i + \Delta z} \right)^2 \quad (3.28)$$

This equation now relates the value of a $[\Delta x \Delta y \Delta z 1]$ triple that keeps a certain vertex somewhere on the eye point-pixel line with the distance any other vertex on the model is from its desired rendering point. All this is based on the value of $\Delta z$. What is required now is the value of $\Delta z$ that gets the vertex $P_i$ closest to its rendering point. To do this, the derivative of $D_i^2$ with respect to $\Delta z$ is calculated.

$$\frac{d(D_i^2)}{d\Delta z} = 2 \cdot \left( \frac{(x_i - x_p)\beta + (x_p - z_i) x_{sp}}{z_i + \Delta z} \right) \cdot \left( \frac{(x_i - x_p)\beta + (x_p - z_i) x_{sp}}{(z_i + \Delta z)^2} \right) + 2 \cdot \left( \frac{(y_i - y_p)\gamma + (z_p - z_i) y_{sp}}{z_i + \Delta z} \right) \cdot \left( \frac{(y_i - y_p)\gamma + (z_p - z_i) y_{sp}}{(z_i + \Delta z)^2} \right) \quad (3.29)$$

To find the $\Delta z$ value that renders closest to the required pixel, $\frac{d(D_i^2)}{d\Delta z}$ is set to zero and the equation is solved for $\Delta z$. The result is shown in Equation 3.30.

$$\Delta z = \frac{A - B}{C + D} \quad (3.30)$$

where

$$A = \left( (x_i - x_p)\beta + (z_p - z_i) x_{sp} \right) \left( (x_i - x_p)\beta - x_{ri}z_i + x_{sp}z_p \right)$$
\[ B = \left( (y_i - y_p) \gamma + (z_p - z_i) y_{sp} \right) \left( (y_p - y_i) \gamma + y_{ri} z_i - y_{sp} z_p \right) \]

\[ C = \left( (x_i - x_p) \beta + (z_p - z_i) x_{sp} \right) (x_{ri} - x_{sp}) \]

\[ D = \left( (y_i - y_p) \gamma + (z_p - z_i) y_{sp} \right) (y_{ri} - y_{sp}) \]

With the above equation, it is now possible to pick a vertex on the silhouette edge of the model object and calculate \( \Delta_i = \begin{bmatrix} \Delta x_i \\ \Delta y_i \\ \Delta z_i \\ 1 \end{bmatrix} \) for each of the remaining vertices on the silhouette edge. This \( \Delta_i \) is the amount the object should be moved in eye space to get the vertex \( P_i \) closest to the point where it would render at \( \begin{bmatrix} x_{ri} & y_{ri} \end{bmatrix} \) and also maintain the point \( P_p \) at a point where it renders to \( \begin{bmatrix} x_{sp} & y_{sp} \end{bmatrix} \).

In a perfect situation, the value of each \( \Delta_i \) would be the same. However, as mentioned previously, the probing process does produce errors. It is possible that there will be some variations in the value calculated by Equation 3.30 for each of the vertices on the silhouette edge of the model object. In fact, if the position the randomly picked point, \( P_p \), should render to is in error, then all the values for \( \Delta_i \) will vary. Also, since the pixel that a vertex should render to is not a point but an area, there is some error associated with that. Because of this, even if every probe vector and vertex adjustment vector has been calculated correctly, because of the size of the pixels, the values of every \( \Delta_i \) might not be the same.

An acceptable amount of error was developed for the amount of "play" that could be associated with the values of \( \Delta_i \) as follows. The \( \Delta_i \) with the smallest \( \Delta z \) (most negative) value is found. Adding that value to the object position, \( P_o \), results in the closest position that the object could come to the observer. The point \( P_o + \Delta_{i_{\text{min}}} \) is rendered to the screen using Equation 3.1. This results in a pixel on screen space.
\[ \begin{bmatrix} x_r \ y_r \end{bmatrix} \]. The \( x \) and \( y \) coordinates of that pixel are incremented by one — the diagonally adjacent pixel. \( \begin{bmatrix} x_r + 1 \ y_r + 1 \end{bmatrix} = \begin{bmatrix} x_s \ y_s \end{bmatrix} \). Now, using Equations 3.2 and 3.6, the eye space coordinate of the point that renders to \( \begin{bmatrix} x_s \ y_s \end{bmatrix} \) and is closest to the point \( P_o + \Delta_{i_{\text{min}}} \) is calculated. Using this point, \( P'_e = \begin{bmatrix} x'_e \ y'_e \ z'_e \ 1 \end{bmatrix} \), and the original point \( P_o + \Delta_{i_{\text{min}}} \), the distance between the two can be calculated using the standard distance equation \( (D_{\text{err}}) \). This value is the maximum distance a vertex can move in eye space and still render to a point that is at most the distance of a pixel diagonal away from its original pixel, at the distance \( P_o + \Delta_{i_{\text{min}}} \).

Using the value \( D_{\text{err}} \), the list of all \( \Delta_i \) calculated for a fixed point \( P_p \) is traversed. Each unique \( \Delta_i \) has a count associated with it. This count keeps track of the number of other \( \Delta_i \)'s that are closer than \( D_{\text{err}} \). The \( \Delta_i \) value with the highest number of close neighbors is considered the best \( \Delta \) for the object.

This \( \Delta \) is acceptable only if the number of neighbors for the best \( \Delta_i \) is greater than half of all the vertices. If it isn't, the randomly chosen point \( P_p \) was probably an error point. Another point is chosen and the process begins again. If, after choosing all the points, no \( \Delta_i \) value satisfies the majority criteria, there were too many probes in error for a good match. The \( \Delta_i \) with the highest number of matches from all the fixed points is chosen as the \( \Delta \) for the object.

The model object is then moved by adding the \( \Delta \) value to all the vertices in the object. If none of the probe vectors were in error, this should move the model object so that it is as close as it can get to rendering at the same spot as the original reference object. This means that it must be in the same three-dimensional location.
Figure 30: Because of digitization of the original image, aliasing errors will occur when calculating the position of a point in three space.
as the reference object.

**Iterative application of process**

This method is not perfect: the edges found in the digitized image may be ragged and the normal probes might hit an edge they shouldn't hit. However, as the modeling object gets closer in size to the actual reference object, the chance of a miss hit by a normal probe goes down. Therefore this algorithm is naturally iterative. After the initial placement of the model object by the user, the normal probes are sent out and the object is moved again and again, until the change in position of the object from one iteration to another falls below a limit, $D_{err}$. This limiting distance is dependent on the distance the object is from the screen.

This brings up a big problem. As the reference objects get farther away from the camera, the size of $D_{err}$ goes up and the amount of information detectable by the edge detection algorithm goes down. In other words, the object gets smaller as it get farther away. Unfortunately, the limit that halts the recursive application of this algorithm grows as the distance of the object grows. This means that the accuracy of the algorithm gets worse as the object gets far away.

Of course, this should seem obvious. When we are confronted with judging the distance of a far away object the best we can do is say something like, "Well, that car looks to be about a mile or so off." As the car comes closer, we can update that opinion more accurately.

In a single image, the error is uncorrectable. However, in a sequence of images
where either the camera or the object is moving, more information about the position can be collected. By remembering the position of the model object from frame to frame, some smoothing of the object's position can be done. This is necessary to avoid problems where the object's shadow jumps from frame to frame because of the error term in calculating the object's position.

**Refinement of the alignment using more probes**

Because of the difficulty of generating model objects for the reference objects, it is assumed that they will be fairly simple. If a certain amount of positional error can be tolerated, a complex object could be modeled with a relatively simple one, as long as the basic shape of the two objects are similar. However, this leads to a silhouette that is composed of very few edges. Which, in turn, generates only a few probes. If one of those probes finds an erroneous edge, that error will have a much more devastating effect than if there was a large amount of probes. So, if there was a way to increase the edges, and hence the probes, in the silhouette of our model object, the effect of erroneous probes would be reduced. This can be done by subdividing the polygonal representation of the model object.

**Errors in the adjustment of the model object**

Various errors in the calculation of the three-dimensional position of the model object have been discussed previously in this chapter. Other types of error can also occur.

When the reference image is edge detected, the optimal result would be edges around the reference objects that were exactly one pixel wide. That isn't always the
case. The basic concept behind all edge detection algorithms is to examine the image for rapidly changing parts and mark those parts. If the original image is taken away, the marks that are left make up the edge detection image. Unfortunately, the types of edges in images vary between edges that are very sharp and distinct and edges that are very soft and fuzzy. A good edge detection algorithm will mark both kinds of edges — but it may mark the soft edges with more than just a thin line. The lines that represent a soft edge may be many pixels thick. Edge thinning algorithms do exist (see Levialdi [24]), but the thinning might not accurately position the edge. (An example of the types of edges an image might have is shown in Figure 14.)

![Figure 31: Soft edges around a reference object result in large edge detection lines. The grey pixels represent the optimal results of an edge detection. The black pixels are extra pixels added because of the soft edges of the reference object.](image)

If the model object is aligned to a reference object with fuzzy edges, there is the
possibility for error. The actual edge of the reference object may be represented by
the pixels in the center of the line generated by edge detection but the probes sent
from the model object edges will stop on the first pixel of the line that the encounter.
They could therefore miscalculate the distance between the model object edge and
the reference object edge by one or more pixels. An example of this is shown in
Figure 31.

The actual error in the calculated three-dimensional position that these additional
pixels cause can be calculated. To do so, imagine a horizontal bar, length \( D \) units
long. An photo of the bar is taken, digitized and edge detected. There are two cases:
the edge detection process could have been correct, or the edge detection process could
have added additional pixels to the outline of the bar. This example is illustrated in
Figure 32.

In Figure 32, the horizontal bar is represented by the lines labeled \( \text{Bar A} \) and \( \text{Bar B} \). \( \text{Bar A} \) is the three-dimensional position of the bar if no error during edge detection
has occurred. If that was the case, the photo of \( \text{Bar A} \) produced a line on the image
of length \( L \).

If, however, there was error in the edge detection process, the length of the line
on the image would have gotten longer. In this case, an error term \( E \) was added to
both ends of the line. The line would be \( L + 2E \) units long. When the model of the
bar is aligned to that image of the bar, its position will be calculated to be at the
place indicated by \( \text{Bar B} \).

Based on the length of the bar \( (D) \), the correct length of the image of the bar
Figure 32: An example of the three-dimensional positioning error that can occur due to extra pixels added during the edge detection process.
(L) and the error in edge detection (E), the error in the calculation of the threedimensional position of the model object can be calculated. By using a horizontal line as the example, the calculations are simplified but the results are applicable to all objects.

In the figure, the points labeled 1 and 2 have the screen coordinates:

\[
\text{Point 1 : } [x_s \ y_s]
\]
\[
\text{Point 2 : } [x_s + L \ y_s]
\]  
(3.31)

Using Equation 3.1, the three-dimensional positions of the endpoints of Bar A (labeled 3 and 4 in the figure) based on a value \(z_e\) can be calculated:

\[
\text{Point 3 : } [x_s z_e \tan \theta \ y_s z_e \frac{\tan \theta}{\alpha} \ z_e]
\]
\[
\text{Point 4 : } [(x_s + L) z_e \tan \theta \ y_s z_e \frac{\tan \theta}{\alpha} \ z_e]
\]  
(3.32)

Since the bar is horizontal and parallel to the image plane, the z coordinate of both endpoints will be the same. Since the length of the bar is known, the x position of Point 3 should be \(D\) units away from the x position of Point 4. Expressing that in an equation allows for \(z_e\) to be calculated:

\[
x_s z_e \tan \theta + D = (x_s + L) z_e \tan \theta
\]  
(3.33)

Solving for \(z_e\):

\[
z_e = \frac{D}{L \tan \theta}
\]  
(3.34)

If the error pixels added by the edge detection are taken into account, a new set of equations can be developed. The points 1 and 2 are now represented by:

\[
\text{Point 1 (error) : } [x_s - E \ y_s]
\]
Point 2 (error) : \[ x_a + L + E \ y_a \] (3.35)

The endpoints of the bar, given these new rendering points, are shown in the figure as points 5 and 6. These points have the following equations:

Point 5 : \[ (x_a - E) z_e \tan \theta \ y_a z_e \tan \theta/\alpha \ z_e \]  
Point 6 : \[ (x_a + L + E) z_e \tan \theta \ y_a z_e \tan \theta/\alpha \ z_e \] (3.36)

Again, solving for \( z_e' \) using the \( x \) coordinates of the endpoints of the bar and the length of the bar, \( D \):

\[ z_e' = \frac{D}{(L + 2E) \tan \theta} \] (3.37)

Subtracting the two \( z \) values results in an equation which relates the amount of positional error to the amount of edge detection error, based on the actual and apparent size of the object:

\[ \Delta z = z_e' - z_e = \frac{D}{\tan \theta} \left( \frac{1}{L + 2E} - \frac{1}{L} \right) \] (3.38)

If \( E = 0 \), the amount of error is equal to zero. As \( E \) gets very large, \( \Delta z \) approaches a limiting value of \( \frac{-D}{L \tan \theta} \), which would mean that the bar is right on top of the eyeposition. As \( E \) gets close to \( -L \) the \( \frac{1}{L+2E} \) term gets very large, which makes sense.

If the object appears to the program as very small, but that appearance was caused by errors in the edge detection, the program will calculate it to be very far away.

Another source of error in the calculation of the model object's three-dimensional position occurs because the model object is oftentimes just an approximation of the reference object. In Figure 20, a cylindrical reference object is modeled by an hexagonal model object. Because of this, when the probe vectors are sent from the center
of the silhouette edges of the model object, they will not be intersecting similarly sloped reference edges. In the case of the model object's and the reference object's matching edges having the same slope, the model object will eventually be able to align perfectly with the reference object. (Assuming, of course, that none of the other errors mentioned here occurs.) However, if the slopes of the matching edge are not aligned, one can get errors similar to the errors found in a misrotated object. (See the discussion on page 67.)

This entire project relies on the fact that the size of a reference object isn't changing. Calculating the distance from the camera of an inflating balloon using this method is impossible. (Unless it was possible to measure the size of the balloon at each frame of the film.) While balloons don't often inflate in the movies, things like clothes and pillows and other soft object are often filmed. In the case of clothes, as long as an actor is not wearing too billowy of an outfit, the error associated with the changeable nature of the clothes will be acceptable. (This also occurs with loose, large hair styles.) If the clothes or hair or pillows are too malleable for acceptable error, the fact that they are attached to some more solid thing can be used to correct the error. If the hair on an actor moves and changes size too much, the face of the actor can be used as the reference point of the head, and the hair model can be positioned so that it matches with the position of the model of the face of the actor. Likewise with billowy clothes and fluffy items like pillows. There will most likely be a nearby (hopefully attached) object that the billowy object can use as a reference point for its position. This ad hoc positioning would all have to be done by hand, as
3.3.4 Computer Assisted Rotation

One of the basic tenants of this thesis is that the user of the program must position the model object near the reference object. This positioning operation also includes rotating the object so that the model object and the reference object are very closely aligned in terms of orientation. This is vital for the correct operation of the algorithm described above. (See the discussion on page 67.) Large errors in the calculation of the distance of the objects can occur if the objects are not properly aligned. This section will explain why the user of the program is responsible for this step and not the computer.

Figure 33: The three basic steps to the procedure of correctly orienting a model object with a reference object.

