Synthesis of Navigable 3-D Environments from Human-Augmented Image Data

by

Matthew E. Antone
B.S., Electrical Engineering
Massachusetts Institute of Technology, 1996

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

September, 1996

© Matthew E. Antone
All Rights Reserved
The author hereby grants to MIT permission to reproduce
and distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author

Department of Electrical Engineering and Computer Science
May 26, 1996

Certified by
Andrew Lippman
Thesis Supervisor

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

MAR 21 1997
LIBRARIES
Synthesis of Navigable 3-D Environments from Human-Augmented Image Data

by

Matthew E. Antone
Submitted to the
Department of Electrical Engineering and Computer Science

August 28, 1996

in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

A system was designed, implemented, and tested which generates virtual environments based on real photographic image data. The system estimates camera parameters such as focal length and pose, generates three-dimensional (3-D) models, and interactively renders these models. Instead of attempting a fully automatic solution of the unknown scene and camera parameters, the system allows the user to supply a-priori knowledge about the scene to facilitate its reconstruction and calculates the more precise and complicated geometry automatically. Thus, using only a few images and a minimal set of user-input data, a 3-D environment can be synthesized efficiently and accurately.

Implemented system components include graphical user interfaces for specifying geometric constraints and for rendering 3-D scenes, and a mathematical computation system to calculate camera parameters and scene geometry. Several problems in machine vision and graphical user interface (GUI) design are addressed, including modeling moving objects, compositing surface texture information from multiple images, and representing scene data.

Thesis Advisor: Andrew Lippman
Title: Associate Director, MIT Media Laboratory
Acknowledgments

First and foremost, I owe my deepest gratitude to Henry Holtzman, my undergraduate research supervisor and mentor, without whom this project would not have been possible. All the work I have done in this area, both undergraduate and graduate, was inspired by his original Masters thesis and his creative suggestions. I thank him for not only getting my foot in the door, but for pushing me all the way through it.

Other people to whom I owe thanks, and to whom credit should be given, include:

Andy Lippman, for offering me the opportunity (and funding) to continue this project, and for allowing me to work on what I wanted.

Shawn Becker, whose knowledge, talent, and accomplishments are an inspiration.

Alysia Moon, for believing in me.

Jeff Wong, my symbiote. Not only would I not have survived these five years at MIT without him, but I also would probably never have known the joys and perils of inline skating.

Ross Yu, for matching (maybe even surpassing) my quoting ability.

Eugene Lin, who actually understood what I was doing. 15,000 lines...

Steve “GNU” Waldman, for his bottomless store of C++ knowledge.

Nancy Xiao, my drumming partner, for dragging me away from my workstation once in a while to go out and have something called “fun.”

The Simpsons, for giving me 30 minutes every day to forget about work.

The great folks at Quik Chow, Yummi (Ummm!), Sbarro, and the food trucks, for keeping me alive (though barely). Without them, I would be both wealthier and skinnier.

And of course, Mom and Dad, for providing emotional and financial support through it all.

“Let’s roll!”
Contents

Chapter 1: Introduction .................................................................15
  1.1 Problem Statement ............................................................15
    1.1.1 The Vision Problem ....................................................16
    1.1.2 Assumptions ..............................................................16
    1.1.3 Project Specification ....................................................16
  1.2 System Overview ..............................................................17

Chapter 2: Background ............................................................19
  2.1 Motivation .................................................................19
  2.2 Past Work .................................................................20
    2.2.1 Camera Parameters ....................................................20
    2.2.2 3-D Structure ..........................................................21
    2.2.3 User Interface Design ................................................21
  2.3 Overview of 3-D Graphics ................................................22
    2.3.1 Coordinate Frames ....................................................22
      2.3.1.1 Affine Transformations .......................................22
    2.3.2 Scene Description ....................................................24
    2.3.3 Camera Description ....................................................25
      2.3.3.1 Intrinsic Camera Parameters .................................26
      2.3.3.2 Extrinsic Camera Parameters ................................27
    2.3.4 View Specification ....................................................27
      2.3.4.1 View Reference Coordinates ................................27
      2.3.4.2 Projection Types ................................................28
Chapter 3: Single-Frame Model Synthesis .............................................. 33

3.1 Data Input ................................................................................. 33
  3.1.1 Graphical User Interface .................................................. 34
  3.1.2 Points ............................................................................. 35
  3.1.3 Edges ............................................................................. 37
  3.1.4 Surfaces .......................................................................... 38

3.2 Solving for Camera Parameters ................................................ 38
  3.2.1 Intrinsic Parameters ....................................................... 38
    3.2.1.1 Finding Vanishing Points ........................................ 39
    3.2.1.2 Computation of Focal Point ................................. 39
  3.2.2 Extrinsic Parameters ....................................................... 41
    3.2.2.1 Camera Orientation .............................................. 41
    3.2.2.2 Camera Translation .............................................. 41

3.3 Determination of Scene Geometry ............................................ 42
  3.3.1 Using a Known Direction and Point ................................. 42
  3.3.2 Using a Known Planar Surface ....................................... 43
    3.3.2.1 Planar Surface Determination ............................. 43
    3.3.2.2 Finding a Point on the Plane ............................ 44
  3.3.3 Model Refinements ........................................................ 44

Chapter 4: Multiple-Frame Analysis .................................................. 45

4.1 Data Input ............................................................................. 45
  4.1.1 Point Features ............................................................... 45
4.1.2 Transient Feature Identification ............................................. 46

4.2 Camera Rectification ................................................................. 46
  4.2.1 Parameters to be Solved ...................................................... 46
  4.2.2 Rectification Method ......................................................... 47

4.3 Augmenting the Model ............................................................. 48
  4.3.1 Utilizing Point Correspondences ....................................... 48
  4.3.2 Mobile Features ............................................................... 49

Chapter 5: The Renderer .................................................................. 51

  5.1 Determination of Surface Polygons ......................................... 51
    5.1.1 Convex Outline .................................................................. 51
    5.1.2 Concave Outline .............................................................. 53

  5.2 Rendering Surface Polygons .................................................... 54

  5.3 Rendering Surface Textures ...................................................... 55
    5.3.1 Polygon Scan Conversion ................................................ 56
    5.3.2 Visible Surface Determination ........................................... 56
    5.3.3 Texture Mapping .............................................................. 58
      5.3.3.1 Problems with Traditional Texture Maps ................... 58
      5.3.3.2 Dynamic Texture Mapping ....................................... 60
      5.3.3.3 Implementation of Texture Rendering .................... 60

  5.4 Manipulating the View ............................................................ 63
    5.4.1 Direct Camera Controls ................................................... 63
    5.4.2 Interactive Viewing ......................................................... 64
    5.4.3 View Menu Controls ....................................................... 65

Chapter 6: Experimental Results .................................................... 67

  6.1 Synthetic Images ................................................................. 67
    6.1.1 Single-Frame Performance .............................................. 68
      6.1.1.1 Camera Calibration Results ................................... 68
      6.1.1.2 Point Estimation Results ....................................... 70
6.1.2 Multiple-Frame Performance ........................................... 71
   6.1.2.1 Relative Camera Orientation ................................. 71
   6.1.2.2 Planar Surface Determination .............................. 72

6.2 Outdoor Scene .................................................................. 72
6.3 Indoor Scene .................................................................... 74

Chapter 7: Conclusions ......................................................... 75

7.1 Technique Assessment ..................................................... 75
   7.1.1 User Interface ........................................................ 75
   7.1.2 Analysis Methods .................................................... 76
   7.1.3 Texture Rendering ................................................... 76

7.2 Future Directions ........................................................... 76
   7.2.1 Interface Improvements .......................................... 76
   7.2.2 Analysis Improvements .......................................... 77
      7.2.2.1 Generalizing Camera Calibration ...................... 78
      7.2.2.2 Other Generalizations ..................................... 79
      7.2.2.3 Improving Estimation Accuracy ....................... 79

Appendix A: Vanishing Point Concept ................................. 81

   A.1 Origin of Vanishing Points ......................................... 81
   A.2 Focal Point Computation ............................................ 82

Appendix B: Mathematical Formulations ............................... 85

   B.1 Solving for Camera Translation ................................. 85
   B.2 Projection Equations ............................................... 86
   B.3 Frame Rectification .................................................. 87

Appendix C: Data Structures ............................................... 89

   C.1 Viewports ............................................................. 89
   C.2 Image Data Elements ................................................. 90
Figures

2-1 Affine Coordinate Transformations ............................................. 24
2-2 Pinhole Camera Model ............................................................ 26
2-3 View Reference Coordinate Frame ............................................. 28
2-4 Infinite View Volumes ............................................................. 29
2-5 Clip Planes ........................................................................ 29
2-6 Projection onto View Plane ...................................................... 32

3-1 Builder Image Window ............................................................. 34
3-2 Builder Control Window .......................................................... 35
3-3 Point Mode in Builder ............................................................ 36
3-4 Edge Mode in Builder ............................................................ 37
3-5 Relationship Between Focal and Vanishing Points ....................... 40
3-6 Solution From Known Point and Direction ................................ 43
3-7 Solution From Known Plane .................................................... 44

4-1 Three-Dimensional Feature Triangulation .................................. 48

5-1 Finding a Convex Outline ......................................................... 52
5-2 Inside and Outside a Polygon Edge .......................................... 53
5-3 Render Pipeline .................................................................. 54
5-4 Canonical View Volumes ......................................................... 55
5-5 Polygon Scan Conversion ....................................................... 56
5-6 Sample Z-Buffer .................................................................. 58
5-7 Unwarping a Texture Boundary .............................................. 59
5-8 Tagging Surfaces for Dynamic Texture Rendering............................61
5-9 Flowchart for Determining Pixel Color........................................61
5-10 Buffers for Desired Textured View............................................62
5-11 Renderer Display Window.........................................................63
5-12 Camera Dialog Box.................................................................64

6-1 Synthetic Images........................................................................68
6-2 Synthetic Single-Image Point Error..............................................70
6-3 Single-Frame Renders of Synthetic Images.....................................70
6-4 Render of Composited Cube Images.............................................72
6-5 Original Photograph for Single-Frame Analysis...............................73
6-6 Synthetic Views of Outdoor Scene...............................................73
6-7 Original Photographs for Multiple-Frame Analysis...........................74
6-8 Synthetic Views of Indoor Scene..................................................74

7-1 Camera Calibration for One and Two Vanishing Points......................78

A-1 Coordinate Frame of Focal Point and Vanishing Points....................83

C-1 Sample ViewPort File..................................................................89
C-2 Sample ImageData File...............................................................90
C-3 Point Structures..........................................................................91
C-4 Edge Structures...........................................................................91
C-5 Surface Structure.........................................................................92
Tables

6-1  Single-Frame Camera Accuracy of Synthetic Image A ...........................................68
6-2  Single-Frame Camera Accuracy of Synthetic Image B ...........................................69
6-3  Single-Frame Camera Accuracy of Synthetic Image C ...........................................69
6-4  Camera Accuracy of Synthetic Image B After Rectification ...............................71
6-5  Camera Accuracy of Synthetic Image C After Rectification ...............................71
Chapter 1
Introduction

Many tools exist for fabricating artificial virtual environments, but there are few which allow three-dimensional (3-D) scenes to be created from photographs of real objects. Most such tools require certain strong constraints on the images or the devices used to produce them, or both; some, for example, assume previous knowledge of the camera’s focal length or pose, some require images generated by a stereoscopic camera pair with a known distance of separation, and some require that a known calibration object be present in the images.