Reference Points

In Figure 33 the three basic steps to automatically orienting a model object are shown. The reference object is the shaded cube in the figure. In Step 1, the user has placed
a model object on top of reference object. Then, in Step 2, the computer locates matching reference points on both the model and reference object. At this point, we run into impossibilities. For an arbitrarily oriented reference and model object, it is impossible to have the computer automatically locate matching reference points on the two objects. It has a hard enough time finding lines in the reference image, much less points. Because the lines in most images are digitized and hence bumpy and irregular, the automatic location of corner points is impossible. The reference object might not even have corners!

If, however, some reference points are found, the next step would be to calculate the amount of rotation to apply to the model object so that the reference points on both objects are approximately the same distance away. This, too, is difficult because it is not always the case that the screen space distances between points should be the same. Consider the two points marked A and B in Figure 33, Step 3. The matching reference and model points in both cases are at obviously different distances from each other.

**Refinement of an Already Oriented Model Object**

Because of the difficulties in automatically orienting a model object based on random initial orientation, a refinement of an initial orientation was considered. In this case it is assumed that the user has positioned the model object "close" to the reference object, not only is it centered on the reference object, but it has also been oriented so that it is approximately in the same orientation as the reference object.
At this point, the user identifies a single point on the silhouette of the reference object and the matching vertex on the model object. For reasons that will become clear soon, the point must be on the silhouette of the object. Points in the interior of the object do not give enough information. Also, for the best performance of this algorithm, the point picked should be as far away from the center point of the object as possible. This reduces the chance for error, as will be explained soon.

There is one more limitation to the procedure that follows: the procedure must assume that the model object has been placed so that the center of it is aligned with the center of the reference object. This doesn't mean that the center of the model object must be at the exact three space location as the reference object's center. (That would mean that the location of the reference object is known!) It does mean that the center of the model object is somewhere along the line defined by the reference object center and the eye point. On other words, the center of the reference object and the center of the model object render to the same pixel on the screen.

Using the user located reference point and the edges on the silhouette of both the model and reference object, it is possible to more accurately orient the object than was first done by hand.

**Automatic Orientation Refinement of Model Objects Using a Reference Point**

Once the user has identified a reference point on the silhouette of the reference and model object, a vector can be generated from the eye point through the pixel on the
screen. The correspondingly identified point on the model object should eventually end up at that position.

Without moving the object in space — that step comes later — a specific point on the model object can only be rotated around the center of the model object. By simple rotation of the object, the reference vertex describes a sphere of around the center of rotation of the object, with a radius of the distance between the center and the reference vertex.

Figure 34: A vector is sent from the eye point, through the reference point, and intersected with the sphere of possible positions.

The vector sent from the eye point through the pixel will either completely miss the sphere, strike the sphere at one point (tangent point) or intersect the sphere in two places. The best possible case is that the vector is tangent to the sphere. This
implies that the perceived size of the object is approximately correct and the position of the object is nearly correct. It is trivial to calculate the amount of rotation to apply to the object so that the reference model vertex moves to the tangent point.

Figure 35: Possible cases for the intersection of the reference point vector with the sphere of possible positions.

If the vector either misses the sphere or hits it in two spots, that means that the object is either too far away from the camera (too small) or too close to the camera (too big). In this case, the closest point on the vector to the center of the object is found. Another vector is created from these two points. That vector is intersected with the sphere of possible positions. That point is the place that the reference model vertex should be moved. It is the place that gets the point the closest to the place it's supposed to be. When the object is moved in three space and correctly positioned,
the points should line up correctly.

Figure 36: The center of rotation and the reference point location create an axis around which the object can be rotated.

Once the vertex has been moved, another vector is created. This vector is an axis that the object can rotate around. Since one vertex is in the correct spot, it should not be rotated away from where it is positioned. So, the only possible rotation is around the axis defined by the center of the object and the reference model vertex in its new position.

Even though the rotation of the object has been severely constrained, there must still be a way to tell when the model object is aligned with the reference object. Once again, probes from the edges of the model object must be used. In the size-alignment phase, only one probe per edge was sent out. However, in this phase, many probes
Figure 37: Additional auxiliary probes are sent from the silhouette edge of the model object. There provide a difference in slope between the edges — unless a corner or a bump is encountered.
must be sent so that the difference in slope of the two edges can be calculated. In Figure 37, multiple probes are sent from the middle section of each silhouette edge of the model object. The probes are gathered around the center of the edge to avoid what has happened on the right hand side of the object. In that case, the probes have encountered a corner in the object and will not give an accurate calculation of the angle between the two lines. One can't say that only the probes that return a calculation of constant slope will be used since in the non-ideal case the edges will be bumpy and not all together straight. It will be rare that a smooth edge will be encountered. Therefore the average of the slopes returned by all the probes will be used as the slope between the two lines. The closer to zero the better.

Because there is no mathematical representation of the digitized reference object silhouette edges, no direct mathematical solution to the amount of rotation can be found. However, by constraining the possible rotation of the model object to only around the axis defined above, it is possible to do a search of possible rotations. Iteratively, the rotation that returns the minimal amount of slope difference between all the edges can be found. Thus, with an initial placement of the model object and the identification of a reference point on both the reference and model objects by the user, a refinement of the initial rotation can be accomplished.

3.3.5 Calculation of the Lens Parameters of the Camera

The parameters of the camera lens are currently arrived at by ad hoc methods. Steve Anderson, in his Master's Thesis [1], shows how to accurately model a camera lens in a renderer. When filming the image, it is perfectly reasonable to have the photographer
record the type of lens used. However, it is be possible to calculate the lens parameters based on information in the photograph.

If an object's orientation and size are known, it is be possible to calculate the aspect ratio to be used when rendering based on distance between various reference points on the image. This calculation has the same limitations that the previous rotation calculation had; there is no reliable way to identify various reference points (corners) on an image. The user of the program would be required to identify the position of various points on the object.

3.4 Deforming the Model Object for a Perfect Fit

Once the model object has been translated so that its apparent size is the same as the reference object’s size, the object’s approximate distance from the camera is known. However, the model object and the reference object may not be precisely aligned yet. At this point, the task is to get the silhouette of the model object to exactly match the silhouette of the reference object. The model object is used to calculate where new shadows would fall on the reference object. If the silhouettes of the model object and the reference object don’t match exactly a shadow might fall across empty space or start to late on the object. Both of these possibilities are very noticeable on either a single photograph or as part of a film and must be avoided.

There are three reasons for possible mis-alignment of the silhouette:

• The model object is of less complexity than the reference object and hence the bumpy silhouette edges of the reference object have been modeled with a single
edge in the model object.

- The model object is being used to approximate a reference object that possibly changes size and shape (e.g., hair, trees). Not only will the model object be simpler than the reference object, but the shape of the reference object might have changed from the one that was used to generate the model object.

- The model object is mis-rotated and the alignment of the two objects is consequentlly off. A line on both objects that should be parallel and aligned might be mis-aligned after the translation.

By deforming the silhouette edges of the model object it is possible to match the silhouette of the model object and the reference object. While this will change the shape of the model object, in most cases this is desirable. As an example, consider the example of modeling a person's head with a simple egg shape. If the egg is properly sized, it is possible to find the three-dimensional position of the person's head in the scene with reasonable accuracy. Now, the egg is used to calculate additional shadows that may fall on the head. If a smooth egg is used, a smooth shadow will be created. This is not desirable. What is wanted is a bumpy shadow that follows the contours of the head. This can achieved by modeling the head with a more accurate model object than an egg or by deforming the egg according to the contours of the head and thus getting some perturbations on the egg's surface. The best method would be combining the two methods described; modeling the head with an accurate head-shaped model object and deforming the model object so that it fits more perfectly with the reference head object.
This section will describe the deformations that can be performed on an object and discuss some of its limitations.

3.4.1 Alignment of the Silhouette Edges of Objects

Initially, the silhouette of the model object is deformed so that it exactly matches with the silhouette edge of the reference object. To minimize the effect of the silhouette edge deformation on the interior of the object, the model object is first triangularized. Any polygons in the object are decomposed into triangular subpolygons. This makes sure that the upcoming deformations will still result in planar polygons. (Planar polygons are usually necessary for rendering algorithms.)

For each silhouette edge in the model object, the following procedure is followed:

1. The vertices of the silhouette edges are compared to the edge detection image. If the vertex renders to an edge pixel, it is assumed that it is in the correct spot and its position is unchanged. If, however, the vertex does not render to an edge pixel, the edge detection image is searched, starting from the rendering point of the vertex, to find an edge pixel that is closest to the rendering pixel of the vertex.

2. The vertex is moved in three space, using the algorithm discussed on page 64, so that it will render to the located edge pixel yet is a minimum distance away from its starting point.

3. Once this is done, the algorithm has produced a model object of the type shown in Figure 38, “Deformed State”. If the example is examined closely, it is obvious
Starting state: Model object positioned on top of Reference object.

Deformed state: Silhouette vertices have been moved to a nearby edge pixel.

Figure 38: Once the model object has been positioned in three-dimensional space (the starting state), the vertices on the silhouette edge are moved to align the model edges with the reference edges.
that some of the silhouette edges of the model objects don't completely overlap the reference edges.

The next step in the algorithm is to check for this type of mismatch. Using Bresenham's line drawing algorithm, a line is drawn from the two endpoints of each model object silhouette edge. The line is traversed and any pixel that isn't marked as a reference edge is marked.

If no pixels are marked, the edges match to an acceptable degree. On the other hand, if there are areas where the model object isn't aligned with the reference object, more adjustment must be made.

4. If more adjustment is to be made the polygons that have edges along an edge with a mismatch in it are subdivided into more triangular polygons. The center of subdivision occurs at the point in the middle of the largest mismatch on the silhouette edge.

5. The entire process is repeated with this new, more complicated object. The higher complexity of the object along the edges with more mismatches will allow the object to be sufficiently deformed so that all the pits and bumps in the reference object's edges are matched. The results of the second iteration of this algorithm are shown in Figure 39.

It is important that when the object finds a reference edge pixel to move a vertex to, that edge is not already covered by some other edge of the object. This could lead to an infinite loop situation where, for a certain gap, one pixel is the shortest
Figure 39: The polygonal representation of the model object is triangularized and then the model object silhouette edges are matched to the reference object edges with more accuracy.
distance away from the midpoint of the line, and by moving that midpoint to that pixel, nothing is changed in the line. To solve this, only pixels that are not covered by edges of the object are considered as possible choices for vertex placement.

Unfortunately, it's very difficult to calculate the optimal position for the silhouette edges of the model object. Given that there might be fuzzy edges (as shown in Figure 39) it is very difficult for the algorithm to decide on whether to align the silhouette edges to the outside or inside of the edge detection area. Since this is the case, the user has been given the ability to manually place the vertices of the object.

By placing the mouse near a vertex of the model object, the user can move the vertex so that it renders to a specific pixel. The two-dimensional motion of the vertex (being dragged across the screen) is translated into a three-dimensional position change by the method described on page 64. This algorithm ensures that when a vertex is positioned so that it renders to a certain pixel it is only moved the minimal distance in three space from its starting point. With this, the user can affect fine tuning changes to the deformation that the algorithm calculates.

In Figure 39 it is plainly visible that what started out as a perfect cube is no longer anything close to perfect! Even though the reference object may be a perfect cube, this step is necessary to make sure that the lighting effects line up correctly on the cube. If the shadows that are cast on the cube have a slight distortion in them because of the deformation of the cube, that's all right. Those distortions are much less jarring to the eye than a shadow that runs off the edge of the cube and keeps on going into the thin air.
It is also possible to minimize the effect of the silhouette edge deformation by subdividing the polygonal representation of the model object multiple times before the first silhouette edge deformation. By doing this the effects of the deformation are localized to the polygons that are close to the silhouette edge and the interior polygons are left unchanged. Subdivision of this nature is left up to the user of the program.

3.4.2 Applying the Silhouette Deformation to the Interior of the Object

Once the silhouette of the model object has been modified, the interior of the object should reflect the changes wrought on the silhouette. This helps mask any silhouette edge deformations that might stand out when rendering the scene for shadows. It also adds visual complexity to the interior of a simplified model object. (This is not always wanted. It may be the case that the deformations of the silhouette edge should be confined to the silhouette edge. Hence, this step is optional and under user control.) Also, during the silhouette edge deformation stage (described in the previous section) a silhouette vertex might be moved so that an edge that wasn't previously a silhouette edge of the object, becomes one. By applying the silhouette deformations to the interior of the object this problem can be remedied.

A simple weighted average of the translations of silhouette edge vertices on interior vertices produced acceptable results. For each vertex in the interior of the model object that hadn’t been moved during the silhouette edge deformation stage, its neighboring vertices that had been moved were marked. The average amount of
translation that they went through was calculated. This was multiplied by a weighting factor that decreased as the vertices went in to the object, away from the silhouette edge. This average translation was then applied to the vertex and that vertex was marked. If a vertex had no neighbors that had moved, it was not moved during this stage. After all the vertices that had neighbors that moved were moved themselves the process is repeated, with the weighting factor reduced. This ensures that a deformation of the silhouette edge will propagate throughout the entire object, with diminishing effects the farther from the edge it gets.

Figure 40: The translation of the vertices on the deformed object (represented by solid lines) from the initial points on the original model object (dotted lines) are used to calculate adjustments to interior pixels.

An example of this process is shown in Figure 40. Most of the silhouette edges have been adjusted in some way. So, the first pass through the interior adjustment
would calculate the new positions for the vertices which are neighbors of the silhouette edge vertices. One such vertex is labeled 3 in the figure. All the neighbors of point 3 are marked and the amount of translation they underwent is averaged. In this case one of the vertices that is a neighbor to 3 and has been translated during the deformation process is point 2. It has been moved from its initial position, point 1. This translation, along with the others determines the new position for point 3. The point labeled 4 doesn't have any neighbors that have been moved during the silhouette edge deformation process. However, after point 3 and others have been moved, point 4 will have plenty of neighbors that will have been translated from their original position. The process will be repeated again, but with a smaller weight so that points like point 4 are not moved too much.

3.5 Chapter Summary

This chapter has discussed the mathematics and algorithms behind the procedure to model a three-dimensional scene using only a single photograph. First, the photograph of the scene is digitized. In order for the computer to make sense of the photograph, the edges in the photograph are found and, using an edge connection algorithm, the edges are "cleaned up" and made usable for the rest of the procedure. Using the knowledge of the size and shape of the objects in the scene, model objects are created and aligned to the reference objects in the image through an iterative process of edge matching. Since the size of the object and the position and parameters of the camera are known, the three-dimensional position of the objects in the scene can be calculated from their size when aligned. After the objects have been aligned,
in preparation for the composition of new objects into the scene, they are deformed around the reference objects so that the outlines of the model objects and the reference objects match exactly. This is necessary to prevent artifacts from the image composition stage from showing through. Finally, the deformation to “shrink-wrap” the model object to the outline of the reference object is applied to the interior of the model object to give the object some visual complexity and attempt to approximate any interior crenellations that are otherwise un-modelable.

When all of this is performed on a scene, the end result is a complete three-dimensional computer model of the original scene. This can now be used to predict where shadows should fall and which object obscures other objects. These problems will be discussed in the next chapter.
CHAPTER IV

COMPOSITION OF IMAGES

This chapter discusses how, once the object positions are known, a new object can be introduced into the scene and how the lighting effects of that new object can be taken into account by the objects already in the scene.

At this point, the entire image has been modeled. A complete three dimensional representation of the scene has been created. Inserting another object into the scene is trivial. The objects that are either obscured by or obscure the new object can be included into the rendering of the new object. Only the visible parts of the new object will be composited with the old scene and any parts of the original scene that are covered by the new object will be covered.

Visibility is not the only aspect of inserting an object into a scene. Depending on the position of the object, the original objects and the lights illuminating the scene, there might be new shadows cast on both the old and new objects. Depending on the surface characteristics of the new and old objects reflections might be added or altered. If the new object inserted is a source of light, the old objects would be illuminated differently. There are many difficulties in adding lighting effects to the scene: a new shadow may be added so that it intersects a pre-existing shadow and care must be taken not to reduce the intensity of the pre-existing shadow; if
new shadows are added to an object with specular highlights any highlights must be removed completely, not just reduced in intensity; finally, the model of the reference scene is only an approximation, information from the original image must be used to take into account the details not modeled by the three dimensional representation.

It is possible to calculate the changes in lighting that are caused by the introduction of a new object. However, the calculation is hindered by the fact that the three dimensional representation of the scene is only approximate. The overall shape and position of each object in the scene is known, but the objects themselves are only approximated. Any local variations of the surface of an object can not be modeled.

Figure 41: When filming a scene, the camera position, light positions and object sizes must be recorded.
4.1 Required Information to Incorporate New Objects

If a painter were asked to add something to a painting he had just finished — say a cupid in the upper left hand corner of the painting — he would ask various questions: “Should the cupid be in front of, or behind, the column?” and “How far away should the cupid be?” (Which might answer the first question.) The painter would ask himself, “Where is the light coming from in the scene? Will the addition of this object cause more shadows? Will anything in the scene shadow the object?” These questions are some of the things this thesis is trying to solve. The first set of questions can already be answered. Chapter III discussed how the location of various objects in an image can be calculated. An artist would only need to specify the three dimensional location of the object and the coverage of the object by objects already in the scene (and vice versa) can be easily computed.

The second set of questions concerning the lighting effects pose a more serious problem. If the new object is going to cast shadows or have shadows fall on it, the position of the light sources in the scene must be known. If some of the light sources are actual objects in the scene, their position is already known by the calculations done in the previous chapter. However, as is the case in most still photographs and film-making, there are usually many light sources outside the field of view of the camera. For this algorithm to work, the position of the light sources must be recorded.

The position of each light source is not enough. The size and shape of the light source (Is it a point source? Is it a box light? Is it a circular light?), the intensity
of the light, and the color of the light must be measured. The size of the light source is easy to measure, but the intensity and color are a little less standard. For the experiments conducted in this project, a standard set of color bars were photographed at different distances from each light. From these photos the intensity and color of each light could be calculated.

In summary, for every frame of film, this thesis assumes that the position and lens parameters of the camera, the approximate sizes of all objects in the scene and the location and attributes of all light sources in the scene are known. In most cases, this isn't as large an amount of information as it seems. The camera is quite often fixed in place during a shot. If it isn't, there are motion-control devices that record the position of the camera at each frame. Objects usually change their positions, they rarely change their sizes, so once the size of an object is known it can be used for all following frames. Finally, lights very rarely move during a scene, especially the ones outside the field of view of the camera. So, once the position and attributes of the lights are known, they can be set as constants for the rest of the scene.

4.2 Analysis of the Original Image

Returning to the painter example, once the painter knows where the cupid should be in the scene and what lights are being blocked, the artist must now add the object to the scene. Painting the cupid, perhaps in the upper left hand corner, behind the column, with the appropriate shadows on it isn't hard. The painter knows what should be visible and what blocks light. But the cupid, likely as not, is casting shadows on things already in the scene and the painter must modify those objects so
that the shadows on them are now correct, taking into account the fact that there is now a cupid in the scene. This isn’t simple, because an object to be shadowed by the cupid might be already shadowed by some other object. In this case the painter shouldn’t darken the object because no extra light is being blocked.

Figure 42 illustrates this concept. In the original image, the light source coming from the right (not visible in the figure) is causing the rod to cast a shadow onto the plane. The addition of a triangular object in the scene would require a new shadow on both the rod and the plane. A naive method of adding the new shadow would be to compute the area on each object that is shadowed by the new object and reduce the intensity of light on those areas. However, as one can see, that leaves some problems with pre-existing shadows from that light source. In the image labeled “Incorrectly Shadowed”, the new shadows from the triangular object have been added without regard to any previous shadows. Notice the solid black area on the plane. By assuming that the entire area should be reduced in intensity by the same amount, the previously shadowed area is darkened — even though the light from the light source has already been blocked by the rod. The same thing has happened on the rod itself. The rod is self shadowing, but darkening the entire surface that is placed into shadow by a new object is, again, a mistake.

The correctly shadowed image is shown in Figure 42. Notice that the plane doesn’t have any dark spots. This makes sense since, with only one light source in the image, a point on an object can either see the light or it can’t. There should be no variation in the intensity of areas in shadow.
4.2.1 Rendering the Model Object Scene to Approximate the Reference Scene

It is necessary to locate the presence of shadows on a reference object prior to the addition of new shadows so that they are not erroneously darkened. Initially, this doesn’t sound terribly difficult. There exists a three dimensional representation of the reference image, using the method developed in Chapter III. Sending the scene description to a renderer equipped to deal with shadows would show us where pre-existing shadows should be on the image.

Unfortunately, because the three dimensional description of the reference scene is only an approximation, the results the renderer returns will, in all likelihood, be different than the real scene. The effects of this can be seen in Figure 43. The rendered image was created with the model representation of the rod slightly closer to the light source than the actual rod was. This produced a shadow on the plane that was slightly offset from the actual shadow (shown in the dark outline). When

Figure 42: The addition of a new object in the scene requires careful calculation of new shadows.
Figure 43: Using the rendered image of the model objects to locate shadows might lead to errors.

the new shadow was added to the plane, an area that was thought not to be in shadow was darkened — producing the dark area — and an area that was thought to be in shadow was not darkened — producing the white sliver in the shadow.

Therefore, using a rendered image generated from the model object representation of the scene will not produce consistently good results. The rendered image can be a good starting point in the analysis of the reference image and the location of the actual pixels in the reference image that are in shadow.

4.2.2 Problems in Calculating Intensities of New Shadows

There is another reason to find the actual shadows in the reference image before adding new shadows to the image: even though the location and attributes of the lights in the scene are known, the shadows added to the scene might not be of the same intensity as the actual shadows in the reference image.

No matter how much information is known about the lights that illuminate each
photograph, the model of the scene will differ from the actual scene. Consider a simple photograph of a ball on a table. If all the positions and sizes of all the objects and lights in the scene are recorded, and the best radiosity renderer was used, an image produced from the data would not be an exact match of an actual photograph of the image. There is too much that can’t be measured easily: the film’s chemical properties, the atmospheric effects (air pollution, haze), the exact physical properties of the objects in the scene (dirt on the surface, small textures) and light from sources that can’t be measured (reflections off the camera and cameraman, for instance). If a new shadow was added to an object because some new object blocked a light source, that shadow would probably not match the intensity of an actual shadow on the object in the photograph. Therefore, the reference image must be used to calibrate the rendering of the new image.

4.2.3 Using the Model Scene Image to Analyze the Reference Image

To analyze the reference scene image, the model scene is rendered using ray tracing. Ray tracing has advantages over other rendering techniques in that it is simple, it calculates shadows readily and is easy to modify so that more advanced lighting concepts (such as pnuembras, motion blur and complex light sources) can be used. The only disadvantage of ray tracing is that it is slow. This is an acceptable drawback for the proof of the concepts behind this thesis.

To illustrate the rendering of the reference scene and the information that needs to be generated, a sample reference scene was created and rendered using the ray
tracer developed for this project. (See Figure 44.) The scene has three objects in it: a floor, a cylinder and a floating cube. There is a single light source in the image located above and to the right of the observer. The cube is positioned so that it casts a shadow on the cylinder. The light source also causes a specular highlight on the cylinder.

For this example a new object is to be introduced to the scene shown in Figure 44. Another cube is to be added to the scene between the cylinder and the original cube. The desired results are shown in Figure 45. This image was generated using the knowledge of the exact positions of the objects and the same ray tracing package that was used on the first image.

If the positions of the objects in the first image are not known, the procedures
Figure 45: The results of inserting another cube into the reference image of Figure 44.

Figure 46: Wire frame approximations of the actual objects are translated to the same three dimensional position as the reference objects.
developed in chapter III can be used to create a model of the scene. Objects are created that are the same size as the objects in the actual scene. Then, the objects are positioned (using the program LIVE, described in Chapter V) so that they are as close to the position of the actual objects as possible. The results of this procedure are shown in Figure 46.

![Diagram of objects](image)

**Figure 47:** Any new objects that are added to the scene are positioned in reference to the rotoscoped reference objects.

Using LIVE, any new objects that are to be added to the scene are placed in their appropriate spots. The new objects and the rotoscoped reference objects make up the complete description of the model object scene. The new cube added to the example model scene is shown in Figure 47. This model object scene is now rendered using a special ray tracing program written for this project that returns more than just the image.
The ray tracing program first renders the visible portions of the new objects using a full illumination model. Since the light positions and parameters are known and the approximate positions of the reference objects are known, the new objects can be completely rendered with shadows. Only the visible parts of the new objects are rendered — this is possible because of the knowledge of the reference object positions. In this example, the cube is rendered using faceted shading (and in full color, even though the image shown is in black and white). Since the reference cube falls between the new cube and the light, a shadow is cast on the new cube. This is easily calculated in the ray tracing program and the results are shown in Figure 48. In this case, since the entire object is visible, the entire object is rendered.

Figure 48: Any new objects are completely rendered. Only the visible portions of the object are rendered.
The rotoscoped reference objects are now rendered to create an approximation of the original image that can be used in the shadow calculation and integration. Instead of an actual rendering the various components of what goes into the makeup of an image are placed in separate files. Each file created by the special renderer will be examined now.

Components of the Rendering of the Model Scene Used to Approximate the Reference Scene

If the model scene was rendered and a single image was produced, a great deal of information would be lost when the lighting calculations were combined to produce the final image. The ray tracing program written for this project was developed so that the lighting effects are not combined into a single image but are written out into individual images. This makes the analysis of the original reference image much easier.

The ray tracing program first produces a simple image where the coverage of each rotoscoped reference object is recorded. An example is shown in Figure 49. In this image, since there are three rotoscoped reference objects and a background, there are four different colors (represented by shades of gray in the image). This image is used to determine which object is visible at which pixel. Each object has a unique identification number and if, for a certain pixel, that object covers that pixel, its identification number is stored as the value in the pixel.

This image is useful in the analysis of the original image. Each object can be
Figure 49: Each rotoscoped reference object is given a number. For each pixel in this image, the rotoscoped reference object that can be seen is recorded.

accurately located on the image. Since the silhouette edge of each model object and corresponding reference object match exactly (which is explained in Chapter III), the exact extent on the image of each reference object is known. When the color of each object is calculated, this image will be used to limit the analysis to one object at a time.

The next image that the ray tracing program produces indicates which pixels have an ambient light contribution. As shown in the Figure 51, a pixel is either “on” (ambient light hits this object) or “off” (the pixel is a background pixel and ambient light isn’t considered). At this stage in the research, this image is slightly redundant as the ambient light for each non-background pixel is the same, however as
Figure 50: The ray tracing program produces an image which indicates which pixels in the original image are illuminated by ambient light.

More complicated lighting models are used this image could contain more information about the ambient lighting aspects of the scene.

Now, for each light in the scene, the rotoscoped reference objects are rendered and the intensity of light hitting each visible pixel is calculated. The color of the light is not considered at this stage, only the amount of the light reflected towards the eye at each individual pixel. This is done with ray tracing, so any shadows caused by other reference objects are included in the rendering. This rendering is used to calculate the approximate percentage of light hitting a given pixel from an individual light source. Possible specular highlights are also rendered. These are used as “hints” in the analysis of the color of a given object. If an area of an object is likely to have
a specular highlight on it, its true color may be obscured. The single image produced in this example is shown in Figure 51. The image is actually a combination of two planes: the first plane consists of the intensity of diffusely reflected light from the surface area of the object covered by the pixels, a value from 0 — 255; the second plane consists of the intensities of the specularly reflected component of the surface, again a value from 0 — 255.

Finally, the ray tracing program produces, for each light source, an image which indicates the portions of the image that are in shadow because of any new object introduced in the scene. In this example, a single image is produced, shown in Figure 52. Since a point light source was used, there is no possibility of pnueembras.
Therefore the image consists of a 0 value if the pixel can't see the light source and a value of 255 if it can. This can be modified to include information about pneumbras when a more sophisticated ray tracing program is used. There is the possibility that the new shadows will overlap shadows from the same light source already in the image. Care must be taken not to reduce the intensity of light at these pixels. This problem is solved during the image composition stage discussed later in this chapter.

4.2.4 Calculating the Color of an Object

Once the locations of all the shadows to be added to the original image are known, the original image must be analyzed so that the shadows can be incorporated correctly.

If a new shadow is to fall across an object, the intensity of the pixels that make
up the object in the image can not be simply decreased by a constant factor. If there happens to be a specular highlight in the area of the new shadow, that highlight must be removed completely. Instead of simply reducing the intensity of pixels by a constant factor, the color of the pixel must be adjusted to remove the specular component and then the intensity can be reduced to reflect only ambient light.

The first thing that must be done is to extract the color (or colors) of the reference objects that are to have new shadows on them. Calculating the colors of the soon to be shadowed objects is important for two reasons: the specular highlights on the objects must be located and the shadow intensities must be calculated. Both of these calculations can be done only if the underlying color of the object is known.

In the case of specular highlights the underlying color of the object is useful so that the specular component of any given pixel can be calculated. If a pixel on a reference object is to be placed in shadow two things must occur: the specular component of the light hitting the pixel must be removed entirely and the diffuse intensity of the pixel must be reduced by a factor corresponding to the contribution of the light source being blocked on this pixel.

With pre-existing shadow location, the color of each pixel of the reference object is important because the intensity of diffusely reflected light varies as a function of the color of the object. In the analysis of the lighting on the reference object a correlation between the intensities of the reference pixels and the rendered pixel will be derived. This correlation will vary based on the underlying color of the reference pixel. It is important to be able to group pixels with the same color in order to derive a more
accurate model of the reflectance properties of the material.

Grouping Colors in an Object

After the model scene has been rendered by the ray tracing program discussed earlier in this chapter, an image that identifies each pixel with an object is produced. Using this image all the pixels that make up a reference object can be identified. In the following discussion, we will concentrate on a single object but the procedure will be applied to each object that is to have its lighting changed.

There are certain concepts that should be reviewed before the discussion proceeds. Each pixel has a certain amount of information associated with it: its location in the image and its color specified as three intensities, [red, green, blue]. In the image files these values are traditionally stored as integers between 0 and 255. Once these values have been read in a program they are usually converted to real values between 0.0 and 1.0 for ease and accuracy during computation. By lowering each of the intensities by a uniform percentage, the brightness of the color will be reduced, but the hue of the color will remain unchanged. This concept has been used as the basis for a different color model. This model is called HSV (hue, saturation, value). In this model each color is represented by a value for the hue (the basic pure color of the object), the saturation (the tint of the color — the distance the color is from white) and the value (the brightness of the color). In this system, to simulate a shadow on a diffusely illuminated pixel, the value of the color is reduced. There are ways to convert from the RGB model to the HSV model, but in this thesis, all the work was done in RGB
color space. The terminology of hue, saturation and value will be used, however.

When an object is said to be a certain color, that really means that the object has a certain hue and saturation and that the value of the color depends on the amount of light hitting the specific part of the object. In terms of the RGB color model, this means that if you think of the color space as a coordinate system with each of the colors red, green and blue, defining an axis, an object with a single "base color" will have all its pixels fall somewhere along a line from the origin to some maximally lit color value. (This assumes that the object is a perfectly diffuse reflector. Specular reflections will change this behavior.)