These constraints, however, limit the extent of what is actually accomplished; further, the lack of a simple user interface makes such systems impractical and cumbersome. It is thus desirable to design a system that is simple to use, yet powerful and flexible enough to generate accurate three-dimensional scenes quickly and easily from single or multiple photographs.

1.1 Problem Statement

This section outlines some of the problems involved with the field of machine vision. The general goals of this field are discussed, as are the more specific objectives of this project.
1.1.1 The Vision Problem

The goal of machine vision is to allow a computer to “see” objects and understand the spatial relationships between them. The machine should be able to analyze a digitized image and determine the three-dimensional shapes of objects in it, as well as have a sense of where the camera is located with respect to these objects. Ideally, a machine vision system is fully automatic and generalized, requiring no human interaction to perform its tasks. In practice, however, human assistance and constraint is always necessary, as the human brain has a much larger database of visual information from which to draw than a computer system and can more easily understand image content.

1.1.2 Assumptions

Aside from perceptual issues, the mathematical problems of machine vision are very complex, involving many sets of nonlinear equations that have no closed-form solution. It therefore makes sense to impose certain reasonable constraints on both the camera and the objects being modeled in order to simplify the system. Here, a specific set of assumptions is made.

The first of these is that the device used to produce the original photographs is a simple “pinhole camera”, assumed not to introduce radial lens distortion. Objects in the scene are assumed to be composed of planar surfaces, and the input images need to contain enough parallel lines to generate three principal vanishing points—namely, two such lines in each principal direction. Finally, some amount of \textit{a-priori} information must be supplied by the user, including parallelism and coplanarity of lines as well as two known 3-D positions. Many of the terms above, as well as reasons for choosing this particular set of constraints, will be discussed later in this paper.

1.1.3 Project Specification

Once the above constraints are imposed, a specification for the system can be outlined. The general goal is to synthesize 3-D models from single or multiple photographs; in particular the system must meet the following objectives:

- Allow the user to specify the minimal number of geometric relationships for a
model to be generated.

• Estimate camera intrinsics and pose.

• Convert 2-D image elements into 3-D points and lines.

• Map textures from the images onto generated planar surfaces.

• Represent scene data efficiently and accurately.

• Allow the user to interactively view and refine the final 3-D scene.

1.2 System Overview

The implemented system was written in C++ and consists of a computation layer and an application layer. The computation layer includes all underlying functionality, such as 2-D and 3-D graphics libraries, mathematical analysis tools, and functions that manipulate scene data. The application layer consists of two major modules called Builder and Renderer. These modules are separate graphical user interface (GUI) applications built up from the underlying computation layer, and were written using the wxWindows interface library [13].

Builder is a GUI which allows the user to impose constraints on the geometry of a scene and then, given these constraints, solves for camera parameters and three-dimensional point locations. The single-image operations, such as marking points and identifying parallel and coplanar lines, are described in Chapter 3; Builder has multiple-frame analysis capabilities as well, which are outlined in Chapter 4. These include identifying point correspondences between images, tagging objects that move in relation to the background, computing relative camera orientation, and compositing points, lines, and textures into a single, coherent scene database.

Renderer, described in Chapter 5, allows the previously-created three-dimensional scenes to be viewed interactively as seen from an arbitrary camera. It has controls for manipulating the viewpoint, and displays either a simple wireframe model consisting of the user-defined points and lines or a fully texture-mapped image using all available image information.
Chapter 2
Background

This chapter outlines some of the background behind this project. Applications of the system and previous research upon which the project builds are discussed. In addition, Section 2.3 provides the reader with an overview of 3-D computer graphics, defining many of the terms that will be used throughout the rest of the paper.

2.1 Motivation

Creating 3-D models from 2-D image information has many applications, including physical modeling, computer-aided design (CAD), architecture, measurement, and video coding. Three-dimensional representations convey information in a much more powerful way than do their two-dimensional counterparts. They allow complex scenes to be navigated and viewed interactively, rather than forcing a passive, static view.

Many mathematical algorithms exist for solving various single problems in machine vision, including those of camera calibration, 3-D structure calculation, and texture rendering. However, there exist few, if any, systems that unify these methods into a simple, coherent process, one involving minimal user interaction and which can analyze a scene from imposing constraints to producing a fully texture-mapped three-dimensional model.

The goal of this project, then, is to define such a process and create an applications package that utilizes it. The package is simple enough to be used quickly and easily, and
powerful enough to create composite 3-D models from multiple images and interactively view them.

2.2 Past Work

Much research has already been conducted in areas relevant to this work, including the areas of automatic and semi-automatic camera calibration, 3-D structure determination, and texture compositing. The majority of this work falls into one of two main categories. In the first of these are methods that focus on solving a single aspect of the general vision problem, such as camera position, camera intrinsics, or geometric structure, while assuming that the remaining parameters are known. The second category contains systems which attempt to iteratively solve several of these problems at once with appropriately chosen initial conditions. Some of this past work is briefly discussed in the sections that follow.

2.2.1 Camera Parameters

Several different techniques have been devised to solve the problem of calibrating an unknown camera based on an image it produced. Most of these utilize a calibration object of known dimensions and shape placed in the original scene which, when detected in the image, can provide enough constraint to allow the camera’s intrinsics and extrinsics to be determined. For example, Wang and Tsai [8] place a fixed-size rectangular parallelepiped in the camera’s field of view and estimate camera parameters by analyzing how its edges project on the image. Such approaches are limiting, however, because they rely on somewhat contrived scenes and cannot operate on previously imaged photographs that do not contain specific calibration objects.

Echigo [6] proposes using three orthogonal parallel line sets and applying the vanishing point concept to calibrate the camera. His approach decouples camera rotation from camera translation, since vanishing point locations depend only on the orientation and not the position of the camera (see Appendix A.1). Automatic edge detection and correlation methods are used to find lines in an image and group them into three sets, making the approach somewhat more generalized. However, it requires at least three correspondences
between known 3-D positions and 2-D image projections in order to determine the translational parameters, and it assumes that the center of projection is known.

Relative camera orientations across multiple images are usually solved strictly by identifying point correspondences between these images. Horn [2] addresses the problem of unknown relative orientation, but its solution requires that the depths of projected points and the camera's focal length are known.

2.2.2 3-D Structure

Recovering 3-D structure from multiple images with known camera parameters is relatively straightforward. Existing techniques rely mainly on identifying point correspondences between the images produced by these cameras and solving depths by triangulation. Structure calculation thus reduces to the two-dimensional problem of finding these correspondences.

If only a single image is available, however, the problem becomes more difficult, even with a known camera. Constraints other than projections must be devised in order to solve for the 3-D positions of points in the scene. Most such constraints involve assumptions about the shapes of the objects. In the robot location methods used by Lee, Lu, and Tsai [9], for example, rectangular shapes are assumed, and their dimensions can be determined based on the parallelism and orthogonality of their edges.

Horn and Tsai, among others, have done extensive work in more general areas, such as the simultaneous solution of camera parameters and 3-D structure. There are not enough inherent equations in these cases to solve for the vast number of unknowns, however. Constraint must thus be supplied by the user, usually in the form of many correspondences between 2-D projections and 3-D positions, which may not be known. In addition, the equation systems in such approaches are often nonlinear and in general have no closed-form solution. Even iterative techniques require that many boundary conditions be externally supplied in order to provide enough constraint for a feasible solution.

2.2.3 User Interface Design

Very few systems discuss the methods by which user data is input. This is an important consideration, however, as many of these systems could be made much simpler
and the analysis performed more quickly if intuitive interface tools existed to complement the underlying computational components.

Most authors do not address the issues of data input and output at all, despite the fact that this is, in some respects, the most critical element of the system. If data input methods are time-consuming and inefficient, the system becomes virtually useless; similarly, if a simple and intuitive output method is lacking, user feedback is limited and basic system performance cannot be assessed. One of the main focuses of this project is therefore the user interface.

2.3 Overview of 3-D Graphics

This section provides an overview of those elements of computer graphics which are relevant to this project. Descriptions of three-dimensional objects, the ideal camera model, and viewing transformations are discussed here in order to introduce the necessary concepts and lay the groundwork for the rest of this paper.

2.3.1 Coordinate Frames

A coordinate frame, as defined in this context, is a three-dimensional right-handed or left-handed Cartesian space. Coordinate transformations are the transformations, typically represented as 4-by-4 matrices, that give the coordinates of a point in one frame relative to another. These points upon which they operate are represented as 4-vectors of the form

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$$

(2-1)

so that when such a point is multiplied by a transformation matrix, the resulting vector is the new, transformed point.

2.3.1.1 Affine Transformations

There are many different types of transformations that can be applied to three-dimensional points. The four most common types are translation, rotation, scale, and shear. They preserve parallelism of lines, and are known as affine transformations.
Translation is simply a shift or rigid movement of one frame relative to another. It is specified by the amount of motion in each of the three coordinate directions, and the matrix which represents it is

\[
T = \begin{bmatrix}
1 & 0 & 0 & T_x \\
0 & 1 & 0 & T_y \\
0 & 0 & 1 & T_z \\
0 & 0 & 0 & 1
\end{bmatrix}.
\]  

(2-2)

Pure rotations are always performed about the origin of a space, and can be specified in many different ways. The simplest description defines a virtual axis of rotation and an angle through which the frame is rotated. A typical rotation matrix is of the form

\[
R = \begin{bmatrix}
R_{11} & R_{12} & R_{13} & 0 \\
R_{21} & R_{22} & R_{23} & 0 \\
R_{31} & R_{32} & R_{33} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}.
\]  

(2-3)

where the rows and columns of \( R \) are orthonormal.

Scaling stretches a frame along each of its three coordinate axes. A scale transformation requires three parameters, namely the factors by which the frame will be scaled in each of the three coordinate directions. The matrix that represents a scale transformation is

\[
S = \begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}.
\]  

(2-4)

Finally, the shear transformation is a nonuniform shift along a coordinate axis. It is specified by two proportionality constants that dictate how points on the other two axes
should be shifted as a function of their distance along the shear axis. The matrix that
shears points along the z axis is given by

$$SH = \begin{bmatrix}
1 & 0 & SH_x & 0 \\
0 & 1 & SH_y & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}.$$  \hfill (2-5)

Matrices that shear about the x and y axes can be constructed in a similar fashion.

Figure 2-1 shows the effects of applying each of these transformations to a cube.