It is assumed that, in general, if a reference object has many base colors, different sections of the object with the same base color have the same lighting characteristics. That is to say that if an object is colored in a red, purple and brown pattern, a red patch on one side of the object will reflect light in the same way as a red patch on the other side of the object. This will not always be the case. If an object is colored light pink on one side and dark pink on the other, the base color of both sides may be the same but the light pink side reflects light more efficiently than the dark pink side. Unfortunately, this coloration resembles a uniformly pink object with one side exposed to light and one side in shadow. If such an object is encountered it can either be modeled as two separate objects or manual control of the following shading algorithm can be taken so that correct results will occur.

If it is assumed that every pixel of the reference object of a certain base color reflects light similarly, then any variations in the intensity of these pixels must be due
to the lighting. If the surface of an object only reflected light diffusely, the magnitude of the red, green and blue components of each color on the object would be a linear function of the amount of light falling on it. For a particular base color, the red, green and blue intensities of the color would fall somewhere on a vector starting at the origin and extending to the base color illuminated with a very bright light.

This linear response of the color of an object is based on standard illumination models used in computer graphics. These models approximate the real world with enough accuracy that they can be used in the analysis of the lighting of real world objects. In David Rodger's book, "Procedural Elements for Computer Graphics" [29], a simple illumination model is described for a perfect diffusely reflecting object. Lambert's law states that the intensity of light reflected from such an object is proportional to the cosine of the angle between the light direction and the normal to the surface. Also, since the farther away an object is from the light source, the less light it receives, the reflected intensity is reduced by a factor based on the distance the object is from the light source. Additionally, the light from surrounding objects must be taken into account. This is done by approximating the complex lighting equation as a constant diffuse term to be added to the overall intensity. These relations are described in the following equation:

\[
I_d = I_a k_a + \frac{L_b e^{-\cos \theta}}{d + K} \quad 0 \leq \theta \leq \pi/2
\]

where \( I_d \) is the diffusely reflected intensity, \( I_a \) is the ambient light intensity, \( k_a \) is the ambient diffuse reflection constant \((0 \leq k_a \leq 1)\), \( I_i \) is the incident intensity from a
point light source, $k_d$ is the diffuse reflection constant ($0 \leq k_d \leq 1$), $\theta$ is the angle between the light direction and the surface normal, $d$ is the distance from the light to the object and $K$ is an arbitrary constant.

If the surface has a color, then the illumination model is applied individually to each of the three primary colors. If the base color of an object is represented as a vector $\vec{B} = [\text{red} \; \text{green} \; \text{blue}]$, then the perceived color, $\vec{P}$, is given by

$$\vec{P} = I_d \vec{B} \quad (4.2)$$

where $I_d$ is from Equation 4.1. So, the value of $I_d$ merely scales the vector $\vec{B}$.

As an example of this, Figure 44 was modified so that the cylinder is painted with three colors instead of just a single color. The surface properties of the cylinder were changed so that there would be no specular component in the reflected light. The resulting image is shown in Figure 53.

If all the pixels that make up the cylinder are examined and their colors are treated as points in the [Red, Green, Blue] color space, they can be plotted. This has been done in Figure 54. The thin line running from the origin to point $[255 \; 255 \; 255]$ is the light vector. Any pure white pixel on the object would map to somewhere on this line, assuming a white light source. If the color of the light source changes, this line will change. The light vector is placed in the plot as a reference. Since there are no white pixels on the cylinder, the colors fall around the light vector. In this case, there are three basic hues: greenish magenta (the top of the cylinder), cyan (the middle of the cylinder) and a light green (the bottom of the cylinder). These
correspond to the three lines of sample points shown on Figure 54. Because there is no specular component, there is no variation in the linear behavior of the colors. If a pixel receives more light than its neighbor, the ratios between the red, green and blue components won’t change, only the magnitude of the color will change.

However, most objects are not perfect diffuse reflectors. The color of each pixel contains a diffuse component and a specular component. Depending on the angle defined by the observer position and the light source and the normal vector of the object at the point on the surface of the object, a specular component is added to the observed color. This specular component is calculated using the following equation:
Figure 54: A plot of the colors of all the pixels that make up the cylinder in Figure 53 in color space.
\[ I_s = I_I w(i, \lambda) \cos^n \alpha \]  

(4.3)

where \( I_I \) is the intensity of the light source, \( n \) is a power that approximates the spatial distribution of the specularly reflected light, \( \alpha \) is the angle between the line of sight and the direction of reflection, \( w(i, \lambda) \) gives the ratio of the specularly reflected light to the incident light as a function of the incidence angle \( i \) and the wavelength \( \lambda \). However, since \( w(i, \lambda) \) is a complex function, it is usually replaced with a reasonable constant, \( k_z \). This means that the color of the specular reflection won't change with incidence angle. This is not the case with most substances (in particular, metals), but for simplicity this project assumed that all objects have the lighting characteristics of plastic and the color of the specular component is the color of the light no matter the incidence angle. Now, a complete illumination model for a single light source can be developed. If the color of the light source is \( \bar{L} \), the color of an object is given by:

\[ \bar{P} = I_d \bar{E} + I_s \bar{L} \]  

(4.4)

With this equation, it can be observed that the color of the object will change as the specular component increases.

Figure 53 was regenerated with the surface qualities of the cylinder such that specular highlights are created. The result is shown in Figure 55.

The specular component added by the light will change the base color of the observed color of the affected pixels. The ratios between the red, green and blue components of the color will be changed because of the addition of the specular part.
In the analysis of the color of the object it is necessary that all pixels of the same base color (even if they have a specular component) be identified. A grouping scheme based on the Hue-Saturation-Value color model was developed. In the following discussion, a single light source is assumed. The results can be extended to multiple light sources, but that will be discussed later in the thesis.

As seen in Equation 4.4, perceived colors are vectors that are the result of the addition of two vectors: the diffuse color vector \( I_d \vec{B} \) and the specular color vector \( I_s \vec{L} \). The light source vector is defined by the vector, extending from the origin to the color of the light source. If the specular component is very small, all non-white pixels will map to vectors that surround the light source vector. If the object is colored white, the colors of all the pixels will fall along the light source vector. If
the object is made of different colors, the colors of the pixels will fall along various vectors with the light source vector in the “center”. If the specular component is added, the color of each pixel will have a vector parallel to the light source vector added to it. (The size of the vector will vary based on the amount of specular light being reflected.) This specular component will make the color vector of the pixel tend towards the central light source vector. However, the color vector will always fall somewhere on the plane defined by the diffuse base color vector and the light source vector.

As an example, the colors of the cylinder in Figure 55 were plotted in color space. Since this image has a specular highlight on the top of the cylinder, one can see the effects of the addition of the specular component to the colors of the cylinder. The plot is shown in Figure 56.

There are several features of Figure 55 that should be noted. The bottom parts of the cylinder have no specular components added to them, thus the plots of those colors have not changed. The scatter plot of the colors comprising the top of the cylinder can be divided into three sections. The section labeled “A” in the image comprises the pixels that have no specular component added to them. These pixels have not changed from the previous non-specular image. The “B” section shows how, once a specular component is added, the plot bends so it now tends towards being parallel with the light source vector. Finally, the “C” section is where the pixels have been overexposed. At this point one or more of the red, green or blue components of the color is greater than the largest intensity possible for this image. (In this case,
Figure 56: A plot of the colors of all the pixels that make up the cylinder in Figure 55 in color space.
the value is 255. ) This overexposure presents problems in that some of the color information is lost and a heuristic must be developed to discover what the true color of the pixel should have been. This heuristic is discussed later in this section.

The use of computer generated images to test and explain the algorithm is useful, but digitized photographs of actual objects don't behave as nicely. To illustrate this, a full color photograph of the face of a woman floating in a pool was used. ( Figure 57. ) Using the LIVE program described in Chapter III, a model scene was generated. The main light source was assumed to be from behind the camera, off to the right. Using this information, the model scene was rendered. The results can be seen in Figure 58.

![Figure 57: A black and white image from a 24 bit color digitized photograph of a face in water.](image)

Using the model scene, the pixels that comprise only the face in the image can be identified. The results have been plotted in color space and is shown in Figure 59.
Figure 58: The model scene generated for the image in Figure 57. The main light source is near the camera, off to the right.

The light source vector has been included. Since this is an actual photograph there is a considerable amount of noise in the plot. But, the plot behaves as expected: the face in the image is dominated by the same base color of a light flesh tone. Most of the points make up the dark band that follows a line that starts from the origin and continues in a linear fashion as the intensities of the colors increase. Finally, near the top of the band, there is a slight turn towards the light source vector. This is where the small amount of specular light in the image appears.

Since pixels of the same base color, no matter the specular component, naturally fall into planes defined by the light source vector and the base color, the base colors of the reference object can be calculated. The space defined by the possible color
Figure 59: A plot of the colors of all the pixels that make up the face in Figure 57 in color space.
values and the light source vector is divided into a set of slices around the color vector. Because there will be slight color variations in objects (especially natural ones) the colors of pixels are grouped not into planes, but into three-dimensional slices. Figure 60 demonstrates this division of space.

Figure 60: In order to group the pixels of the same base color, the color space is divided into slices around the source light vector.

Since parts of the reference object that have the same base color are assumed to have the same lighting characteristics, once the pixels have been grouped in the manner described previously, the base colors of the object can be calculated. From this the pixels with a specular component can be discovered.

Each individual grouping of colors (slices) can be analyzed separately. In fact, since the differently colored parts of an object may reflect light differently, each slice
must be analyzed differently.

The number of slices that the color space is divided into is dependent on the color complexity of the object. If the object is a single color, then no slicing is needed. If the object has multiple colors more slices are necessary. The desire is to correctly group the colors while making sure that two base colors close to each other in terms of their angle around the light source vector are not grouped together. This can be done by finely slicing the color space and making a plot of the number of pixels that fall into each slice. The local minimums and maximums of the pixel counts can be calculated and the slicing of the color space can be adjusted so that the boundaries of the slice fall between groups of colors.

Once the pixels of the same base color are grouped together, the analysis of each color slice can begin. At this point we desire a model for the base color vector and with that, if a pixel doesn't map near the base color vector, it must have a specular component, which can be removed.

**Determination of Pixels with Specular Components**

Now that each pixel that comprises the reference object has been grouped into their respective color slices, the individual color slices can be analyzed. The “saturation angle” is defined as the angle formed by the light source and the color vector of a pixel. (See Figure 61.)

If the saturation angle is plotted against the magnitude of the color vector for a group of pixels in the same color slice, a plot similar to that of Figure 62 is generated.
Figure 61: The saturation angle is defined as the angle between the light source vector and the color vector of an individual pixel.
In this plot, the pixels with small magnitude color vectors all have the same saturation angle, since there is no specular component in their colors. As the magnitudes of the colors increase, the specular component is likely to increase as well. When the specular component of the pixel color is non-zero, the pixel color is moved closer to the light source vector. This causes the saturation angle to decrease.

Figure 62: A plot of the magnitude of the color vectors vs. the saturation angles of the pixels that comprise the top part of the cylinder from Figure 55.

If a color of the object is completely covered with specular reflection, the analysis of the underlying color of the object becomes much more difficult. If there are no pixels at the base saturation angle of the color, there is not enough information present in the photograph to determine the base saturation angle of the color group. This problem may be solved by using consecutive photographs of the same object if
the object is moving. The specular component would probably change as the object moves and the base color could be discovered. For this thesis project however, it was assumed that the specular highlight covering an object would take up only a small part of the surface area of the object. With this assumption, it is simple to discover the saturation angle of a color without a specular component in a color slice color grouping. The base saturation angle is the saturation angle with the highest number of pixels mapping to it.

Once the base saturation angle is known, any pixel that has a smaller saturation angle than the base saturation angle must have a specular component to its color. This specular component can be easily removed by subtracting a vector parallel to the light source from the color vector such that the resulting vector has a saturation angle that is equal to the base saturation angle.

To illustrate the removal of the specular component from the color of a pixel, refer to Figure 63. In this figure the vector \( \overrightarrow{L} \) is the light source vector, extending from the origin to \([1\ 1\ 1]\). The angle \( \alpha \) is the angle between the light source vector and the base saturation vector for this color grouping of pixels. The color of the pixel in question defines the color vector \( \overrightarrow{C} \). The angle \( \theta \) is the angle between the light source vector, \( \overrightarrow{L} \), and the pixel color vector, \( \overrightarrow{C} \). The object is to find the magnitude \( k \) that, when multiplied by the light source vector and subtracted from \( \overrightarrow{C} \), produces a new vector that has a saturation angle equal to \( \alpha \).

Using standard dot product relationships, the equation that must be solved is
Figure 63: A vector parallel to the light source is subtracted from the pixel color vector to move the color to the base saturation angle.
shown below:
\[
\frac{\vec{L} \cdot (\vec{C} - k\vec{L})}{|\vec{L}| |\vec{C} - k\vec{L}|} = \cos \alpha
\]  
(4.5)

This reduces to the following polynomial equation:
\[
k^2 |\vec{L}|^2 (1 - \cos^2 \alpha) + k^2 (\vec{L} \cdot \vec{C}) (\cos^2 \alpha - \alpha) + |\vec{C}|^2 (\cos^2 \theta - \cos^2 \alpha) = 0
\]  
(4.6)

This can be solved for the value of \(k\) with the quadratic equation. There are two solutions to the quadratic equation, since there are two points on either side of the light source vector that satisfy the base saturation angle, the value of \(k\) that has the smallest absolute value is chosen as it moves the color the least. Once \(k\) is known, the color of the pixel with the specular component removed is \(\vec{C} - k\vec{L}\).

This process is done for each color slice grouping of pixels. Once this is done, the specular component of any pixel can be removed.

Calculation of the Relation Between the Model Scene Rendered Image Intensities and Reference Image Intensities

Now, with the information on specular highlights, if a pixel is scheduled to be put into shadow, the specular component of the pixel can be removed. Once this is done, all that remains is to reduce the intensity of the diffuse component of the color to simulate a shadow. There are two questions that must be considered: “How much should we reduce the intensity of an individual pixel?” and “What if this pixel is already in shadow with respect to this light source?”

The solution to these problems lies with the images generated by the ray tracing of the model scene. Although the rendering of the model scene is only an approximation,
a relationship between the rendered model image and the reference image can be built. From this, the shadow intensity can be inferred.

Because the objects in the model scene are only approximations of the actual objects, such small scale detail as wrinkles or other small surface variations and textures are not modeled. Also, even though the properties of the light sources are known, the rendering of the model scene will not result in an exact match, with respect to light intensity, with the reference image. Because of these two factors, there will not be an exact correspondence between the reference image and the rendered image. However, because the rendered image is made from a reasonable approximation of the actual scene, there should be correspondence between the large scale effects of the lighting between the two images. This relationship can be used to discover the intensity of shadows.

When searching for the intensity of a shadow with respect to a certain light source, the pixel with the lowest intensity can not be used as the shadow intensity. What if the object in question has no parts in shadow from that light source in the original image? If that is the case, the calculated shadow intensity would be in error.

The correct method of computing the shadow intensity from the light source is to plot the actual intensities of the pixels in a color slice group against the intensities of the same pixels in the rendered model image. This will result in a plot that has an amount of error due to small scale variations of the surface of the actual object, but, in general, will describe a straight line. Such a plot of the top portion of the cylinder in Figure 55 against the rendered model scene of Figure 51 is shown in Figure 64.
This plot has been made after the specular component of all the pixels in the image had been removed. The points do not form an exact line because the rendered image doesn't exactly match the reference image in terms of lighting. In the reference image, the light source is dimmer than that used to create the rendered image, so a considerable amount of the top of the cylinder in the reference image is at the ambient intensity, while the top of the cylinder in the rendered image is illuminated from the light source. This was done to show the variations that can occur between the reference image and the rendered image, even if the exact positions of the objects and light sources are known.

![Graph](image)

Figure 64: A plot of the intensity of the pixels in the rendered model scene against the intensity of the same pixels that make up the top portion of the cylinder in Figure 55. The dotted line is calculated using a least squares fitting algorithm.

Using a least squares fit algorithm (see the book by Burden [27]), a line can
be fit to the points of the reference image intensity versus rendered magnitude plot. The magnitude of the pixels in shadow is the point where the line intersects reference magnitude axis (the y axis) of the plot.

In Figure 64, the line intersects the axis at the same intensity as the pre-existing shadows in the image. If there were no shadows in the image, the axis intersection point is a good approximation of the correct value of the shadow.

4.3 Adding Shadows to the Image

Finally, the new shadows can be added to the image. Figure 52, created by the ray tracing program, flags which areas of the reference image should be put into shadow. For each pixel on the current object that is flagged to be shadowed (the value of the shadow image is zero), the intensity of the pixel in the reference image is found. Any specular component present in the pixel is removed by subtracting a vector parallel to the light source from the color vector such that the resultant vector falls on the base saturation angle for this base color. The intensity of the non-specular color of the pixel is compared with the computed intensity of a pixel in shadow for this base color, the calculation of which was discussed in the previous section. If the intensity of the pixel from the reference image is the same as the shadow intensity, nothing is done since this pixel is already in a shadow from the light source. However, if the intensity of the pixel from the reference image is greater than the shadow intensity, the percentage difference between the two intensities is calculated. Each component of the color (red, green and blue) is multiplied by this percentage, to reduce the intensity of the pixel. This is the new color of the pixel.
The result of doing this on the sample image of Figure 55 is shown in Figure 65. Even though the image is in black and white, it is possible to see that the highlight has been removed at the appropriate point and that the new shadow integrates nicely with the pre-existing shadow. Notice, also, that the old shadow has not been dimmed at any point, even though the color changes under the area of the new shadow. This is because the shadowing algorithm behaves differently for each color of the object.

4.4 Chapter Summary

This chapter has explained the process of incorporating new shadows and objects into pre-existing images. In Chapter III, it was shown that a three-dimensional model of the objects in a photograph could be created. Using this model and information
about the location and parameters of the lights and the camera, new objects can be correctly composited into the scene. However, simply adding the object to the scene is not enough for convincing realism. The new object will change the lighting of the old objects in the scene. This must be taken into account for an accurate final image.

Using the scene information and a ray tracing renderer, an approximate image of actual scene can be created. Along with the approximate image, additional images with information about new shadows, old shadows, old object positions and lighting approximations of the old objects are created. These images are used in the analysis of the original image and the incorporation of new objects in the original image.

It was shown that the “base color(s)” of an object could be found, even if they are partially obscured by a specular highlight, by dividing the color space into slices around the light source vector. This division of the space will group similar colors with each other, even if the color is partially obscured with a specular highlight. Once the colors are grouped, the base, non-specular colors of the object can be calculated.

Knowing the base colors of an object gives the ability to remove any specular component from a given pixel. Using the approximate rendering of the object generated by the ray tracer, a correspondence between the large scale lighting of the actual image and the rendered image can be generated. With this correspondence, the intensity of shadows from the light source can be calculated.

The fully rendered, new objects can easily be composited into the original image, since, during the rendering process, only the visible parts of the new objects were rendered. Finally, the shadows from the new shadow image created by the ray trac-
ing renderer can be added by removing the specular component and reducing the intensities of the pixels covered by the shadows. The result is a realistic insertion of an object into a two-dimensional image.
CHAPTER V

IMPLEMENTATION DETAILS AND RESULTS

In Chapter III, the mathematics and algorithms of the procedures used to model a scene in a photograph were discussed. In Chapter IV, the details of compositing a new image with the reference image were discussed. In this chapter, the actual implementation and details of the entire system that was created are discussed. A complete demonstration of the system using an actual photograph is also presented.

The main program developed in this project is called LIVE. It accepts a digitized photograph and allows the user to enter various information about the scene: camera position and attributes, what objects were in the scene and their various sizes. Then, with some help from the user, it uses models of the objects in the scene to calculate approximate three-dimensional positions of the objects in the actual photograph. This is done without measuring the exact positions of each object at the time of the photograph. This means that this program can calculate the three-dimensional position of objects that are moving across the scene in successive frames of a film — something that would be impossible if you had to measure the position of the object with a yardstick for each frame of the film.

LIVE and most of the other programs in this section (unless otherwise mentioned) were written in C on HP 9000s. Because of the desire for fast graphics and to make
use of some of the powerful graphics capabilities of these machines, the programs do not run under any windowing system. In fact, a toolkit for buttons, dials and windows was developed specifically for the programs discussed in this section.

While LIVE was used to calculate the position of objects in the digitized reference scene, two different programs were created to composite any new objects into the reference image. The first program was a specialized ray tracing renderer called RAY. Not only does this program render any new objects that are to be introduced into the scene but it also renders the reference scene objects positioned by the LIVE program. These renderings are used by a program called COMP to finally composite the images together. COMP uses the rendered reference scene as a guide to control the new lighting effects added to the scene.

This chapter will discuss the details of the implementation of these programs and any limitation that they impose. A complete example will be shown to highlight the abilities of the system.

5.1 Assumptions and Required Information

LIVE has various requirements and makes certain assumptions in order to work correctly. If presented with a random image, the program will not correctly locate the objects in the photograph. It must know something about the position of the camera, lights and the objects in the scene.
5.1.1 Digitization of the Image

Each image in the film must be digitized. This alone poses certain problems. The resolution of film is much higher than most currently available scanners. Also, to digitize a single image at a high resolution requires a large amount of memory. To digitize an entire scene or film requires an incredible amount of memory. Digitization of an image at a low resolution will affect the performance of the algorithm. In Chapter III, it was shown that, depending on the distance from the object to the camera, the width of a pixel can introduce error into the calculated position of the object. There is a trade-off between the time and space that is needed for high resolution digitization of images versus the error inherent in low resolution images.

If the scene being photographed is one of a closeup of objects, a relatively low resolution digitization of the image will suffice. If, on the other hand, the scene consists of a long shot of many small objects, the higher the resolution the better.

Throughout this chapter a complete example will be developed to illustrate the capabilities of this system. A scene with simple objects was created and photographed. The photograph was then digitized using an HP ScanJet Digitizer at 75 dots per inch. Each pixel was scanned at 24 bits of color. A black and white version of the digitized reference photo is shown in Figure 66. (Because of the difficulty in reproducing color images, the full color images were not included in this thesis. They are available from the author.)
5.1.2 Measurement of the Scene

The general principle behind the successful functioning of this program is that the more information one has about the scene being filmed, the better. Every time the program has to calculate the position of an object, the possibility of error is introduced. The best case one could hope for is that every object in the scene was measured for size, shape and position and the objects never moved. If this was the case, the need for this program would not exist. A simple scene assembly program would do. In general, this kind of detailed information about a scene is not known. There is a minimal amount of information that must be known for this program to succeed: the camera location, the position of the lights and the sizes and shapes of
the objects in the scene.

Object Sizes and Models

Objects rarely change shape or size during the filming a scene for a film. They do oftentimes move, either on their own, or with respect to some camera motion. Therefore, one of the major tenets of this thesis is that it is easier and more practical to measure the sizes of all the objects in the scene once, before filming begins, than it is to measure their individual positions during filming.

For each object in the scene the three-dimensional size and shape should be recorded. A model of the object will be built using this information and the more accurate it is, the more accurate the positioning of the object will be.

The LIVE program was set up to deal with convex objects better than concave objects. This means that if a complex, concave object is in the scene, it is better to decompose the object into convex parts and assemble them into a model of a concave object. (See Figure 67.)

Instead of physically measuring the objects in the scene, some objects can be measured by making test images of the objects at different distances and angles from the camera. This is particularly useful in the case of flexible, moving objects such as people. In this case, since the camera position, lens parameters and reference object position are known, it is simple to calculate the size of the object in a similar way to the method proposed in this thesis for finding the position of objects when their size is known.
Figure 67: A complicated concave object can be easily modeled by breaking it down into convex parts and measuring them.
The LIVE program is set up so that if a reference image is shot and digitized, it can be displayed and then generic modeling objects of the same basic shape as the objects in the scene can be read in. Then, using the simple models and the knowledge of the positions of the objects in the image, the sizes of the objects can be easily approximated. (As an example, say a set of rectangles is being used to model a table. The exact positions of all the important components of the table are measured relative to the camera. Then a photograph of the table is shot, digitized and displayed using the LIVE program. Simple rectangles can be used to model the legs and the top of the table. The simple rectangles are positioned at the same position as the actual components of the table. Then using only scaling commands, the rectangles are modified so that they approximate the various parts of the table. When the various rectangles are assembled a reasonably accurate model of the table has been created.) Photographs of the object being modeled can be taken from different reference points to heighten the accuracy and also make sure that all parts of the object are modeled. This method of sizing a model object is particularly useful for models of people. People are difficult to measure, but by shooting reference photographs from many angles and at different distances, it is possible to arrive at an accurate model of a person.

Position and Attributes of Lights

By knowing where the lights are and the intensity of the lights at various points in the scene, they can be more accurately modeled. Again, the more detailed knowledge
of the position and attributes of the lights is known, the better a match the lighting
effects introduced to the image will be. The position relative to the camera should be
measured. The size of the light and direction the light is pointing should be measured.
The intensity and color of the light should be measured.

The measurement of most of the parameters for the lights is straightforward —
except the color and intensity of the lights. The method used in this project is to
photograph a sheet of colorbars at different distances from the light. From these
photographs the color and intensity of the light can be modeled. This also has the
advantage of modeling the response of the film stock to the light as well. As long as
the same stock of film and the same lights are used, the parameters for the intensity
and color of a light will remain the same throughout shot.

In the paper by Cook and Torrance [8] an equation is defined for the reflected
intensity of light off an object based on the qualities of the object and the intensity
and solid angle of the light source:

\[ I_r = I_{ia}R_a + I_i (\cos \alpha)d\omega_i (sR_s + dR_d) \]  

(5.1)

Table 1 shows an explanation of all the symbols in the preceding equation.

The task here is to calculate the ambient intensity (\( I_{ia} \)) and the average intensity
(\( I_i \)) based on the recorded values of the reflected light (\( I_r \)). To make the calculations
simpler and more accurate, it is assumed that the colorbars being photographed are
held so that the surface normal and the normal in the direction of the light are
parallel. Also, if the colorbars are photographed so that the light is directly behind
the camera, the \( \cos \alpha \) term reduces to 1. It is also assumed that the colorbars are
made of a very non-shiny material so that there will be no specular reflection with the camera and lights oriented this way. In that case it can be assumed that \( d = 1 \) and \( s = 0 \). Using these assumptions Equation 5.1 becomes:

\[
I_r = I_{ia}R_a + I_i R_d d\omega_i
\]

In Cook and Torrence's paper, they mention that the reflectance spectra (\( R_d \) and \( R_s \)) for thousands of materials have been measured and collected. By looking up the diffuse bidirectional reflectance for the material that the colorbars are made of, the equation only has two unknowns. (It is also mentioned in the paper that \( R_a \) can be approximated by \( \pi R_d \).) The solid angle made by the light (\( d\omega_i \)) can be calculated if one knows the dimensions of the light emitting surface and the distance the colorbars are from that surface. By photographing and digitizing the colorbars at two different positions it is possible to solve for \( I_{ia} \) and \( I_i \) for the light. One begins
with the following two equations:

\[ I'_r = I_{ia} \pi R_d + I_i R_d \omega'_i \]
\[ I''_r = I_{ia} \pi R_d + I_i R_d \omega''_i \]

(5.3)

Solving the preceding equations for \( I_i \) and \( I_{ia} \) yields:

\[ I_{ia} = \frac{I'_r \omega''_i - I''_r \omega'_i}{\pi R_d (\omega''_i - \omega'_i)} \]
\[ I_i = \frac{I''_r - I'_r}{R_d (\omega''_i - \omega'_i)} \]

(5.4)

(5.5)

These two equations can be applied to all three colors; red, green and blue, to come up with a complete description of the direct intensity of the light, along with the ambient component that light contributes to the scene.

**Camera Location**

In still photography it should be possible to measure the exact location of the camera for each shot. However in motion picture filmmaking, this becomes more difficult. If the camera doesn’t move during the scene, there isn’t a problem. However, if the camera pans, trucks or zooms, its position (or lens setting in the case of zooms) will be changing from frame to frame. In this case motion control systems are used routinely in film-making when shooting special effects. They are used to record the position of the camera and models in the scene so that the position of the camera and the models throughout the shot can be repeated.
This way many elements of the same shot can be photographed in the same way. This technology is perfect for recording the information this project requires.

**Example Scene Measurements**

In the example scene, all the information mentioned in the previous sections was measured and the appropriate modeling objects were built. The sizes of the objects and the locations of the lights are listed below:

- All measurements were done in inches assuming a coordinate system with an origin on the floor below the camera.

- The **camera position** was at \([0, 25.5, 0]\).

- The **center of interest** was at \([-1.0, 0, 28.5]\).

- The **light position** was at \([42, 38.5, 17]\). The light was a standard 100 watt bare bulb, but for this experiment it was assumed to be a point light source for this experiment.

- The **cookie tin** was measured to be 4.625 in. tall and 4 in. in diameter.

- The **book** was 9 in. tall, 6.25 in. wide and 1 in. thick.

- The **ball** was 2.5 in. in diameter. The ball was fairly malleable, so the size could be in error by 0.25 in.

- A **spray paint can** was inserted as a control. (See Figure 98.) The spray can was measured to be 7.75 in. tall and 2.5 in. in diameter.
5.2 Edge Detection

Once the image has been digitized, the edges of each object in the scene must be located. Because of its simplicity and acceptable results, Shiozaki's edge extraction algorithm using an entropy operator [35] was implemented. (A discussion of this and other edge detection algorithms can be found in Chapter III, Section 3.1.1.) This algorithm was developed with color pictures in mind and was well suited for the color images necessary for this thesis. The implementation of Shiozaki's edge detector was written in C and accepted a Run-Length Encoded color image. It returned a black and white Run-Length Encoded image where white pixels were considered edges and black pixels were non-edges.

![Image](image_url)

Figure 68: The results of finding the edges of the example image

The result of executing the edge detection program on the example photograph
of this chapter is shown in Figure 68.

5.3 Edge Connection

In Chapter III, Section 3.1.2 the basic concepts of edge connection were discussed. In this section the actual algorithm used will be presented. As an overview, the algorithm repeatedly follows these steps: (After an initial cleanup of the image.)

- A large, empty area of the edge detected image is found.

- Boundary pixels of the area are located and marked.

- The boundary pixels are traversed in one direction and certain types of gaps are filled.

- The boundary pixels are traversed in the opposite direction and the rest of the gaps are filled.

- The area is marked and the process is repeated until all areas of the image have been marked.

5.3.1 Removal of Lone Pixels from the Edge Detected Image

The edge detection image must be prepared before the edge connection can begin. Edge detections algorithms have a tendency to misinterpret small, localized variations in the texture or color of the original image as edges. When this happens, a single pixel is marked as an edge. These small aberrations must be removed because, even if
they mark a very faint edge, a single pixel with no neighbors doesn't contain enough information about the edge to be of use.