![Affine Coordinate Transformations](image)

**Figure 2-1: Affine Coordinate Transformations**

Examples of 3-D affine transformations as applied to a cube. Translation is
applied in (a), rotation in (b), scale in (c), and shear in (d). Lines which are parallel
before the transformation remain parallel afterward. Translation and rotation pre-
serve relative angle and length, while scale preserves only relative angle.

2.3.2 **Scene Description**

A *scene*, or collection of three-dimensional objects, is described using a finite set
of primitive shapes. The scene is specified in its own right-handed Cartesian reference
frame, referred to here as *scene space* or *world space*. It can contain points, lines, poly-
gons, and other more arbitrary shapes, so long as these shapes can be reasonably represented.

Simple points in the scene are specified using three parameters, namely their $x$, $y$, and $z$ coordinates in the scene space. A line segment is specified as two such points, defining the extent of the segment. A polygon consists simply of a list of single points, ordered so that adjacent points in the list form the edges of the polygon; the first and last points in the list are also connected.

Infinite planar surfaces are represented by the four-parameter canonical equation

$$Ax + By + Cz + D = 0.$$  \hspace{1cm} (2-6)

The first three parameters $A$, $B$, and $C$ are the coordinate directions of the plane normal, the vector perpendicular to all lines that lie in the plane. The last parameter $D$ is the offset of the plane from the origin.

2.3.3 Camera Description

The camera is the device used to actually produce the images. Here, an ideal pinhole camera model is assumed, as it is a reasonable approximation and significantly simplifies computations. In this model, the camera is defined completely by nine parameters, three intrinsic and six extrinsic, which are described in more detail below. Other more complicated parameters, such as those governing radial lens distortion, are ignored here.

Two alternate depictions of the pinhole camera model are shown in Figure 2-2. In the first, light rays of varying intensity are focused through a single point, known as the focal point, and projected onto a planar surface at a fixed distance from this point, known as the focal distance or focal length. Note that the image on the plane appears inverted. In the second, the rays are projected onto an image plane located in front of the focal point rather than behind it, but at the same focal distance. Here, the image is not inverted. The latter representation will be used throughout this paper.
Figure 2-2: Pinhole Camera Model
In the pinhole model, light rays are focused through a single point and project onto the image plane. The actual model, which inverts the objects on its image plane, is shown in (a), while (b) shows a conceptually simpler model in which objects are not inverted.

Objects in the scene which are further away from the focal point appear smaller than objects which are close. The magnitude of this foreshortening effect is controlled solely by the focal distance: a smaller focal distance, relative to the size of the imaging window, will cause the effect to be more drastic, while a larger focal distance approaches orthographic projection, described in Section 2.3.4.2. The equations governing perspective are given in Section 2.3.5.3.

2.3.3.1 Intrinsic Camera Parameters
The three intrinsic camera parameters describe the internal characteristics of the camera. They are measured in a coordinate frame relative to the camera itself, independent of the surrounding environment. This coordinate frame has its origin at the center of the image window, and its $x$ and $y$ axes are the horizontal and vertical directions, respectively, on the image plane. The frame is called the view reference coordinate (VRC) frame, and it is a left-handed frame, meaning that its $z$ axis can be found by taking the vector cross-product of the $y$ axis with the $x$ axis. Thus, its positive $z$ direction is perpendicular to the image plane, away from the focal point and toward the objects being projected, as shown in Figure 2-3.

The three intrinsic parameters, then, are the three Cartesian coordinates $F_x$, $F_y$, and $F_z$ representing the location of the focal point $F$ in this camera-relative VRC system. The $z$ component of this point is the negative of the focal length.
2.3.3.2 Extrinsic Camera Parameters

The remaining six parameters give the camera an extrinsic location in scene space. They are a translation and a rotation denoting the position of the VRC origin and orientation of the VRC axes relative to the scene space frame. The translation, from Section 2.3.1.1, is specified by x, y, and z offsets. The rotation can be specified in several different ways; the minimum description, however, requires only three parameters, specifically an angle of rotation about each of the coordinate axes, since any arbitrary rotation can be decomposed into its Euler angles.

2.3.4 View Specification

Objects in scene space are view-independent, meaning that their positions in the scene space coordinate frame do not change regardless of the viewpoint from which they are seen. In order to specify this viewpoint, we define a camera’s focal point as the viewer’s “eye”, with which the scene objects are viewed.

The final image is displayed in a rectangular window on the computer’s screen. This window forms a rectangular bound in the image plane and can be thought of as an actual window in the scene space that the camera looks through. Anything outside the camera’s field of view, i.e. outside the window’s view extent or view volume, is ignored, since it is effectively invisible.

2.3.4.1 View Reference Coordinates

In order to perform the transformations that project objects in the scene onto the planar view surface, it is necessary to define the camera-relative VRC frame introduced in Section 2.3.3.1. The frame is specified relative to the global scene space, with its x-y plane coincident with the view plane, or plane onto which scene points are projected. Its z axis is directed from the camera’s focal point towards the view plane, and its origin is at the center of the image window.

Several parameters define the VRC frame. The first of these is the view reference point (VRP), which is a point in scene space that corresponds to the VRC origin. Next is the view plane normal (VPN), a scene space vector in the direction perpendicular to the image window.
The image window itself is a bounded rectangle in the view plane with its center at the VRP. This region corresponds to the rectangular window or buffer onto which the scene is mapped. It is specified by a width, a height, and an up vector (VUP) which, when projected onto the view plane, forms the y axis of the VRC space. The x axis of the space is then found by taking the vector cross product of the z axis with the y axis.

![View Reference Coordinate Frame](image)

**Figure 2-3: View Reference Coordinate Frame**
The left-handed VRC space has its origin at the reference point (VRP), its z axis coincident with the view plane normal (VPN), and its x axis along the vector formed by taking the cross product between the VPN and VUP vectors.

### 2.3.4.2 Projection Types

Once the coordinate frame and the view window have been defined, a projection type, which dictates how light rays converge through the camera’s lens, must be specified. The two main types are known as *perspective projection* and *parallel projection*.

In perspective projection, all visible light rays converge to a single point, namely the focal point defined in Section 2.3.3. This point, once specified, completes the definition of the view volume in the pinhole camera model. As shown in Figure 2-4(a), the volume is an infinite right pyramid whose apex is the focal point and whose sides pass through the edges of the view window. Under this type of projection, parallelism is generally not preserved.

In parallel or *orthographic* projection, the light rays do not converge, and parallel lines in the scene do project to parallel lines in the image. If an orthogonal direction of projection is assumed, nothing more needs to be specified in order to define the view vol-
ume. It is an infinite right parallelepiped whose sides are coincident with the edges of the viewing window, as shown in Figure 2-4(b).

![Figure 2-4: Infinite View Volumes](image)

The infinite view volumes formed under (a) perspective projection, and (b) parallel projection.

### 2.3.4.3 Clip Planes

In general, it is not desirable to project objects that are extremely far away or extremely close to the viewer, as they do not contribute significantly to the final image. Thus, two more pieces of information are typically supplied to augment the definition of the view volume. The front and back *clip planes* truncate the view volume at each end. They are parallel to the view plane and defined by two distances from the VRP along the VPN. The view volume thus becomes a truncated, finite pyramid or rectangular solid.

![Figure 2-5: Clip Planes](image)

Truncated view volumes formed by front and back clip planes under (a) perspective and (b) parallel projection.
2.3.4.4 Vanishing Points

As a direct consequence of perspective projection, lines which are parallel in scene space generally tend to converge to a single point when projected onto the image. This point is known as a vanishing point. A principal vanishing point is the point to which lines parallel to a principal coordinate axis converge.

A scene can have one, two, three, or no apparent principal vanishing points. The number is dictated both by the type of projection—for example, there are never vanishing points under parallel projection—and by the orientation of the camera. The axes that have principal vanishing points under perspective projection are those cut by the view plane; Appendix A provides a more detailed discussion.

2.3.5 View Transformations

Having defined the specification for a given view, we now delve into the transformations that map scene points onto the view plane. Four affine transformations, which transform points into camera space, are followed by a perspective warp if applicable.

2.3.5.1 Transformation into VRC Space

Two affine transformations, a translation and a rotation, take points from scene space to VRC space. The translation shifts the origin of scene space to be coincident with the origin of VRC space, namely the VRP. The matrix which represents this transformation is

\[
T_1 = \begin{bmatrix}
1 & 0 & 0 & -VRP_x \\
0 & 1 & 0 & -VRP_y \\
0 & 0 & 1 & -VRP_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]  \hfill (2-7)

The three scene space coordinate axes are then rotated about the VRP so that the z axis becomes coincident with the VPN and the y axis becomes coincident with the projection of VUP onto the view plane. An appropriate rotation matrix can be formed from the
vectors $R_x$, $R_y$, and $R_z$, which are the normalized directions in scene space that represent the three axes of VRC space. They are defined as follows:

$$R_z = \frac{VPN}{\|VPN\|}$$

$$R_x = \frac{R_z \times VUP}{\|R_z \times VUP\|}$$

$$R_y = R_x \times R_z.$$  \hspace{1cm} (2-8)

Once these vectors are computed, the rotation matrix that takes the three original scene space coordinate axes to $R_x$, $R_y$, and $R_z$ is simply

$$R = \begin{bmatrix}
R_{x1} & R_{x2} & R_{x3} & 0 \\
R_{y1} & R_{y2} & R_{y3} & 0 \\
R_{z1} & R_{z2} & R_{z3} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}.$$  \hspace{1cm} (2-9)

where $R_{x1}$ denotes the first component of the vector $R_x$, and so forth.

### 2.3.5.2 Transformation into Camera Space

If perspective projection is desired, two additional affine transformations are performed to take points from VRC space into camera space. The first of these is another translation that shifts the focal point to the $x$-$y$ plane along the $z$ axis, given by

$$T_z = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & -F_z \\
0 & 0 & 0 & 1
\end{bmatrix}.$$  \hspace{1cm} (2-10)

The space is then sheared so that the vector from the focal point to the center of the image window is parallel to the $z$ axis. The shear transformation matrix is

$$SH = \begin{bmatrix}
1 & 0 & -F_x/F_z & 0 \\
0 & 1 & -F_y/F_z & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}.$$  \hspace{1cm} (2-11)
The overall view transformation can thus be written as

$$V = SH \cdot T(F_z) \cdot R \cdot T(-VRP).$$  \hspace{9cm} (2-12)

2.3.5.3 Mapping onto the View Plane

After these affine transformations are performed, the final step is to map the transformed points onto the view plane. Under parallel projection, the mapping is straightforward: each point's $z$ value is simply ignored, and the remaining $x$ and $y$ coordinates give its position on the view plane.

Under perspective projection, however, points must be mapped according to their distance from the focal point, as shown in Figure 2-6. For a given scene point $(x, y, z)$, now in VRC space, this mapping is given by the two projection equations

$$X_p = d \cdot \frac{x}{z},$$

$$Y_p = d \cdot \frac{y}{z},$$  \hspace{9cm} (2-13)

where $d$ is the focal length, and $(X_p, Y_p)$ are the coordinates of the projected point. These equations are derived from the similar triangles formed by the $z$ axis, the focal point, the projected point, and the scene point. The ratio of the focal distance $d$ to the projection $X_p$, for example, is equal to the ratio of scene coordinate $z$ to scene coordinate $x$.

![Figure 2-6: Projection onto View Plane](image)

Projection of a VRC-relative scene point $(x, y, z)$ onto the view plane under (a) parallel and (b) perspective projection. Note that in (a), the projected point's $x$ and $y$ coordinates are the same as those of the scene point. The projection equations are derived geometrically from the similar triangles formed in (b).
Chapter 3
Single-Frame Model Synthesis

This chapter describes the methods used to estimate camera parameters and construct a three-dimensional model from a single image. In previous approaches, some amount of simultaneous and iterative computations were required. Here, however, since user interaction is designed to be optimal for the calculations involved, the process can be neatly broken down into several independent stages, each of which builds on the one before. In addition, most solutions are analytic and require little or no iteration.

3.1 Data Input

Since a machine has no way of “understanding” a scene, several pieces of information must be obtained from the user to assist in the reconstruction. These are:

- A set of points to be solved.
- A set of edges connecting these points.
- A set of planar surfaces containing points and edges.
- The 3-D positions of two points.

This information, if complete, supplies the machine with enough geometric constraint to solve for the desired parameters.

The framework within which the user enters this data is a graphical user interface (GUI) that facilitates data entry by providing a simple mouse-based input method. This section describes the GUI framework, and the remaining sections delve into the analytic solution of the model itself.
3.1.1 Graphical User Interface

The data input interface, called "Builder", consists of two main windows. The first of these is the Image Window, where the original two-dimensional image can be manipulated in various ways. The second is the Control Window, where all of the input controls are located.

The Image Window, shown below in Figure 3-1, allows the user to load an image from disk, hide or show parts of the image, and resize the image. This window is also where most of the actual data input occurs: the user clicks on points with the mouse to select them for various tasks. A transparent overlay floats above the image, showing the scene elements that the user has entered.

![Image Window Diagram]

Figure 3-1: Builder Image Window
The Image Window in the Builder application. Points, edges, and surfaces are manipulated on the image using the mouse. Horizontal and vertical scrollbars allow the image to be shifted, and zoom controls allow the image to be magnified.

The Control Window, shown in Figure 3-2, contains all controls to edit the geometric constraints on what will become the three-dimensional model. From the Project menu, the user can read and write project files to and from disk, start a new project, or rename the current project. The Data menu allows the user to add, remove, clear, read and write sin-
gle-image data sets, which are listed in and viewed from the Image List. The Build button initiates the model construction process when all constraints have been entered.

![Builder Control Window](image)

**Figure 3-2: Builder Control Window**
This window contains controls for editing points, edges, and surfaces. Controls in the Edit Panel change depending on the mode selected in the Mode Panel. Projects can be loaded and saved from the Project menu, and single image data sets are manipulated via the Data menu. If a project contains multiple images, the user can switch between them using the Image List.

Builder's three main modes of operation, along with the constraints they operate on and their functions, are described in the next three sections.

### 3.1.2 Points

Points are the simplest scene elements to specify. From the Control Window, the user switches to Point Mode (shown in Figure 3-3) and clicks on those pixels in the image he wishes to define as points. Points can also be deleted and moved by selecting and dragging them.
An anchor is a special type of point whose three-dimensional position is supplied by the user. As will be seen in the following sections, at least two such anchors must be specified in order to determine camera parameters and reconstruct the scene, as they provide global position and scale information. The Anchor option allows the user to select a point in the image and assign it a 3-D location. Two useful anchors to specify are typically the scene space origin and a point on one of the scene space coordinate axes.

Show Info mode displays all relevant information about any selected point, including its coordinates in the image and its 3-D position if known. The Make Feature option pertains to multiple-frame analysis and is discussed in Chapter 4.
3.1.3 Edges

An edge is a line segment connecting two points. It contains not only references to its two endpoints, which must be defined prior to its creation, but also an associated scene space direction. Edges are manipulated by choosing Edge Mode in the Control Window.

![Image](image.png)

**Figure 3-4: Edge Mode in Builder**

In Edge Mode, edges can be created, deleted, and assigned a direction and sense. The "?" denotes the null direction. If the Show All Edges box is not checked, then only edges in the currently selected direction are shown in the Image Window.

Directions group the edges into sets for vanishing point computation and subsequent camera calibration, described in Section 3.2.1, as well as impose constraints on the scene geometry, as shown in Section 3.3.1. There are four possible directions in which an edge can lie. The first three are parallel to the $x$, $y$, and $z$ scene space coordinate axes, and the fourth is the so-called "null" or "unspecified" direction, which is assigned to all other edges. The unspecified edges need not be parallel to each other; "null" is merely a tag for those edges with no known direction.
Each principal direction has associated with it a sense, which determines whether the associated vanishing point is positive or negative along its axis. The sense is specified in order to ensure that the correct extrinsic camera parameters, not one of their symmetries, are found.

The user defines an edge by selecting one endpoint and then the other. Edges can be created and deleted in much the same way as points; they cannot, however, be moved. If refinement of edges is necessary, the original endpoints themselves can be moved in Point Mode, after which all attached edges change accordingly.

### 3.1.4 Surfaces

The final type of constraint is the surface. A surface is a plane with arbitrary parameters, as defined in Section 2.3.2, and consists of a set of edges and points. Specifying this coplanar set introduces more constraint on the scene geometry; see Section 3.3.2.

Surfaces can be created, named, and deleted, and any number of them can exist at once. As shown in Figure 3-2, the user can choose from the list of existing surfaces to modify, and then add elements to and delete elements from the selected surface. The selection's points and edges are highlighted in the Image Window, distinguishing them from the other scene elements.

### 3.2 Solving for Camera Parameters

Once the Build button is pressed, Builder assembles all the input data and uses it to generate the final model. The first step is the estimation of camera parameters, which happens in the order described below.

#### 3.2.1 Intrinsic Parameters

As defined in Section 2.3.3.1, a camera's three intrinsic parameters are its focal length and the x and y offsets of its center of projection. These can be thought of as the coordinates of the focal point relative to the VRC system.

The focal point is found using the vanishing point camera calibration concept [4]. Edges which are parallel in scene space appear, in general, to converge to a single point
when projected onto the image plane. This assumes of course that there is no lens distortion present, and that the image plane is not parallel to these edges; see Appendix A.1.

Once all edges in the image have been created and tagged, they can be grouped into three parallel sets corresponding to the \( x \), \( y \), and \( z \) scene space directions. Three vanishing points can then be found and used to estimate the camera's focal point in VRC space. Edges which have unspecified directions are ignored in this computation.

Note that if no vanishing points are found, i.e. if all edges in each of the three scene directions project as parallel line segments, then the scene was projected orthographically and the intrinsic parameters are somewhat irrelevant. The direction of projection is then assumed to be perpendicular to the image plane.

### 3.2.1.1 Finding Vanishing Points

The vanishing point for a given edge direction is found by first grouping all edges corresponding to that direction into a single set. The infinite line equation, in the form given by

\[
Ax + By + C = 0,
\]

is then found for each segment in the set, and a list of intersections is computed between all pairs of these infinite lines. The number of intersections is equal to

\[
n\frac{(n-1)}{2}
\]

where \( n \) is the number of edges in the given direction.

Each intersection point is also given a weight according to the product of the lengths of the line segments that produced it. This is done in order to count shorter line segments less heavily, since they are generally noisier approximations to the actual line than longer segments. Once all possible intersections have been found and all weights assigned, a weighted minimum mean-square error (MMSE) estimate of the actual intersection is computed, and this is used as the vanishing point for the given direction.

### 3.2.1.2 Computation of Focal Point

If three independent vanishing points can be computed in the previous step, it is straightforward to calculate the focal point immediately. If, however, only one or two finite
vanishing points exist, this is a degenerate case and the focal point cannot be found using the method that follows.

Recall that vanishing points are a direct consequence of perspective projection, under which parallel lines do not necessarily project as parallel. In the pinhole camera model described in Section 2.3.3, the focal length is the perpendicular distance from the focal point to the image plane.

Three edges can be drawn that intersect the focal point \( F \) and each of the vanishing points \( A, B, \) and \( C \), as shown in Figure 3-5. These three edges, \( FA, FB, \) and \( FC \), must be parallel to the three scene directions; this is shown in Appendix A.2.

![Figure 3-5: Relationship Between Focal and Vanishing Points](image)

Two views of the focal point \( F \) and the vanishing points \( A, B, \) and \( C \). The focal point is positioned such that edges \( FA, FB, \) and \( FC \) are mutually orthogonal.

Thus, \( FA, FB, \) and \( FC \) are geometrically constrained to be mutually orthogonal, and this constraint gives enough additional information to uniquely determine \( F \). Since the image-relative positions of points \( A, B, \) and \( C \) are known, three equations reflecting the constraints immediately follow:

\[
FA \cdot FB = 0 \\
FB \cdot FC = 0 \\
FC \cdot FA = 0.
\]  

(3-3)

Appendix A.2 outlines the solution of these equations for \( F_x, F_y, \) and \( F_z, \) the three unknown components of \( F \).
3.2.2 Extrinsic Parameters

The camera's six extrinsic parameters, as defined in Section 2.3.3.2, consist of three translational and three rotational components. These define the camera's position and orientation in scene space.

3.2.2.1 Camera Orientation

Calculation of the orientation is straightforward, since the three coordinate axes of each frame are known. The scene space axes are simply $e_1$, $e_2$, and $e_3$ (the principal coordinate directions), and the VRC axes are $FA$, $FB$, and $FC$ from Section 3.2.1.2. A matrix $R$ that rotates points in scene space to points in the non-translated VRC space can thus be formed whose columns are the normalized vectors $FA$, $FB$, and $FC$:

$$R = \begin{bmatrix}
\hat{FA}_x & \hat{FB}_x & \hat{FC}_x & 0 \\
\hat{FA}_y & \hat{FB}_y & \hat{FC}_y & 0 \\
\hat{FA}_z & \hat{FB}_z & \hat{FC}_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}. \quad (3-4)$$

3.2.2.2 Camera Translation

The rotation, once found, greatly simplifies the computation of the three translational components $T_x$, $T_y$, and $T_z$. Before these parameters can be uniquely determined, however, two additional pieces of information are necessary; in particular, the 3-D positions of two image points, defined as "anchors" in the GUI, must be specified. The first of these acts as a reference point, typically the origin in scene space, and gives both the camera and the 3-D model a frame of reference. The second provides the arbitrary scale factor necessary to eliminate the last degree of freedom in the system.

If no anchors are specified by the user, two are selected automatically. The longest edge in the image that has a known direction is found, and its endpoints are assigned to be the scene space origin and a point along the known direction that is scaled according to the edge's length. If only one anchor is specified, an attempt is made to find another: the longest edge attached to the given anchor with a known direction is found, if it exists, and the
edge's other endpoint is given a 3-D position along the known direction from the first anchor.