A filter is used on the image to remove extraneous edge points. These are pixels that the algorithm thought might be edges but don't have any neighbors. (See Figure 69) A simple program is used that examines each pixel and removes those with no neighbors.

5.3.2 Locating an Area in the Image

After the lone pixels have been removed from the edge detected image, the edge connection can begin in earnest. The first step in the iterative process is to locate a large area with no edges in it. Initially, a random non-edge point was picked and the search started from there. However if the random non-edge point was in the middle of a small area, the algorithm would edge connect that small area and, as a result,
Figure 70: An example of a seed point picked in a large area of the image.

produce more small areas. It turns out that the algorithm works best if it starts with
the large areas of the image and works its way to the small areas. If it is done the
other way (from small to large) the algorithm produces a lace-ridden image.

In order to decide what is a “large” area, each non-edge pixel of the image is
examined. The largest rectangle that contains that pixel and no edge pixels is found
and recorded. A histogram of the sizes of the rectangles is produced. The pixel that
had the largest rectangle is chosen as the starting point. If there are multiple pixels
that map to the same large rectangle(s), the pixel is picked randomly. An example
of this is shown in Figure 70.
5.3.3 Centering the Seed Point in the Area

The pixel chosen in the previous section will be referred to the *seed point* as it is from that pixel that the search for gaps will begin. In order for the search for the gaps to perform well, it is necessary for the seed point to be located in the center of the area. If the seed point is near an edge the search method described below will terminate prematurely in some cases.

The largest rectangle that contains the seed point and doesn't contain any edge pixels is found. The seed point is then moved to the center of that rectangle. This is shown in Figure 71.
Figure 72: After the center of an area has been discovered, an approximately radial pattern of probes is sent out.
5.3.4 Radial Search from the Seed Point

At this point a conceptual balloon is inflated starting at the seed point. The balloon continues to inflate until something stops it in the direction of the inflation.

To do this a radial fill is done starting at the seed point. (Figure 72) The fill progresses outward from the seed point at a radius of one pixel at a time. Whenever the fill encounters an edge pixel, that part of the fill is stopped. When all parts of the fill are stopped, either by an edge pixel or the edge of the image, the fill is complete. The edge pixels that stopped the fill are recorded.

5.3.5 Creation of Ordered List of Boundary Pixels

The radial fill has located all the edge pixels that border the area that contains the seed point. This method of finding the boundary of the seed point area has the advantage of not relying on the edge detection lines themselves. As mentioned previously, the lines that edge detection algorithms produce are not well-behaved lines. They are thick and spotty. If, on the other hand, the boundary of the seed point area was found by tracing the edge detection lines, the problem becomes much more difficult in that there is no guaranteed way of determining which way a line is headed. This also poses a problem when trying to fill the gaps. If a gap is found in a line, the most probable direction to search for a point to close the gap is in the direction of the line where the break occurred. If it is difficult to determine that direction, the search for the other end of the gap is that much more difficult and inaccurate.
Figure 73: The edges hit by the radial probe are given numbers to establish a direction around the area.
Since the radial fill has located the single pixels that have stopped the fill, there is no problem with thick edge detected lines. The boundary between the area and the edge detection line will be used to determine direction.

An ordered list of pixels is built from the boundary pixels. A number is assigned to each boundary pixel to determine direction around the border of the area. The results of this step are shown in Figure 73.

![Figure 73 and 74: Gaps in the Boundary](#)

**Figure 74:** A closeup of Figure 73 showing the difference between the types of gaps that might occur.

### 5.3.6 Location and Filling of Gaps in the Boundary

The list of pixels is traversed. If the center of some pixel, \( P_i \), is more than \( \sqrt{2} \) units away from the center of its neighboring pixel, \( P_{i+1} \), a gap has occurred. If the distance is less than \( \sqrt{2} \) units the pixels are immediate neighbors and no action need be taken.

Once a gap in the boundary of the area has been found, it must be filled. The direction of the traversal of the list becomes important at this point. There are two possibilities now: (Case 1) \( P_{i+1} \) is closer to the center of the area than \( P_i \) or
(Case 2) $P_{i+1}$ is farther from the center of the area than $P_i$. The gap type described as Case 1 is more difficult to correctly connect than the Case 2 gap type. (The reasons for this will be described momentarily.) However, if the numbering of the list of pixels is reversed, gaps reverse their type as well. Because of this, the algorithm connects the Case 2 gaps and then the process is repeated in reverse to connect the rest of the gaps, that, in a reverse traversal of the list, are now Case 2 type gaps.

Figure 75: A closeup of Figure 73. When a gap is encountered, a probe line is sent based on the slope of the pixels up to the gap.

As the pixel list is traversed, anytime a gap is encountered it is immediately identified as either a Case 1 or Case 2 type gap. If it is a Case 2 gap, steps are taken to connect it. If it is a Case 1 gap, its connection is postponed until it is encountered in the reverse traversal of the pixel list.

Once a Case 2 gap has been identified, a line in the direction of the edge is calculated. To do this, the slope of the line is calculated by examining the previous five pixels in the pixel list. A line is fit to these five pixels using by fixing the
endpoint of the line at the terminal pixel and calculating the slope that results in the least amount of distance from the rest of the pixels to the resulting line. This is illustrated in Figure 75. Tests were done with sample sizes of greater than five pixels and less than five pixels. More than five pixels in the line calculation resulted in lines that were not sensitive to local changes in direction. Less than five samples created lines that were too sensitive to local perturbations in a line.

This projected line from the pixel list is now extended into the gap. This is the direction that the list of pixels would travel, assuming that it did not curve in any direction. The pixels on the pixel list with indices higher than that of pixel $P_i$ are now examined. The first pixel closer to the seed point than the projected line is chosen as the point at which to close the gap, $P_j$.

![Figure 76: A line is traced between the endpoints of the gap and new pixels are added.](image)

Using Bresenham's line drawing algorithm [30], a straight line is drawn between the two endpoints, $P_i$ and $P_j$. Any new pixels added to the image are now considered valid edge pixels, and will be used as area boundaries just like the original edge.
detection pixels. An example of this connection process is shown in Figure 76.

Figure 77: The original image with the results of a clockwise gap fill.

This process of finding the gaps and connecting them is repeated until the entire list of pixels is traversed in one direction. This will connect all the Case 2 type gaps in the pixel list. (See Figure 77.) After the list is traversed, the area is refilled and the boundary pixels are recomputed — the addition of new pixels to the image makes this necessary. The new list is now traversed in the opposite direction from the first traversal. This will complete the connection of all the gaps in this pixel list.

Finally, all the pixels in the area that was just connected are marked as having been visited to insure that no point in this area is considered again.
The process of finding an area, connecting its boundaries and marking the pixels continues until all the areas have been visited. The areas are filled beginning with the large areas moving to the small areas. This creates the fewest incorrect connections since the boundaries of small areas tend to curve a lot. The boundaries of large areas tend to be much more well behaved and predicting the closure of the gaps is more accurate.

5.3.7 Problems with Edge Connection

The algorithm performs remarkably well, but there are some problems. Most of them stem from the process of finding a point closer to the center of the area than the imaginary line; some areas just don’t lend themselves to that solution. However, in most cases the gap connections are what would be expected. The algorithm tends to added more lines than it should, which is acceptable, since extra lines do not affect the performance of the alignment program too much. The only drawback to extra edges introduced to the image is that the model object may have to be placed closer to the reference object by the human operator.

Also, because of the backtracking nature of the algorithm — once new edges have been added to the edge detection image, the areas that the boundaries define are constantly changing and must be recomputed at various intervals — the algorithm was slow. To remedy this, it was decided to implement the algorithm in parallel, without changing any of its functionality.
5.3.8 Results of Edge Connection on the Example Image

The edge connection algorithm was executed on the example image of Figure 68 and the results are shown in Figure 78. While the example image had fairly well defined edges, there were some gaps that needed to be filled. This algorithm has attempted to fill them. The area between the cookie tin and book was filled with shadows and did not produce many edges. The algorithm added more edges that attempted to connect the appropriate gaps.

5.3.9 Parallel Implementation of Edge Connection

To parallelize the edge connection algorithm, it is not possible to simply distribute the area connection tasks over the available processors. One processor might connect
a line that conflicts with the solution of another area by another processor. The solution to a connection task is affected by the area’s neighbors.

While it is impossible to calculate the exact changes to the image that the connection of an area would cause, it is possible to create a bounding box, outside of which, no changes will occur. This is done during the initial fill stage of the algorithm where the boundaries of areas are calculated. Once the incomplete edges surrounding the area have been identified a box surrounding those edges can be created. Because the new lines generated to connect gaps are linear and extend only from one identified edge to another identified edge, the new edges introduced will not extend beyond the box.

Because of the need to start connecting large areas and then work down to the smaller areas, a list of bounding boxes that are larger than some limit (30^2 pixels proved to be reasonable in tests of 320 by 240 pixel images. A square this size takes up about 1 percent of the image area. Bigger squares were not necessary to prevent the “lace” effect from happening. ) is created in parallel. At this point in the algorithm, there is no need to check for intersection between the bounding boxes — that would restrict too many possibilities.

After the list of areas of a certain size was built, the areas could be distributed to the available processors for calculation. Once an area was given to a processor, it was placed on an active area list. This list keeps track of the areas being computed and prevents areas that intersect the current areas from being started. In this way the processors are insured of only simultaneously working on areas that can not possibly
IN INITIALIZE image array, lists and shared variables.
SCATTER arrays over memory to reduce contention.
READ image from file into image array.

FOR size = 30 DOWNTO 5 STEP 5 DO
BEGIN
IN PARALLEL FOR each pixel, \( P_i \), in the image array DO
BEGIN
IF \( P_i \) is edge THEN
BEGIN
IF \( P_i \) is not included in one of the areas in area list THEN
BEGIN
CALCULATE size of maximum edge-free area, \( A_i \), that contains \( P_i \)
IF \( A_i \) is the size \* size THEN
BEGIN
LOCK write access to area list
IF \( A_i \) on the area list already THEN UNLOCK and terminate process
ADD \( A_i \) to area list
UNLOCK write access to area list
END
END
END
END
END
END

IN PARALLEL FOR each area, \( A_j \), on the area list DO
BEGIN
CALCULATE center point of \( A_j \)
IF \( A_j \) is the size \* size THEN \{ the size of the \( A_j \) may have changed. \}
BEGIN
C: WAIT until the bounding box of \( A_j \) doesn't intersect any area on the bounding box list
IF part of \( A_j \) hasn't been filled by another process THEN
BEGIN
CALCULATE surrounding edges and bounding box for \( A_j \)
LOCK write access to bounding box list
IF bounding box of \( A_j \) doesn't intersect bounding boxes on the list
THEN push \( A_j \) on the list
ELSE UNLOCK list and GOTO (C)
UNLOCK write access to bounding box list
RECOMPUTE surrounding edges of \( A_j \)
CONNECT edges clockwise and anti-clockwise
FILL connected area
LOCK write access to bounding box list
DELETE bounding box from list
UNLOCK access to bounding box list
END
END
END
END

CONNECT all remaining areas, regardless of size
WRITE the completed image.

Figure 79: Parallel Edge Connection Algorithm
affect one another.

Eventually, all the areas on the bounding box list are processed. The image must be re-analyzed for areas and a new bounding box list is created, but this time the limit on the size of acceptable areas is reduced. The process of locating areas and connecting areas is continued until all the areas are filled. The entire algorithm is presented in Figure 79.

This algorithm was implemented on a Butterfly Multiprocessor with 8 nodes. The Butterfly is a shared memory multiprocessor with a simple locking mechanism for shared variables. It consists of two commands: Lock <variable> and Unlock <variable>. With these commands it was possible to make sure that the writing to the active and bounding box lists was done as an atomic operation.

The performance of the algorithm varied with the size of the bounding boxes. At the beginning of the algorithm, the bounding boxes are large and there is a big chance that a single bounding box will intersect all the other bounding boxes on the list. In this case, only one processor can work, while the rest wait for it to finish. The parallelism of the algorithm is very low at this point. However, as the bounding boxes get smaller, the chance that they will intersect each other is reduced. More and more processors can run at the same time.

5.4 LIVE: An Object Positioning Program

This section will describe the implementation of LIVE, a three-dimensional rotoscoping program. After a brief description of the program, each of the major design decisions and features of the program will be discussed.
In traditional rotoscoping, the animator sits underneath a film projector and a film is projected, one frame at a time, onto sheets of paper. For each frame, the animator traces the outlines of the objects in the image. After this is done, the animator has a complete record of the film on paper and is free to use that information to incorporate new animation or live action into the scene.

The traditional rotoscoping setup was used as a basis for the LIVE program. The animator is presented with a digitized frame of films, in sequence, and (instead of tracing the outlines) places models of the objects near the actual objects in the scene. Because the alignment of the objects is tedious, the computer then aligns the model objects to the outlines of the actual objects in the film automatically. Also, because the model objects were generated so that they were the same size as the actual objects in the frame, the three-dimensional positions of the actual objects in the original scene can be computed. This information, like the outlines of the objects in traditional animation, is used to composite new things into the image.

Since the digitized frames of film will be presented in sequence to the animator, the program LIVE also behaves as an animation tool. If an object is not moving in the scene, there is no need to place it for each frame. Also, if an object is moving in an easily predictable fashion, the object can be positioned at the starting frame and the ending frame of the motion and the position of the object for the intervening frames can be interpolated. While this is not completely accurate, it is a time saving device for the rotoscoping animator.

LIVE has tools to ensure that, once the model objects are aligned to the reference
objects in the image, the outlines of the two objects match exactly. This is necessary for accurate image composition.

![Figure 80: The LIVE object positioning and alignment environment.](image)

In summary, when the rotoscoping technician starts the program he or she is presented with a complete object manipulation environment. It allows the technician to view an image or sequence of images, place three-dimensional model objects on the image and finally fit the objects to the actual objects in the scene. The environment, with the example image for this chapter visible, is shown in Figure 80.
5.4.1 Explanation of HP 9000 Environment

The large amount of user interaction with the LIVE program made the choice of the platform on which to develop it important. The Hewlett Packard HP 9000 series of computers were used because of the color displays and numerous simplifying features that they have.

There is a considerable amount of information that must be displayed at the same time in the same place for this implementation to be useful. In order to make the information usable to the technician running the program, a color display was deemed necessary. The HP98721 SRX has a 24 bit plane display with a 3 bit plan overlay. With this machine it is possible to display a 24 bit image on the lower planes and manipulate multicolored wireframes in the overlay plane without the overhead keeping track of image overwrites. The HP98550 is a lesser model than the HP98721 with only 8 bit planes for display and 2 bit planes for overlay, however this is enough display capability to manage the problem of displaying all the needed data. The Graphics Lab at the Ohio State University, where this program was developed, has two HP98721 machines and eight HP98550 machines.

As all the machines have a mouse and a keyboard it was decided that no special input device would be used to control the positioning of the model objects.

The program, at this stage will only run on these machines. It would be very difficult to modify it so that it would be able to display all the needed information on a black and white display.

There was a considerable need for speed in the manipulation of the wireframe
model objects if this program was to usable. The overhead in speed associated with
the X windowing system installed on the HP machines was unacceptable. It was
decided to write the program without the aid of a windowing system. However, the
buttons and sliders and other graphical user interface tools that are provided with
windowing systems would be missed in a program such as this. To compensate for
this, a minimal graphical interface tool set was developed for the windowless machines.
It consisted of the following objects: a button, a slider, a text input window and a
text display window. Each of these objects had various operations that could be
performed on them (place, hide, show, input...) and were placed in a library that
the LIVE program could use.