These two additional constraints provide enough information to solve for camera translation, since they give four equations that can be used to solve for the three unknowns \( T_x, T_y, \) and \( T_z \). These equations are derived in Appendix B.1.

### 3.3 Determination of Scene Geometry

Once the camera parameters are known, solving the scene's geometry becomes a much simpler problem. The process utilizes the now-known camera and all of the geometric constraints previously input by the user, such as coplanarity, parallelism, shared endpoints and vertices, and anchors. The goal here is to solve for the 3-D positions of all user-specified points, since the edges and surfaces are defined completely by these points.

With the chosen set of constraints, individual points can be solved by two main methods, described below. These methods are combined to determine as many unknown positions in the scene as possible.

#### 3.3.1 Using a Known Direction and Point

The first of these methods solves a partially determined edge. If an edge exists with a scene space axis direction (\( x, y, \) or \( z \)), and one of its endpoints is known, the other endpoint can be found analytically.

The unknown endpoint is constrained to lie on a line in the given direction that passes through the known endpoint. Only one degree of freedom thus remains for the unknown point, namely the distance along the infinite line. This can be found readily, since the projection of the on the image is also known. In fact the system is over-constrained, because there are two projection equations (one for each of the \( x \) and \( y \) coordinates of the image point) and only one unknown.
Figure 3-6: Solution From Known Point and Direction
Edge $E$ contains a known point and is parallel to the $z$ axis in scene space. The unknown point thus has the same $x$ and $y$ coordinates as the known point. A ray is drawn from the focal point through the projection of the unknown point onto the image plane; this ray's intersection with the infinitely-extended edge determines the unknown point's $z$ coordinate.

3.3.2 Using a Known Planar Surface

The second method of finding a point's 3-D position utilizes planar surface constraints imposed by the user. The plane's parameters are determined by examining the points and edges contained in the surface.

3.3.2.1 Planar Surface Determination

A plane, as discussed in Section 2.3.2, is defined by a normal direction and an offset from the origin. The normal can be determined from the surface's components in several ways, one of which is to examine the directions of edges contained in the surfaces. For example, if the surface contains edges in the $x$ and $y$ directions, its normal must be in the $z$ direction. The fourth parameter, the offset of the surface from the origin, can then be easily determined if the 3-D position of any of the points contained in the surface is known.

If this method is not sufficient—for example, if there is only one known edge direction present—a best-fit plane through the solved 3-D points in the surface can be found using a linear least-squares approach.
3.3.2.2 Finding a Point on the Plane

The plane equation in canonical form is linear in \( x, y, \) and \( z, \) as given by Equation 2-6. If a point is known to lie in this plane and its projection in the image is also known, the plane equation and the point’s two projection equations provide enough constraint to give a unique solution. See Appendix B.2 for a more detailed derivation.

Figure 3-7: Solution From Known Plane
A ray from the focal point through the projection of the unknown point onto the image plane is drawn. The intersection of this ray with the known planar surface gives the 3-D position of the unknown point.

3.3.3 Model Refinements

Discrepancies occur between different types of geometric constraints. For example, an unknown point could be specified to lie along a particular direction from a known point, and also to be coplanar with a known edge; depending on which of the two above methods is used to solve for the location of this point, however, a slightly different solution will be found.

The approach used here is thus to collect, for a given unknown point, the set of all equations relevant to its solution, from coplanarities, parallel directions, and projections onto the image. This set of equations, which are all linear, can be solved using a linear least-squares approach. This gives a “best fit” to the point’s actual location given the entire set of user constraints.
Chapter 4
Multiple-Frame Analysis

Builder has the ability to create composite geometric models based on information
gathered from multiple as well as from single images. This chapter discusses the data nec-
essary for creating a composite model and the way in which the user supplies this data. It
also outlines the methods used to find relative camera orientations between two frames
and solve for 3-D point locations across multiple images.

4.1 Data Input

The interface for entering multiple-image data is much the same as that for single
images. In fact, the individual frames of an image sequence are treated independently for
most operations. Points, edges, and surfaces are defined for each image, after which some
additional data is supplied.

4.1.1 Point Features

Possibly the most important data necessary for multiple-frame calculations are features,
which are special points identified across more than one image. Normal points, as
well as anchors, can be tagged as features by selecting them in Builder’s Make Feature
mode (see Figure 3-3).

Each feature has a unique name which distinguishes it from other features that may
exist in a given image data set; the name is input by the user when the feature is created.
Points in other images can then be assigned the same name, effectively identifying point
correspondences across images. At least four features must be shared by a given pair of images for their respective cameras to be rectified, as will be discussed in Section 4.2.

4.1.2 Transient Feature Identification

Most scenes involving multiple images contain objects that move in relation to the background. Such objects must be treated differently from static objects, since certain constraints and assumptions do not apply to them. It is thus useful to be able to identify moving objects from frame to frame. The user can do so by tagging feature points as mobile in Builder.

These mobile features are not considered in triangulation equations or camera rectification, and can be treated separately during scene rendering. Anchors may not be made mobile, since they are inherently assumed to be static points.

4.2 Camera Rectification

Since the user inputs data for each image independently, he may arbitrarily choose certain attributes. Thus, the anchor positions and edge directions in one image may not be consistent with those in another. This leads to problems in merging scene data: although the geometry of the images can be solved independently, their scene space coordinate frames may not line up with each other. For this reason, the cameras solved independently in the two images must be rectified so that the scene coordinate frames align and the geometry is consistent.

4.2.1 Parameters to be Solved

Anchor positions and edge directions do not affect the solution of intrinsic camera parameters (focal length and center of projection). Thus, only the extrinsic camera parameters must be rectified for coordinate frame consistency.

Two given frames may differ by at most by a translation, a rotation, and a scale, in arbitrary order. A single composite affine transformation is therefore enough to bring them
into coincidence. A generalized affine transformation representing these three operations, is given by the matrix

$$A = \begin{bmatrix}
A_{11} & A_{12} & A_{13} & A_{14} \\
A_{21} & A_{22} & A_{23} & A_{24} \\
A_{31} & A_{32} & A_{33} & A_{34} \\
0 & 0 & 0 & 1
\end{bmatrix}, \quad (4-1)$$

where the twelve entries $A_{ij}$ are the unknown rectification parameters.

Having defined this transformation, all that remains is to solve for the twelve unknowns. Since the matrix $A$ takes points from the world coordinate system of one frame to that of another, a single point correspondence introduces three linear equations in the twelve unknowns. If at least four such correspondences are well-chosen (i.e., not coplanar), this provides enough constraint to uniquely determine the matrix $A$. See Appendix B.3 for a more detailed derivation.

### 4.2.2 Rectification Method

The twelve unknown matrix parameters can only be found if at least four point correspondences exist between the two given images. This means in particular that the scene position of each feature is known in its respective non-rectified coordinate frame. The first step in the rectification process is therefore to find a solution for all relevant feature points in each frame of interest. This is done using the single-frame solution methods outlined in Section 3.3.

Once all features are solved, a set of linear equations in the twelve unknowns is constructed from the point pairs and solved using a least-squares method. The newly-solved matrix $A$ is then used to transform the anchors and scene directions in the second frame to the first frame so that they are consistent.

Finally, the extrinsic parameters of the second image's camera are solved using the newly-corrected anchors. The two frames should now be, in a least-squares sense, as close to each other as possible.
4.3 Augmenting the Model

After all cameras are rectified and their coordinate frames in coincidence, the scene geometry is solved again over all images. Additional information is used for point solutions this second time.

4.3.1 Utilizing Point Correspondences

A new type of constraint aside from parallelism and coplanarity can now be exploited, namely feature correspondence across multiple images. A sort of “triangulation” can be performed, allowing points to be solved from correspondence information alone.

If all intrinsic and extrinsic parameters are known for each camera, then correspondence between two images is more than enough information to solve the 3-D position of a given feature point. The basic idea is to draw a ray through the focal point of each camera and the point’s projection onto its image plane; the intersection of all such rays is the point’s actual position in space. Figure 4-1 shows this concept more clearly.

![Image Plane of First Camera](image1)

![Image Plane of Second Camera](image2)

![Triangulated 3-D Point](image3)

![Focal Point of First Camera](image4)

![Focal Point of Second Camera](image5)

Figure 4-1: Three-Dimensional Feature Triangulation
A ray is drawn from the focal point of each camera through the point’s projection on that camera’s image plane; the intersection of all such rays is the feature’s location in space. The simplest two-camera case is shown here.

The two projection equations (Equation 2-13) for each point are collected and solved using linear least-squares, which yields the average intersection of all triangulation
rays. The set of projection equations is added to the set of existing single-image constraints for a given point before it is actually solved.

4.3.2 Mobile Features

If a feature is tagged as Mobile, the triangulation equations are not added to the single-image constraints. Such points have been identified by the user as transient, and thus may move in relation to the established scene coordinate frame from image to image. It is therefore useless to attempt a triangulation on these points, since their motion between frames is not known in advance. However, once such points are solved using the other constraint types, their motion can be determined simply by calculating the difference in 3-D position from one frame to another.
Chapter 5
The Renderer

Renderer is a GUI application that allows three-dimensional scenes to be viewed interactively. Renderer produces wireframe or fully texture-mapped images based on geometry and texture data read from previously generated scene files. The final texture-rendered images can also be stored on disk for later viewing.

A discussion of the rendering process is presented first, followed by descriptions of the user interaction methods.

5.1 Determination of Surface Polygons

When the scene information is read from disk, a list of surfaces is created. Each surface, upon creation, contains a list of the points and edges that define it, which alone is not enough information for texture rendering. It is thus necessary to find a polygonal approximation to that portion of each surface which is visible in its original image.

5.1.1 Convex Outline

As a first-pass approach to finding the desired visible surface patch, a convex outline that surrounds all points contained in the surface is computed. Since the only data available is an unordered list of points, a search must be performed to find the best such outline. The basic algorithm follows here.

First, infinite lines are generated through all pairs of points until one is found such that all other points lie to its left. The point pair that generated this line is then added to the
polygon as denoting its first edge. Next, infinite lines are generated through the endpoint of this first segment and all other points until again one is found such that all other points lie to its left. The new point is appended to the polygon, and this process continues until the polygon closes itself, i.e. until the new point found is the first point in the polygon. This process is depicted more clearly in Figure 5-1.

![Figure 5-1: Finding a Convex Outline](image)

A simple four-sided polygon is found around an unordered list of points. In (a), some of the possible initial edge candidates are shown, and (b) shows the chosen initial edge along with the candidates for the next edge. The remaining choices and candidates are shown in (c) and (d), while (e) shows the final convex bound polygon. Note that edges are always chosen so that all remaining points lie to their left.

Now, we have an ordered list of points that represents the convex bounding polygon around all the other points. Since "left" and not "right" was chosen as the criterion for whether points are inside the polygon, the points are in counterclockwise order around it, which by design is consistent with the way polygon normals are computed for rendering.

To determine whether a point is to the left of an infinite line, we must first define what exactly is meant by "left". The line has a direction, namely the vector formed by subtracting its first endpoint from its second endpoint. The normal to this vector is defined to be perpendicular and to the left of its heading, and will be denoted as \( N \). Now we define \( V \) as the vector formed by subtracting either endpoint from the point to be tested. The candidate point is then to the left of the line if the vector dot product \( V \cdot N \) is positive.
Figure 5-2: Inside and Outside a Polygon Edge
The polygon edge is the segment from \( E_1 \) to \( E_2 \); a directed infinite line is defined which passes through these two points, shown as a dotted arrow. The edge normal \( N \) is perpendicular to the edge and points to its left toward the interior of the polygon. Vectors \( V_1 \) and \( V_2 \) are formed by subtracting \( E_1 \) from candidate points \( P_1 \) and \( P_2 \), respectively. \( P_1 \) is inside the polygon (to the left of the edge) because the dot product of \( V_1 \) with \( N \) is positive. \( P_2 \) is outside (to the right of the edge) because the dot product of \( V_2 \) with \( N \) is negative.

5.1.2 Concave Outline

The surface data structure contains edge information as well as a list of points. Edges can provide more information about the visible surface patch, allowing a possibly concave bound to be created that is more precise than the convex one.

It can be shown that any concave bound around all points is a superset of the convex bound from Section 5.1.1. In other words, an arbitrary boundary polygon contains, at the least, all points in the convex bound. However, there are many boundary polygons that can be formed from the same unordered point list.

If the user has specified surface edges in such a way that they form a closed polygon surrounding all other points, this more accurate boundary can be found automatically. First, any point from the convex polygon is added to the new polygon point list, since the new polygon must contain it. Then, all the surface's edges which contain this point as an endpoint are searched. The other endpoint of such an edge forms an infinite line against which the remaining points can be tested. That edge is selected which has more of these points to its left than any other, and its endpoint is added to the new polygon. All the surface's edges containing this new vertex are then checked, and so on.
If the polygon closes, i.e. if the best vertex found is the starting point, and if the newly-formed polygon contains all points in the convex bound, it is chosen as the representative surface polygon. Otherwise, the convex polygon is used.

5.2 Rendering Surface Polygons

This section describes the process by which the outline of the model is rendered. Display of the model follows the standard render pipeline (Figure 5-3). First, each polygon is transformed into a canonical view space. The transformation that takes points in the world coordinate frame into this space normalizes the view volume. Under parallel projection, the view volume becomes a 2 by 2 by 1 rectangular solid, and under perspective projection it becomes a pyramid of height 1 whose sides have unit slope in the x-z and y-z planes. (Figure 5-4).

Figure 5-3: Render Pipeline
The four main stages of the render pipeline. First, objects are translated, rotated, and scaled into a normalized VRC space. Next, they are clipped to the planes defining the view volume. Finally the objects are warped by the perspective transformation and scaled to device (pixel) coordinates.
Figure 5-4: Canonical View Volumes
Under parallel projection, shown in (a), the canonical view volume is a 2 by 2 by 1 rectangular solid. Under perspective projection, shown in (b), it is a pyramid whose sides have unit slope.

Next, the polygon is clipped to the planar edges of the canonical view volume in the new space, eliminating the portions that are outside it, i.e. those which are not visible or are too close or far away. This is performed using the Cyrus Beck polygon clipping algorithm [3].

The polygon is then warped according to the perspective projection equations if applicable, and transformed into display coordinates. In most cases the actual display window has its origin in the upper left corner rather than the center, with positive x to the right and positive y directed downward; a combination of translation and scale transformations must thus be performed. The final x and y values represent the position in the image window to which each vertex should be drawn.

5.3 Rendering Surface Textures

Once the polygonal outline is in its final clipped and rendered form, it can be drawn to the image buffer. If surface textures are desired, however, more work must be done to produce the final image. This section describes the process by which surface polygons are texture-rendered to the image buffer.
5.3.1 Polygon Scan Conversion

The process by which polygons are filled and drawn to a discrete (pixel-based) buffer is known as polygon scan conversion. Possibly the most common scan conversion algorithm intersects a series of horizontal scan lines with the edges of the polygon to determine the spans of pixels that should be filled in.

![Figure 5-5: Polygon Scan Conversion](image)

*Figure 5-5: Polygon Scan Conversion*

Drawing a filled polygon to a pixel-based buffer. Horizontal scan lines, two of which are shown here, are intersected with the edges of the polygon to determine the horizontal spans that should be drawn. Grid intersections represent pixel positions, and filled pixels are shown as grey dots.

The scan conversion process is not limited to simply filling polygons with a solid color, however. It is also used to generate depth buffers and to render textures, and in fact can be used to perform arbitrary pixel-by-pixel computations on a given polygon. The scan conversion process simply determines which pixels should be operated upon.

5.3.2 Visible Surface Determination

Three-dimensional polygons can obscure one another when viewed from different perspectives. It is thus necessary to determine which portions of each polygon are actually visible from a given viewpoint.

Many algorithms exist to do this. One particularly simple method is called the z-buffer or depth buffer algorithm. This method is pixel-based rather than object based; in other words, it operates on rendered pixels rather than ideal 3-D primitives. The basic operation of the algorithm is described here.
A separate z-buffer of the same dimensions as the original image is created. Each point in the buffer can be assigned an arbitrary floating-point decimal value, which represents the distance from the viewer of the planar surface seen through that point. The buffer is initially filled with the maximum possible positive number, indicating that every point is "infinitely" far away from the viewer. When the algorithm is complete, the z-buffer contains the depths of all visible portions of each surface in the scene.

Each polygonal surface bound is rendered and scan-converted to the z-buffer line by line. At every point in the scan conversion, the x and y coordinates correspond to the position of the point in the z-buffer, and a z coordinate is computed which represents the depth of the plane at that point. The z coordinate is found using the plane equation given by Equation 2-6, namely

\[ Ax + By + Cz + D = 0. \]  
(5-1)

Solving for z, since all other parameters at a given point are known, we have

\[ z = \frac{-Ax + By + D}{C}. \]  
(5-2)

Once the depth of a given pixel is computed, it is compared with the value already stored in the z-buffer at that point. If the existing value is less than the new value, then the new value is discarded; otherwise, the new value replaces the old one. Each polygon is scan converted in this way until, in the end, the z-buffer contains the depths of all visible surface pixels. Figure 5-6 depicts a simple z-buffer generated from two cubes.
5.3.3 Texture Mapping

Texture mapping is the process of projecting a 2-D image onto a 3-D planar surface during rendering. Flat-shaded or wireframe objects can be given more interesting and accurate surface properties by performing a direct mapping from this image to the desired surface polygon. The method of texture rendering used here is somewhat specialized to this application.

5.3.3.1 Problems with Traditional Texture Maps

Traditional texture rendering involves using previously-generated texture maps, which are images that contain a pre-computed “head-on” view of the textures. The rectangular boundary of each texture represents the outline of the planar surface patch onto which it is mapped, and during rendering these static images are projected onto their respective surfaces.

In most cases, this method of texture mapping is sufficient. Several problems arise, however, when the textures are generated from polygons in real images, especially those with a large degree of perspective foreshortening.
The first disadvantage of traditional texture maps, as it applies to the problems posed here, is the extra space required for storing their data. Each texture is a full-color image whose size is at least that of the polygon in the original image which generated it. In general, texture sizes are significantly greater, since the original image information is stored in “unwarped” form in the texture. The texture image is stored as it would be seen head-on, rather than from the perspective it is viewed in the original image.

This can cause serious storage problems, especially in cases where the surface in the original image is viewed from a “glancing” or nearly edge-on angle. When the perspective transformation is undone, the texture boundary can be arbitrarily large even though it contains very little data (see Figure 5-7).

![Figure 5-7: Unwarping a Texture Boundary](image)

Transforming a texture from its original image form in (a) to a canonical head-on view in (b). The canonical view is notably larger, requiring more storage space to maintain all pixel information. As the view becomes more edge-on, the information content of the texture decreases while the area needed to store a canonical view increases.

The second main problem with traditional texture maps is that they are static. Different views of the same texture are usually blended together into a final texture map, which is then rendered. This can result in a loss of resolution over various parts of the texture and a loss of flexibility in how multiple-view textures are rendered.
5.3.3.2 Dynamic Texture Mapping

These two problems are solved by performing dynamic texture mapping at render time. Instead of pre-computing separate texture buffers and storing them in canonical form, only pointers to the original images are stored. Since these images were used to create geometric constraints on the scene geometry and are already stored on disk, there is no need to create additional texture maps. All relevant information is contained in the images.

The camera for each image has already been solved. Thus the transformation that takes points in image space to points in scene space is known, and texture pixels can be taken directly from the image and mapped onto the scene space objects at render time. Pixels from multiple views of the same textured surface can be arbitrarily combined so that the rendering process is flexible and less susceptible to resolution loss.

Dynamic texture mapping, however, does have its drawbacks, the most significant of which is speed. Blending and rendering textures dynamically is on average slower than rendering pre-computed texture maps. In fact the speed decreases with each additional view of a texture, while static texture maps can be drawn in more or less constant time.

5.3.3.3 Implementation of Texture Rendering

The first step in this implementation of dynamic texture rendering is to determine which pixels in the original images belong to which surfaces. A z-buffer is created for each original image, along with a so-called “tag buffer” of the same dimensions which contains, at each pixel position, an identification number tagging that pixel as belonging to a specific surface.
Figure 5-8: Tagging Surfaces for Dynamic Texture Rendering
The z-buffer (a) and tag buffer (b) from the original camera’s viewpoint. Each surface in the tag buffer is marked here with a different shade of gray.

After this step, several new buffers are created, all of identical size. These are the output buffer, which will contain the final texture-rendered scene, a z-buffer for the new view, and a so-called “hit-buffer”. The surface polygons are scan converted again, this time from the new camera’s viewpoint. Several operations are performed for every pixel in the scan conversion.

Figure 5-9: Flowchart for Determining Pixel Color
Each polygon is scan converted to the desired view, and the depicted steps are followed for each pixel in the scan conversion.

- The pixel’s z value is computed based on its x and y values and on the plane equation for its surface (Section 3.3.2.1).
- This z value is compared with the value stored at the same (x, y) position in the new z-buffer. If it is greater than the old value, it is discarded because it is obscured by another object, and the process repeats for the next pixel. If not, the
new z value replaces the old one and the next step is performed.

- Since the pixel's x, y, and z coordinates are now given, and since both the current view's camera and the original image's camera are known, the pixel is transformed to the original camera's coordinate frame.

- The transformed (x, y) position of the pixel gives its position in the original image and in the tag buffer. The identification number of the pixel's surface is compared with the value stored in the tag buffer at that point. If the numbers match, the next step is performed. Otherwise, the pixel was not visible from the original viewpoint and is ignored.