5.4.2 Display of Images

There should not be a limit to the size of the digitized image. There are often
space versus resolution problems that must be addressed when the frames of a film
are digitized. The higher the resolution of the digitization the better, but higher
resolution requires more memory to store. In this project, limitations on the size of
the image were imposed. Because most television images are approximately 640 pixels
wide and 480 pixels tall, an image of that size was taken to be the largest displayable.

In the image display area of the screen, the 640x480 image is displayed in an area
that is 900x600 pixels wide. The extra display area around the actual image is needed
for objects that extend beyond the boundary of the frame. It is very helpful to be
able to see the complete object, even if only a section of it is in the frame.

The image is displayed in either the 24 bit plane area of the HP9721 or the 8 bit
plane area of the HP98550. The buttons, sliders and other tools are also displayed in this area too. All the wire frame model objects are displayed in the 3 ( or 2 ) bit overlay planes of the machines.

![Image](image.png)

Figure 81: In the LIVE environment it is possible to display the original image and the detected edges.

The image produced by the edge detection and connection algorithms is read in at the same time as the digitized image. This can be examined by displaying it in the overlay planes of the screen. The original image is beneath it and it is possible to see exactly how well each edge of the image was calculated. With this it is possible for the rotoscoping technician to compensate for extraneous edges in the edge detection image. The example image is shown in this fashion in Figure 81.
5.4.3 Placing the Model Objects

After a digitized frame of film is displayed in the image display area, the rotoscoping technician must locate the objects in the scene by placing computer generated model objects near the actual objects they represent.

Before the technician can do this the model objects must be created. When the scene is first filmed, all the shapes and sizes of all the significant objects in the scene must be recorded. Then, using standard three-dimensional object generation software, relatively accurate three-dimensional models of the objects in the scene are created. The objects are created using the exact sizes of the actual objects. It is not enough that the model objects have the same proportions as the reference objects, the sizes of the objects must match for the three-dimensional positioning to work correctly.

These three-dimensional model objects can be loaded into the program and positioned by the operator using the translate and rotate controls on the right of the screen. There are scaling controls on the right, but it imperative that the technician does not use them at this stage. If the size of the modeling object is changed to make it fit better, the calculation of the three-dimensional location of the object will be in error.

For the example of this chapter, a cylinder has been placed near the cookie tin in the scene. This is shown in Figure 82.

Also, before any objects can be placed, the camera position must be entered and the lens of the camera set. On the lower left hand of the display there are controls
to set the camera location, its center of interest and its viewing angle. With this, the camera position and lens are modeled. In this project, all experiments have been done with a stationary camera. If the camera is moving during the scene, the camera position and center of interest can be read from a motion control position recorder and read into the program for each frame.

Limitations on Modeling Objects

Ideally, there would be no restrictions on the kinds of three-dimensional modeling objects used to extract information from the scene. However, for ease of programming in the proof of the concept, certain concessions were made. All modeling objects were represented as polygonal meshes. Curves, splines and other techniques for ob-
ject representation were not allowed. There are tools that allow conversion between different representations of objects. Additionally, the modeling objects were required to be triangularized objects. This allowed for certain optimizations to be made in the display of the objects and some of the alignment calculations. This is not an extreme restriction since the triangularization of a polygonal mesh is not difficult and tools exist for just that.

Again, for proof of concept, the program can only accurately deal with convex objects. If a concave object exists in the scene, it must be modeled by attaching two convex object together. There is no added difficulty in calculating the position concave objects, only in calculating their silhouette edges.

5.4.4 Matching Silhouette Edges of the Objects

Once the rotoscoping technician has placed the three-dimensional model object so that it is "near" (in both position and size) to the reference object it is to model, the program aligns the model object with the reference object. The first step is the calculation of the silhouette edges of the model object. With the knowledge of the eye position and the normal from each polygon of the object, the silhouette edge is defined as the shared edges of polygons that change from having a normal that points in the direction of the eye position to a polygon with a normal pointing away from the light position. The boundaries between such pairs of polygons are calculated and saved as the silhouette list.

The silhouettes of the reference objects have already been calculated during the edge detection and connection pre-processing step described earlier in this chapter.
Figure 83: The model object cylinder has probes extended from its silhouette edges. They stop when they encounter a edge detection pixel.

The Probing Process

The silhouette list is traversed and the midpoint of each edge in world space is computed. This midpoint is transformed into perspective space. The two-dimensional normal of the edge projected onto the viewscreen is calculated. Two vectors are extended from the perspective transformed midpoint in the direction of the two normals of the edge. These vectors are referred to as "probes". They are looking for an reference image edge to match with the model silhouette edge. A Bresenham's line drawing algorithm is used to calculate the path of the two lines. The first probe to encounter an edge pixel in the reference image is used as the matching probe. (It is assumed that the model object has been placed close to the outline of the actual
object and that the nearest edge to the model edge is the correct edge. This is not always the case and can produce errors. If the first edge pixel encountered is the edge of the image it is disregarded. If both edges encounter the edge of the image as their first edge pixels, this model edge is marked as being unusable for position calculation. The probing process for the cylinder model object in the example image is shown in Figure 83.

Once one of the probes extended from a silhouette edge encounters a reference image edge pixel, the position of the edge pixel and the distance from the midpoint are recorded for this model silhouette edge. This information will be used later to calculate the distance to move the object. This probe/record process is done for each silhouette edge of the object.

In the program LIVE, this is done for each model object. It is initiated by pressing the “probe” button in the center set of buttons. The probe lines are extended from the midpoints of the object and are visible on the display. If the probe lines misidentify more than a few of the edges, the object should be repositioned and probed again.

**Objects Covered by Other Objects**

A common occurrence in images is that one object might obscure another object or part of another object by being closer to the camera than the other object. It is also possible that the two objects might intersect one another. This is actually quite common in the case of concave reference objects modeled with intersecting convex
modeling objects. In either case there will be some loss of model object positioning information.

In the case of one reference object being in front of another, the closer object will have all of its silhouette edge information visible. The farther object will not. When this is the case, the closer object is probed normally, but the only the visible edges of the farther object are probed. The edges of the farther object that are clipped by the closer object, yet still are partially visible, can still be used. The point of the clipping is calculated and the midpoint of the shortened edge is used. When computing the final position for the farther model object, only a few of all the possible edges will be usable in the calculation. The algorithm tries to move the model object as little as possible for the edges to match.

If the two objects intersect, there is loss of information for each object. If, after the initial placement of the model objects by the rotoscoping technician, two of the model objects intersect, the completely obscured overlapping edges on both objects are removed from the probing process. In effect only the silhouette edges around the combination of the objects are used, even if the modeling objects are not attached to each other and are free to move independently. In Figure 84 a rod intersects a cube. Just as in the case with non-intersecting objects, the endpoints of the clipped edges (an example of a clipped edge is marked “Shortened Edge” in the figure) are recalculated and the midpoints of the edges are computed using these new points.

The midpoint probes are not the vectors that are used to calculate the three-dimensional position of the model object, rather corner vectors are calculated using
Figure 84: An example of two intersecting objects. The black outline edges are the edge used in the alignment process.
neighboring midpoint probes as described in Chapter III, Section 3.3.3. Unfortu-
nately, in the case of intersecting or obscuring objects, certain corners may be ob-
scured. It is even possible to have an edge's middle obscured while its endpoints are
visible. As long as both the edges of a corner are visibly connected to the corner the
corner vector can be computer and used in the positioning of the object. All other
cases cause the corner to be dropped from the positioning calculation.

These limitations also apply to objects that extend beyond the boundaries of the
photograph. The viewing area on the display is larger than necessary to accommodate
such objects, however it is possible to have a larger object where only one edge or
segment of an edge is in the image. In this case there is not enough information for
the computer to position the object and the object must be positioned by hand from
measurements taken at the scene.

Subdivision of the Object

Initially, for quick response during the interactive positioning phase and for ease of
generation, the less complex the object the better. If an object is composed of only
a few polygons, it is easier to model and is quicker to draw on the screen. However,
if the silhouette edge list of the simple object is small and one or two of the probes
sent from the edges match to the wrong pixel, the minimization techniques discussed
in Chapter III will not work very well and there is a high degree for error. However,
if more probes are cast from the silhouette edges and a greater percentage of these
are correct than the the few that are in error, the minimization procedure will have
a greater chance of finding a more correct solution. So, a feature was added to the LIVE program that allows the user to subdivide the existing polygons. This has the effect of producing more probes and more corner adjustment vectors since there are now more edges and more corners.

This subdivision must be used with care however, if the object is subdivided too much the chances that the midpoint probes of the some of the edges near the endpoints of the original un-subdivided edge will miss their intended edges entirely. By probing in the middle of a long edge, it is hoped that it will have the best chance of hitting its matching edge.

Also, later, after the object has been positioned, the object description will be deformed so that the silhouette edges of the model and reference objects match. The subdivision of an object can happen at this stage to make the object more “flexible.” By having more vertices to move, the object can more completely match the silhouette of the actual object.

The Calculation of the Object Translation

Once the probes have been sent the rotoscoping technician activates the positioning button and the program does the calculations described in Chapter III, Section 3.3.3. The display reflects the results of the calculations. The result of the translation of the cylindrical model object of Figure 83 is shown in Figure 85.
Figure 85: The cylindrical model object is translated to a point where the silhouette edges match based on the silhouette probes.

Iteratively Repeat the Process

After the initial placement of the object by the technician there is a large possibility for error. The model object edges may be far enough away from the reference object so that, in some cases, the wrong edges are found as matches for the model object edges. In this case, the minimization positioning part of the program will likely move the object closer to the correct position (assuming most of the probes found the correct edge) but the object will not be in its optimal position. This is unavoidable since the program has no real knowledge of where the object is meant to be.

The solution to this is to repeat the process over again. The modeling object will be closer to the reference object and the next application of the process should
result in even better positioning. The probing/moving of the object can continue indefinitely, although eventually the object will settle on a few positions that it will move through repetitively. There will very rarely be an exact solution because of the inaccuracies of modeling the reference objects. The more accurate the model object, the smaller the error in the final positioning of the object. This error is acceptable since the outlines of the objects will be matched exactly and the slight mis-positioning of the object should not be noticeable in the shadows and lighting effects added to the image.

\subsection{5.4.5 The Rotation Problem}

In Chapter III, Section 3.3.4 the automatic orientation with respect to rotation of a model object was discussed. It was shown that it was possible, under certain circumstances, for the computer to rotationally orient the model object given that the rotoscoping technician identifies numerous reference points on both objects. This orientation is subject to the same errors as that of the positional orientation of the object. Also, it is not necessary that the technician use the rotational orientation and only orients the object by hand. In either case there will probably errors in the final position of the model object.

The errors involved in the misrotation of an object are unavoidable and to some extent acceptable. The three-dimensional location of the object is calculated by using the overall size of the object and not the minute details. A small error in the rotation of the model object will not result in an unacceptable error in the final positioning of the object. Also, when calculating the shadows that will be added to
the reference object, the model object is only an approximation in the first place. The interior details of the reference object are not guaranteed to match those of the model object. The overall shape of the shadow will be correct and the interior details will be compensated for in the composition phase of the integration.

What is unacceptable about a misrotated model object is that the silhouette edges of the two objects will not be exactly aligned. If this is the case, any lighting effects added to the reference objects will either extend beyond the boundaries of the objects or will not cover all area on the object that should be covered. This problem must be solved before the image composition can continue.

5.4.6 The Solution to the Rotation and Silhouette Edge Problems

The problem of misaligned silhouette edges can be solved if the modeling object is allowed to deform around the reference object. This is an acceptable option since the modeling object, in most cases, will be much less detailed than the reference object. Because of this, slight variations applied to the modeling object will not effect the quality of the resulting image by any discernible amount. In fact, the deformation is likely to add complexity to the modeling object. This complexity will be the result of the details of the reference object and as such, will reflect the details of the reference object and add to the realism of the added lighting effects.

To deform the model object, the silhouette edges of the model object will first be moved so that they are aligned exactly with the reference object's edge detected silhouette edges. Then, the deformation will be applied to the entire object to make
sure that the deformations will not look to incongruous.

Figure 86: The silhouette edges of cylinder are “scrunched” so that they more accurately align with the reference object’s silhouette edges.

Scrunching the Object

Once the object has been placed, the technician usually will press the “Scrunch” button. This button causes all the vertices that lie on the silhouette edge of the model object to find the closest edge detected pixel and move in three space so that they map to that point. The move is computed so that they move the smallest distance from their starting point. This calculation is described in Chapter III, page 64. The result of applying “scrunching” to the cylinder in the example image is shown in Figure 86.

If the model object is too coarsely defined for a good match between the silhouette
edges of the two objects — the vertices of one of the model object’s silhouette edges may lie on edge detected pixels, but because the non-linearity of the reference object’s silhouette edge, the interior of the model object’s silhouette edge might not completely match the reference object silhouette edge — the model object description can be subdivided, using the “Divide” button. This will provide the model object with more of an ability to follow the curves in the reference object’s silhouette edge. The scrunching algorithm keeps track of all the matched pixels so that no more than one vertex can match one pixel. In this way, by repeatedly subdividing the model object, every pixel can have a vertex assigned to it. Of course, this would result in a terribly complex object, and in most cases, this option does not have to be exercised, but it does show that the edges can be matched exactly. An example of this process is shown in Figure 87. In the figure only the subdivision of one edge is shown for simplicity, in the actual case, all edges would be be equally subdivided.

Figure 87 also illustrates another problem with this algorithm: If the vertex of a model object silhouette edge is not placed so that it exactly matches the corresponding corner of the reference object, a considerable amount of complexity must be added to the model object to get the edges to match. In Figure 87 the object must be subdivided and matched four times for the two silhouette edges to match. (The last subdivision is not shown.) If the vertex shared by Edge 1 and Edge 2 were moved to the pixel marked by the star, no subdivision would be necessary and the edges would match exactly. Because similar cases involving corners occur fairly often, the rotoscoping technician was given the ability to pick any vertex of a model object and
Figure 87: Model object silhouette edges are subdivided and matched with the reference object silhouette edges.
move it to a given pixel. In this way, if the computer program requires too many subdivisions to get a good match, the technician can override it and manipulate the object by hand. Any two-dimensional motion of the vertex is translated into a three-dimensional translation using the equation described in Chapter III, page 64. This restricts the translation of the vertex so that it is always a minimum distance from its starting point, yet still maps to the correct pixel on the image.

**Mangling the Object**

Once the the vertices of the silhouette edges of the model object have been moved so that they will render to a reference object edge pixel, the object deformations should be applied to the interior of the object. Without this additional deformation, there will be large discontinuities around the silhouette edges of the model object that may be visible in the addition of lighting effects.

The interior deformation is done by first locating the neighbors of the pixels that were moved during the silhouette edge deformation. For each one of these pixels that have previously moved neighbors, the following procedure is applied: the positional change of each previously moved pixel is calculated as a vector; all the change vectors of the neighbors of the pixel in question that have moved are averaged; this average motion vector is then scaled by a factor that gets smaller as the distance from the silhouette edge increases; this final motion vector is applied to the pixel. This procedure is applied repetitively until all pixels in the object description have been adjusted. (Although, pixels that are far into the interior of the object may not be moved at
all because of the scaling factor. The scaling factor makes the changes near to the silhouette edge large, while farther into the description of the object, the changes are much smaller. The mangling process is the next step applied to the example image and the model object cylinder. The results (which are very slight) are shown in Figure 88.