- Finally, the color of the pixel is looked up in the original image and blended into the output buffer at the pixel's original location. This blend is performed using the hit-buffer, which contains the number of pixels drawn to every output buffer location. The value stored in the hit-buffer is incremented, and the process repeats for the next pixel in the scan-conversion.

![Buffers for Desired Textured View](image)

**Figure 5-10: Buffers for Desired Textured View**
The z-buffer for the desired camera view is shown in (a); the output buffer containing the final texture-rendered surface polygons is shown in (b).

After all polygons have been scan-converted in this way, the output buffer contains the blended, texture-rendered image.
5.4 Manipulating the View

The user has a wide variety of control over how the scene is viewed. Several methods are provided to manipulate the camera's position and other parameters.

![Renderer Display Window](image)

**Figure 5-11: Renderer Display Window**
The main display window of the Renderer application. A simple wireframe version of the scene is rendered in (a), showing the points and edges previously specified by the user. In (b), the fully texture-mapped version is displayed.

5.4.1 Direct Camera Controls

From the Camera menu, the user can save the current camera to a file, open an existing camera file, and create a dialog box that contains controls to directly modify camera parameters.

The dialog box, shown in Figure 5-12, allows the user to vary all appropriate camera attributes, including the VRP, VPN, VUP, center of projection, virtual window size, display window size and depth, and clip plane distances. The camera can also be given a name and projection type. The changes are not committed until the user presses the "OK" button, which dismisses the window and updates the view.
Figure 5-12: Camera Dialog Box

This window contains controls for manipulating all camera parameters. Numbers can be typed in the various fields, and the camera's name and projection type can be altered. When the Apply button is pressed, all changes are applied and the scene is rendered from the new camera. When changes are made interactively to the view, the parameters in this window are updated.

5.4.2 Interactive Viewing

Several camera parameters can also be altered using the mouse in the image display window. The three mouse buttons, along with drag directions, determine how the viewpoint is changed.

The right and middle mouse buttons control camera translation. By holding down the right button and dragging, the user can translate the camera in the horizontal and vertical screen directions. The middle button allows forward and backward translation along the z screen axis, with up and down mouse motions, respectively.

The left mouse button controls camera rotation. The entire VRC coordinate frame is rotated about the "center of mass" of the objects, which is found by averaging all x, y, and z point values in the scene. Rotation is specified by an axis and an angle, which are both determined from the drag vector, or the vector formed by subtracting the mouse's start position from its end position.

Angles are normalized so that a drag distance equal to the diagonal size of the screen window corresponds to a rotation of 180 degrees. The ratio of drag distance to
diagonal window size is thus multiplied by 180 to obtain the angle of rotation. The axis about which the VRC frame is rotated is always parallel to the view plane, and is taken to be the two-dimensional normal (in the view plane) to the drag vector.

5.4.3 View Menu Controls

The View menu provides the user with control over how the scene is displayed. Textures can be enabled and disabled, as can scene element icons; thus, when the user releases a mouse button after changing the viewpoint, or after manually altering camera parameters, textures and element symbols can either be drawn or ignored. Texture-rendered images can also be saved to disk via the File menu.

The Lock Aspect menu item allows the aspect ratio of the view window to be fixed, meaning that if the user changes the size of the window, the ratio between height and width is preserved.
Chapter 6
Experimental Results

Several sets of images were processed using the Builder and Renderer tools. The first of these consists of synthetically-generated 3-D scenes whose camera and geometry are known, and whose purpose was to aid in obtaining a measure of the tools' performance. Another image was produced by digitizing a photograph of an outdoor scene, and was used for single-frame analysis. The final set of images consists of digitized photographs taken indoors and was used for multiple-frame analysis. Scenes were synthesized for all data sets; the results are presented here.

6.1 Synthetic Images

The synthetic scene consisted simply of two cubes, and was rendered from different views to generate the three images shown below. The dimensions and positions of the cubes are of course known, as are the parameters for all three cameras.
Each image was analyzed individually first, after which all three were merged into one scene database and analyzed simultaneously.

### 6.1.1 Single-Frame Performance

Intrinsic and extrinsic camera parameters, as well as scene geometry, were first calculated for each view independently.

#### 6.1.1.1 Camera Calibration Results

Numerical results for camera parameters are presented here, and compared with the known ideal parameters.

### Table 6-1: Single-Frame Camera Accuracy of Synthetic Image A

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual</th>
<th>Calculated</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focal Length</td>
<td>200.0</td>
<td>200.294</td>
<td>0.294</td>
</tr>
<tr>
<td>COPx</td>
<td>0.0</td>
<td>-2.072</td>
<td>-2.072</td>
</tr>
<tr>
<td>COPy</td>
<td>0.0</td>
<td>-5.994</td>
<td>-5.994</td>
</tr>
<tr>
<td>VRPx</td>
<td>170.0</td>
<td>177.05</td>
<td>7.05</td>
</tr>
<tr>
<td>VRPy</td>
<td>170.0</td>
<td>173.591</td>
<td>3.591</td>
</tr>
<tr>
<td>VRPz</td>
<td>170.0</td>
<td>167.611</td>
<td>2.389</td>
</tr>
<tr>
<td>VPNx</td>
<td>-0.57735</td>
<td>-0.5711</td>
<td>0.00625</td>
</tr>
<tr>
<td>VPNy</td>
<td>-0.57735</td>
<td>-0.56508</td>
<td>0.01227</td>
</tr>
<tr>
<td>VPNz</td>
<td>-0.57735</td>
<td>-0.59542</td>
<td>-0.01807</td>
</tr>
<tr>
<td>VUPx</td>
<td>-0.40825</td>
<td>-0.42017</td>
<td>-0.01192</td>
</tr>
</tbody>
</table>
### Table 6-1: Single-Frame Camera Accuracy of Synthetic Image A

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual</th>
<th>Calculated</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>VUP&lt;sub&gt;y&lt;/sub&gt;</td>
<td>-0.40825</td>
<td>-0.4219</td>
<td>-0.01365</td>
</tr>
<tr>
<td>VUP&lt;sub&gt;z&lt;/sub&gt;</td>
<td>0.8165</td>
<td>0.8034</td>
<td>-0.0131</td>
</tr>
</tbody>
</table>

### Table 6-2: Single-Frame Camera Accuracy of Synthetic Image B

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual</th>
<th>Calculated</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focal Length</td>
<td>150.0</td>
<td>151.138</td>
<td>1.138</td>
</tr>
<tr>
<td>COP&lt;sub&gt;x&lt;/sub&gt;</td>
<td>0.0</td>
<td>-2.54</td>
<td>-2.54</td>
</tr>
<tr>
<td>COP&lt;sub&gt;y&lt;/sub&gt;</td>
<td>60.0</td>
<td>58.419</td>
<td>-1.581</td>
</tr>
<tr>
<td>VRP&lt;sub&gt;x&lt;/sub&gt;</td>
<td>170.0</td>
<td>173.665</td>
<td>3.665</td>
</tr>
<tr>
<td>VRP&lt;sub&gt;y&lt;/sub&gt;</td>
<td>250.0</td>
<td>257.592</td>
<td>7.592</td>
</tr>
<tr>
<td>VRP&lt;sub&gt;z&lt;/sub&gt;</td>
<td>150.0</td>
<td>152.948</td>
<td>2.948</td>
</tr>
<tr>
<td>VPN&lt;sub&gt;x&lt;/sub&gt;</td>
<td>-0.40825</td>
<td>-0.39342</td>
<td>0.01482</td>
</tr>
<tr>
<td>VPN&lt;sub&gt;y&lt;/sub&gt;</td>
<td>-0.8165</td>
<td>-0.8173</td>
<td>-0.0008</td>
</tr>
<tr>
<td>VPN&lt;sub&gt;z&lt;/sub&gt;</td>
<td>-0.40825</td>
<td>-0.421</td>
<td>-0.01275</td>
</tr>
<tr>
<td>VUP&lt;sub&gt;x&lt;/sub&gt;</td>
<td>-0.18257</td>
<td>-0.19039</td>
<td>-0.00782</td>
</tr>
<tr>
<td>VUP&lt;sub&gt;y&lt;/sub&gt;</td>
<td>-0.36515</td>
<td>-0.37558</td>
<td>-0.01043</td>
</tr>
<tr>
<td>VUP&lt;sub&gt;z&lt;/sub&gt;</td>
<td>0.91287</td>
<td>0.90702</td>
<td>-0.00585</td>
</tr>
</tbody>
</table>

### Table 6-3: Single-Frame Camera Accuracy of Synthetic Image C

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual</th>
<th>Calculated</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focal Length</td>
<td>200.0</td>
<td>199.274</td>
<td>-0.726</td>
</tr>
<tr>
<td>COP&lt;sub&gt;x&lt;/sub&gt;</td>
<td>-30.0</td>
<td>-32.947</td>
<td>-2.947</td>
</tr>
<tr>
<td>COP&lt;sub&gt;y&lt;/sub&gt;</td>
<td>10.0</td>
<td>10.334</td>
<td>0.334</td>
</tr>
<tr>
<td>VRP&lt;sub&gt;x&lt;/sub&gt;</td>
<td>-100.0</td>
<td>-80.204</td>
<td>19.796</td>
</tr>
<tr>
<td>VRP&lt;sub&gt;y&lt;/sub&gt;</td>
<td>170.0</td>
<td>171.758</td>
<td>1.758</td>
</tr>
<tr>
<td>VRP&lt;sub&gt;z&lt;/sub&gt;</td>
<td>120.0</td>
<td>125.024</td>
<td>5.024</td>
</tr>
<tr>
<td>VPN&lt;sub&gt;x&lt;/sub&gt;</td>
<td>0.6931</td>
<td>0.66675</td>
<td>-0.02635</td>
</tr>
<tr>
<td>VPN&lt;sub&gt;y&lt;/sub&gt;</td>
<td>-0.19803</td>
<td>-0.184</td>
<td>0.01403</td>
</tr>
<tr>
<td>VPN&lt;sub&gt;z&lt;/sub&gt;</td>
<td>-0.6931</td>
<td>-0.72222</td>
<td>-0.02912</td>
</tr>
<tr>
<td>VUP&lt;sub&gt;x&lt;/sub&gt;</td>
<td>0.66644</td>
<td>0.69406</td>
<td>0.02762</td>
</tr>
<tr>
<td>VUP&lt;sub&gt;y&lt;/sub&gt;</td>
<td>-0.19041</td>
<td>-0.1998</td>
<td>-0.00939</td>
</tr>
<tr>
<td>VUP&lt;sub&gt;z&lt;/sub&gt;</td>
<td>0.72084</td>
<td>0.69164</td>
<td>-0.0290</td>
</tr>
</tbody>
</table>
6.1.1.2 Point Estimation Results

Each view of the cube was used to generate a 3-D model. Since the actual locations of the corner points were known, they could be compared with the computed locations for error evaluation. The following figures gauge the single-frame geometry calculation performance.