![Figure 88: The interior points of the model object are adjusted to reduce the discontinuities of the scrunching process.](image)

5.4.7 The Results of This Process

After the process of matching, positioning, scrunching and mangling is done to each object in the reference scene, a complete three-dimensional description of the scene will have been created. The result of this for the example scene are shown in Figure 89. For each object in the reference scene an approximate model will be positioned so that it is at the same three-dimensional location as the object in the reference scene.
These model objects will have been slightly deformed so that their silhouettes match the reference objects. At this point the rotoscoping technician presses the "Write" button on the display. This causes all the information about the image to written to a file that a rendering program can understand.

Figure 89: The entire scene is modeled through the process of placement, probing, translating, scrunching and mangling.

The LIVE program is also set up to handle sequences of frames as well. After the first frame has been modeled, the next frame can be read in and the positions from the first frame can be used as a starting point for the modeling of the next frame. In fact, as long as the objects do not move excessively, the rotoscoping technician should not have to manually reposition the objects, only press the match button and have the program calculate the next position. If the object has moved too much, the technician will have to move the object as the matching process will likely use incorrect edges.
The real advantage comes when dealing with stationary objects. They should not have to be repositioned at all, even if the camera moves. If, however, the motion of the camera exposes some error in the position of a stationary model object, the error can be corrected at the new perspective and this new position can be used to refine the position arrived at in the first frame.

After all the frames of the sequence of film have been modeled and saved, the next step is to composite any new images into the original image. The next section will describe the software developed to accomplish this task.

5.5 The Composition Process

In Chapter IV the mathematics and theory of compositing two images together where the two images can affect the lighting on each other was discussed. Two programs to implement the concepts developed in that chapter were created. The first program developed was a modification of the concept of a ray tracing renderer. It is called RAY and generates a multipart rendering of the model scene description created by LIVE. This information is used by COMP to composite the new image into the old reference image and add any new shadows and lighting effects to the reference and new image. This section will describe the details of how each program works.

Both programs were written in C and will run on any machine running the UNIX operating system. RAY, as is traditional for ray tracers, takes quite a long time to run, so the more powerful the machine, the better.
5.5.1 The Ray Tracer

RAY is a traditional ray tracer. It casts a ray from the eye point through a pixel and intersects it with all the objects in the scene. The closest point of intersection is the visible object at that pixel and it determines the color of that pixel. Once the point of intersection is found, more rays can be cast from that point in the direction of the light sources to see if objects come between the intersection point and the light sources and cast a shadow on that point.

The ability to easily calculate shadows was the reason that this method of rendering objects was chosen. Traditional ray tracers generate a single image. This was unacceptable as the final image is actually the combination of a large amount of information. At each pixel the amount of light hitting that object at that point is calculated for each light source in the scene. To produce the final image, all the information for all the light sources is combined for each pixel. In the process of doing that, the individual light source information is lost.

RAY does not generate a single image. Rather, it generates multiple images that keep all the information that the ray tracing process creates separate.

RAY works by accepting as input the scene description file generated by LIVE. An example of such a file is shown in Figure 90. There are three main parts to this example file. The first paragraph of commands describe the image and where the observer is located. The second paragraph of commands specify all the information about any lights that exists in the scene. This section can be repeated as many times as is needed. (In this example, there is only one light.) The paragraphs that follow
Figure 90: The scene description file created by LIVE for the example scene.
the lights describe the individual objects in the scene. The position of the objects and the file that contains the object description are given. In this example, the first object is a cube that represents the book and the second object is a cylinder that represents the cookie tin. (The objects representing the ball and the floor were not included for brevity.) Both objects started as simple descriptions of geometric objects. However, when LIVE deformed the objects so that they would fit the silhouettes of the reference objects, new object descriptions were created that were more complex.

The Different Information Calculated

Many files are created by the ray tracing program. Their purpose was described in Chapter IV, Section 4.2.3. In this section the images produced for the example shown in this chapter will be shown and explained.

RAY produces a text file which describes the image files it produces and is used by the program COMP to show where the files are stored. The text file produced after running RAY on the scene description file in Figure 90 is shown in Figure 91.

Each line of the file specifies the location and parameters associated with a different aspect of the output. Each line is explained below:
Figure 92: The preliminary rendering of the example scene.

- **rendered**: The file specified on this line of the scene description file is an image of the complete rendering of the model scene with any new objects inserted. This file is not used during the composition process as it does not have its components separated, but it is useful to examine as an intermediary step, to see if the approximate rendering is close to the actual image. The example scene rendering is shown in Figure 92.

- **new_rend**: This image contains the visible portions of any new objects that are to be added to the reference scene. The objects have been completely rendered with all lighting effects (shadows and highlights) included. The composition program will simply add this image to the new image. The background of this image is a special color that the compositer recognizes. Only
A object to approximate the control object is added to the scene. It is rendered by the ray tracer using faceted shading.

non-background parts of this image will be overwritten on the new image. For the example scene an object was created to approximate a control object added to the actual scene. This new object was rendered by the ray tracer at this step, the results of which can be seen in Figure 93.

- **object**: This line of the scene description file specifies the number of objects in the reference scene and the name of the file that contains an image where each pixel identifies which object is visible at that point. This is used by the composition program to locate areas of the image that are covered by an object. The object image for the example scene is shown in Figure 94.

- **ambient_light**: The intensity of the ambient light and the image showing
Figure 94: All the pixels that are covered by a specific object are identified in this image.

which objects are illuminated by ambient light is given on this line of the scene description file. The intensity is a real number relative to the other lights in the scene. In the case of the sample scene, every pixel is illuminated by ambient light so the image is completely white and is not shown here.

- **lights**: This line in the file describes the number of lights and the maximum intensity a pixel can experience. This is useful for correctly scaling the intensity of light hitting a pixel.

- **light**: There can be multiple occurrences of this line. The first number on the line of the scene description file is the identification tag for the light. After that the intensity of the light, the name of file with the model scene rendering using
Figure 95: The model scene of the example image is rendered illuminated by the light.

only that light and the position of the light is given. In the example scene there
is one light and the rendering is shown in Figure 95.

• shadow: For each light there must be a corresponding shadow image. The
first number on the line is the identification number that ties this image with the
appropriate light. Then the name of the file that stores the image of where new
shadows fall, from the indicated light, caused by the addition of new objects.
Such a file for the example scene is shown in Figure 96.

• original_image: Finally, the file name of the original reference image must be
specified. This is the image that will have the new objects and lighting effects
added to it. The reference image for the example covered in this chapter was
Figure 96: The areas of the image that have new shadows falling on them are represented by this image.

All the images specified in the text file are written to disk and stored in a Run-Length-Encoded format. This should be enough information for the composition program to correctly add the new components.

5.5.2 The Compositor

The final program in the image composition sequence is COMP. It is used to take all the elements produced by the program RAY and correctly integrate any new objects into the original image. COMP is run by giving it the text file produced by RAY as input. There are also some parameters that can be specified to control how the image is analyzed.
As explained in Chapter IV, in order for shadows to be correctly added to the old image, the underlying color (or colors) of each object must be computed. In this way, it is possible to locate specular highlights on the objects and correctly remove them from the image. To do this, the pixel colors that comprise each object are grouped into "slices" around the central light axis. (Again, this procedure is more thoroughly explained in Chapter IV.) The program COMP accepts as a parameter the number of slices the color space should be divided. In general, the more colors on an object require a higher resolution of slices to accurately separate one color from another. The number of slices increases the memory and decreases the speed of the program. A reasonable number seems to be 50 slices. Also, there must be a way to differentiate between specular highlights and parts of an object that are simply colored white. In a specular highlight, the underlying color of the object is combined with the color of the light making the resulting color closer to the central light vector. A pure white section of the object will have its color fall directly on (or very near) the light vector. Therefore a cone around the light vector is defined by specifying an angle from the light vector. If the color vector of a pixel and the light vector describe an angle that is smaller than the specified cone angle, that pixel is treated as white. Both the number of slices and the white cone angle can be specified when the program is started.

The program is executed by giving it the name of the file produced by RAY that contains all the names of the images used in the composition process. (Such a file was shown in Figure 91.) There are default values for the number of color slices and the white cone angle but those values may be modified by providing them at
the initiation of the program. The program also has several debugging options which allow the user to produce graphs and images of the performance of the program.

![Image of the composition process result](image_url)

Figure 97: The results of the composition process on the example image.

When COMP is finished it produces a final image file. This is the final result of compositing any newly rendered objects with the reference image. The final image for the example program is shown in Figure 97. For comparison and as a control for the experiment, a spray can was inserted into the original image and a photograph of it was taken at the same camera position. The spray can’s size was measured and the location of the spray can was recorded. This photograph is shown in Figure 98.

The results shown in Figure 97 compare favorably with the actual photograph of Figure 98. Of particular interest is the edge of the lid of the cookie tin. In the original image, a specular highlight is present at that point. In the final image, the area of the specular highlight that is covered and a new shadow is at the same intensity as
Figure 98: A control image photographed with a spray can in approximately the same place as the inserted cylinder.

...the other areas of the new shadow. The specular highlight has been removed. This is also evident on the control image where the spray can was specifically set to partially cover the specular highlight.

Image Analysis

During this stage the example image that has been shown throughout the chapter is analyzed, object by object. Using the object image (Figure 94) the area of the image covered by each object can be identified. Then, for each object in the scene, the base colors of the object are calculated. (The procedure for this calculation is explained in Chapter IV, Section 4.2.4.) Once this is done, pixels with specular highlights can be identified. Also, by building a correspondence between the actual image and
the rendering of the model scene (Figure 95) using a single light source, the actual intensity of a shadow caused by obscuring this light source can be calculated.

**Adding New Lighting Effects**

When the new lighting effects (shadows) are added to the scene, a pixel can be identified as either having a specular component or being in shadow from a light source. When a shadow is added to the pixel, any specular component is removed and the intensity of the pixel is lowered only enough to get it to the correct shadow intensity. This algorithm guards against reducing the intensity of an already shadowed pixel.

There is a problem that was encountered during this phase of the algorithm. When digitizing the image, because it is a sampling process, aliasing artifacts are introduced. In particular, consider the case of an area of an image that is the boundary between a colored area of an object and a completely white area of the object. If the area is sampled by a single pixel when digitized, the recorded color will not be white or the color but an averaging of the two. Unfortunately, this averaging — adding a color vector to a white vector — looks exactly the same as a specular highlight. This means that the borders of a white pattern in a object will appear as specular highlights to the image analysis part of the program and if a shadow falls on them the white portion of the color will be removed, just as if it was a specular highlight. This is unacceptable since part of the white pattern will be erased from the image when it shouldn't have been. Digitizing the image at a higher resolution doesn't solve the problem, it only
makes it smaller. A possible solution to this problem is to use the position of the highlights on the rendering of the model object as a guide to the colors that would be considered "white" and not a highlight. If a pixel is flagged as a highlight pixel and it is very close to an area on the model object that was rendered to have a highlight on it, then there is a good chance that the pixel has a highlight. However, if the pixel is a considerable distance from where a highlight is expected, the pixel is probably just a aliased white pattern and should not be modified. (Problems occur when the reference object has many bumps and wrinkles. If the model object is smooth and doesn't predict any highlights on the wrinkles, these highlights will be treated as a white pattern.) In all other cases, this algorithm returns a seamless way to integrate a new object into a digitized scene.

5.6 Chapter Summary

This chapter has presented the basic workings and mechanisms of the implementation of the system used to composite images with lighting effect together. The system, while having a few shortcomings, is simple to use and quite effective. The object positioning program is fast, highly interactive and effective. The ray tracing program and the image composition program are slower, but they require no user interaction. This system was developed as a proof of concept and necessarily has some limitations: the object positioning program needs a better user interface and should handle articulated objects; the ray tracing program should have antialiasing, pnuembras, and complicated light sources; the composition program should deal with the white border problem and handle more light sources. All of these problems are just extensions on
the basic concepts and should not prove too difficult, just time consuming. The next chapter will discuss future research in more detail and conclude the thesis.
In this thesis the mathematics and algorithms to extract three-dimensional information from an image using knowledge of the object sizes was presented. It was shown that, under certain circumstances, a reasonably accurate model of the scene can be created. This model can be used in the analysis of the scene for many purposes.

In this thesis, one use of the three-dimensional model of the scene was examined. It was shown that it is possible to integrate new images into the original scene so that the lighting effects of these new images are taken into account. This has applications in the entertainment industry, in artistic endeavors and architectural rendering. These techniques apply to any application where visualizing a new object in an environment is useful.

6.1 Future Work

The size of the project necessitated that some possibilities were only slightly touched on. In this section, the directions that future research into this topic should follow will be discussed.
6.1.1 Three-Dimensional Analysis

A key assumption of this thesis is that there is a human technician operating the program and guiding the initial positioning of the model objects. There is a great potential to tie this work in with object recognition programs and scene understanding to remove some (or all) of the technicians work.

New edge detection and connection programs producing better results than those in this project can be used to increase the accuracy of the three-dimensional analysis.

Different object representations would allow the program to be more flexible and more accurate in matching model objects to reference objects.

A considerable amount of work should be spent looking into calculating the three-dimensional location of "fuzzy" objects such as trees and plants. At this point, this program works best on solid, inflexible objects such as books and actors with short haircuts. Calculating the location of objects that flex and change size and do not have hard silhouette edges is currently prone to a considerable amount of error.

Adding constraints to the scene description could increase the accuracy of the three-dimensional location. At this point when a complex object is built out of simple objects, the relationship between the objects is defined. This helps define the location of the overall object. However, less rigid relationships such as requiring that certain objects can not penetrate other objects could increase the accuracy of the overall scene analysis. As an example, in the example of Chapter V, the ball is resting on the floor. However, since the positioning program knows nothing of this constraint, the location of the ball might be found so that it is slightly above the
floor. In the application of integrating new elements into the scene, this inaccuracy is usually acceptable, but for a more precise reconstruction of the scene, knowledge of the relationships between objects should be taken into account.

Also, in a sequence of images, if the objects or the camera moves, information in one image might be applicable to a previous or future image. At this point, this information is used in an ad hoc manner: if the location of an object in one image is far off from the previous or next image, then it should probably be recalculated. In the future, the sequence of locations for an object could be analyzed and a "best" location or path for the object could be calculated. This should result in improved accuracy in the algorithm.

Finally, this project was developed on the other side of the fence from computer vision. By tying in the advances in computer vision, the human technician could be spared work and eventually removed from the process. Projects in image recognition could be included to calculate the starting search point that the technician provides. Some very interesting work in calculating the velocity of moving objects in images [16, 23] could be included to predict the position of objects in following frames in a film.

6.1.2 Image Composition

While the image composition phase performs well, there are some areas that should be examined further. In this project a ray tracer was developed to calculate a reference rendering of the scene. Unfortunately, ray tracers do not model many complex lighting effects correctly. By using a radiosity renderer instead, a more accurate reference image could be calculated and hence a more accurate analysis of the actual image
would take place. In general, since a real world image is being analyzed, the more photo-realistic the renderer, the more accurate the analysis.

Multiple lights in a scene cause many problems. The model scene rendering must be used to calculate which light causes a specific highlight or shadow. More sophisticated analysis of the scene must be done to correctly add new lighting effects to the scene.

In addition to shadows and highlights, there is a large problem when the phenomenon of reflections is taken into account. If a new object is placed so that a reflection on a reflective object is changed, that change should be carried out. This involves careful recording of the reflectiveness of the objects in the original scene and accurate models of those objects.

6.2 Conclusion

A complete three-dimensional rotoscoping system has been presented. With this system it is possible to extract the three-dimensional position of measured objects in an image. Using this information, it is possible to accurately integrate new images with the original image so that lighting effects introduced by the new object are integrated correctly.
BIBLIOGRAPHY