![Graphs showing point error](image)

**Figure 6-2: Synthetic Single-Image Point Error**
The error in estimating 3-D point positions in all individual cube models. The $x$-$y$, $y$-$z$, and $x$-$z$ projections of the errors are shown in (a), (b), and (c), respectively. The sizes of the cubes are 300 and 150 units.

![Synthetic images](image)

**Figure 6-3: Single-Frame Renders of Synthetic Images**
Each original synthetic image as rendered from a new camera. The rendered images (a), (b), and (c) here correspond to the original images A, B, and C.
6.1.2 Multiple-Frame Performance

The three cube views were composited into a single model using the camera rectification methods described in Section 4.2 and triangulation of point correspondences. The results are presented here.

6.1.2.1 Relative Camera Orientation

The tables below compare the calculated parameters of the composited images with the actual, known values. The results for the first image are not presented here, since they are identical to those in the single-frame case. In fact only the extrinsic parameters are listed, since intrinsic parameters are not affected by camera rectification.

Table 6-4: Camera Accuracy of Synthetic Image B After Rectification

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual</th>
<th>Calculated</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRP(_x)</td>
<td>170.0</td>
<td>175.476</td>
<td>5.476</td>
</tr>
<tr>
<td>VRP(_y)</td>
<td>250.0</td>
<td>254.872</td>
<td>4.872</td>
</tr>
<tr>
<td>VRP(_z)</td>
<td>150.0</td>
<td>151.071</td>
<td>1.071</td>
</tr>
<tr>
<td>VPN(_x)</td>
<td>-0.40825</td>
<td>-0.39342</td>
<td>0.01483</td>
</tr>
<tr>
<td>VPN(_y)</td>
<td>-0.8165</td>
<td>-0.8173</td>
<td>-0.0008</td>
</tr>
<tr>
<td>VPN(_z)</td>
<td>-0.40825</td>
<td>-0.421</td>
<td>-0.01275</td>
</tr>
<tr>
<td>VUP(_x)</td>
<td>-0.18257</td>
<td>-0.19039</td>
<td>-0.00782</td>
</tr>
<tr>
<td>VUP(_y)</td>
<td>-0.36515</td>
<td>-0.37558</td>
<td>-0.01043</td>
</tr>
<tr>
<td>VUP(_z)</td>
<td>0.91287</td>
<td>0.90702</td>
<td>-0.00585</td>
</tr>
</tbody>
</table>

Table 6-5: Camera Accuracy of Synthetic Image C After Rectification

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual</th>
<th>Calculated</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRP(_x)</td>
<td>-100.0</td>
<td>-80.355</td>
<td>-19.645</td>
</tr>
<tr>
<td>VRP(_y)</td>
<td>170.0</td>
<td>171.986</td>
<td>1.986</td>
</tr>
<tr>
<td>VRP(_z)</td>
<td>120.0</td>
<td>125.258</td>
<td>5.258</td>
</tr>
<tr>
<td>VPN(_x)</td>
<td>0.6931</td>
<td>0.66675</td>
<td>-0.02635</td>
</tr>
<tr>
<td>VPN(_y)</td>
<td>-0.19803</td>
<td>-0.184</td>
<td>0.01403</td>
</tr>
<tr>
<td>VPN(_z)</td>
<td>-0.6931</td>
<td>-0.72222</td>
<td>-0.02912</td>
</tr>
<tr>
<td>VUP(_x)</td>
<td>0.66644</td>
<td>0.69406</td>
<td>0.02762</td>
</tr>
<tr>
<td>VUP(_y)</td>
<td>-0.19041</td>
<td>-0.19979</td>
<td>-0.00938</td>
</tr>
<tr>
<td>VUP(_z)</td>
<td>0.72084</td>
<td>0.69164</td>
<td>-0.02920</td>
</tr>
</tbody>
</table>
6.1.2.2 Planar Surface Determination

The results of compositing textures from all three views are shown here. The edge boundaries are not clearly defined because of error in camera rectification; surface textures thus “bleed” into one another.

![Figure 6-4: Render of Composited Cube Images](image)

Two different views of the three composited cube images. Note that (a) shows cube faces simultaneously which do not exist in any single image.

6.2 Outdoor Scene

The image used for single-frame analysis was an aerial photograph of the Chicago Art Institute, shown in Figure 6-5. A simple model including the ground, the building’s major surfaces, and the small booth at the bottom of the photograph was created. The results are depicted below.
Figure 6-5: Original Photograph for Single-Frame Analysis
A photograph of the Chicago Art Institute was used to generate a single-frame model. The building's front, sides, roof, and entrance were specified as surfaces, as were the ground and the sides of the small booth on the sidewalk.

In Figure 6-6(a), the building is viewed from a more drastic perspective. The low image resolution of the booth can be seen as pixelization. In Figure 6-6(b), part of the main building's roof is missing because it was obscured in the photograph. The objects on the ground appear flattened because the ground was modeled as a simple planar surface.

Figure 6-6: Synthetic Views of Outdoor Scene
These two views were generated by creating a model from a single photograph and then rendering the results from two different cameras.
6.3 Indoor Scene

Two photographs, shown in Figure 6-7, were taken indoors and composited into a single model.

![Figure 6-7: Original Photographs for Multiple-Frame Analysis](image)

Two photographs taken indoors from two different perspectives. The photographs were composited into a single model.

Different views of the calculated model are shown below. Errors in camera rectification due to too few point correspondences can be seen in Figure 6-8(b).

![Figure 6-8: Synthetic Views of Indoor Scene](image)

Two views of the composited model of the indoor scene. Information from both original images is present in the output images.
Chapter 7
Conclusions

Graphical user interfaces and computational components of the proposed system were designed using mathematical tools and synthetic data to simulate parameter solution methods. When these methods were shown to work correctly, they were integrated into the system, which was then tested thoroughly using many different images and image sets.

Although the techniques that were chosen and implemented worked well for his application, there are many improvements to be made and areas that could be expanded upon. This chapter assesses the overall operation of the system and presents some possible improvements.

7.1 Technique Assessment

Overall, the techniques used proved effective and satisfied the goals in the project specification. The objectives of creating 3-D models from sets of 2-D images, identifying moving objects, and interactively texture-rendering the final scene data were met, and the system produced satisfactory results, as seen in the previous chapter.

7.1.1 User Interface

The graphical user interface was kept as simple as possible for the given constraints, and was intuitive and straightforward to operate. Several features not required by the original project goals were added to the applications, such as the ability to query point information and save a texture rendered image to disk. Geometric constraints took only a
few minutes to specify using Builder’s input features, and Renderer allowed solved scenes to be viewed and navigated interactively.

7.1.2 Analysis Methods

Camera parameters in some images were very sensitive to inaccuracies in edge specification. However, the calculated scene geometry was consistent with any deviations in these parameters, and usually projected correctly when rendered. Coplanarity and parallelism constraints were generally not met, however, since a least-squares approach was used to solve for point locations and aggregate solutions were found.

7.1.3 Texture Rendering

The dynamic texture rendering methods proved to work well. Separately-stored texture images were unnecessary; texture data was efficiently stored in the original images and retrieved as needed. The simple blending methods for textures from multiple views produced satisfactory results, but the images were not always aligned properly and the blends sometimes produced “ghosting” effects due to projection error. The blend method could easily be changed, however, since dynamic textures allow for arbitrary compositing.

Another disadvantage of this rendering method was speed: although adequately fast, it was unable to produce fully textured images in real time, taking several seconds to render a moderately complex scene. Wireframe models, however, could be viewed arbitrarily quickly.

7.2 Future Directions

Although this project provides working solutions to the problems presented, there are still many areas that could be improved upon. These include enhancements to both the user interface and the analysis tools.

7.2.1 Interface Improvements

The most significant and useful change to the user interface would be to further automate the scene element input process. An intelligent edge detector could be run on the
images to be analyzed, automatically finding edge boundaries. Also, lines could be
grouped into parallel sets automatically based on their position and slope. The user would
still be able to refine errors in the parallel sets and detected lines if necessary.

Other possible improvements include an intelligent “snap” feature in Builder’s
data input mode. When the user clicks to add a point to the image data, the closest “cor-
er” in the image could be found using a local gradient or a Laplacian operator and the
new point drawn to that corner. Similarly, when an edge is created, it could automatically
snap to the closest automatically-detected edge in the image.

Since most objects being modeled are based on solid 3-D shapes rather than indi-
vidual planar surfaces, the user could be given a set of solid primitives to start with and be
able to refine them to fit the image. Corners of the solid shape could be moved by the user
to match their image locations. This would provide automatic constraints on edges and
points, since, for example, a rectangular solid has many geometric properties, such as par-
allel and orthogonal edges, which are particularly convenient to the point solution process.

One final improvement is standardization of the system. The user interface for the
analysis tools currently only runs on UNIX platforms, but could easily be adapted to other
platforms such as Microsoft® Windows. Porting to a platform-independent, internet-
based language like Java would open a vast set of new applications, since a huge database
of images is accessible to millions of users through the Worldwide Web (WWW). Virtual
worlds and maps based on real images could be built and navigated quickly and easily, and
models output to Virtual Reality Markup Language (VRML) or any other standard format
for 3-D data. They could be stored locally or in a central database, and viewed interac-
tively using standard renderers.

7.2.2 Analysis Improvements

Many improvements can be made to the analysis tools in the system’s computation
layer. The changes presented here focus on making the system more generalized and more
accurate.
7.2.2.1 Generalizing Camera Calibration

Presently, camera calibration relies on the fact that three vanishing points exist in the image to be analyzed. Unfortunately, many photographs contain only two or, in some cases, one principal vanishing point. It is thus desirable to devise a calibration system similar to the one implemented here which can also handle fewer than three vanishing points.

In the case of one such point, two of the scene space axes are parallel to the two image line sets that do not form a vanishing point, and the other axis is perpendicular to the view plane. The center of projection is taken to be the vanishing point itself, but the focal length cannot be computed without an additional constraint. In the case of two principal vanishing points, the focal point is constrained to lie on the semicircle that orthogonally cuts the view plane and whose diameter is defined by the two points. Two of the scene space axes are the vectors from the focal point to each of the two vanishing points, and the third, which is parallel to the view plane, is found by taking the appropriate vector cross product of the first two.

![Figure 7-1: Camera Calibration for One and Two Vanishing Points](image)

Only one vanishing point exists in (a). The camera orientation is determined, but the focal point lies at an unknown position along the ray drawn through the vanishing point perpendicular to the image plane. In (b) there are two vanishing points; the focal point must lie on the dotted circle. The top view shows the dependence of the camera's orientation on the position of the focal point.

In both cases, at least one additional constraint is required to uniquely determine the position of the focal point. This constraint could be obtained by using 3-D anchor information from other images, or could be chosen arbitrarily if appropriate.
7.2.2.2 Other Generalizations

Besides expanding camera calibration, other improvements can be made to generalize the analysis tools. For example, some amount of surface specification can be automated by utilizing connectivity of points and edges. A given point is attached to one or more edges; if it is attached to edges parallel to two different scene space directions, those edges form a planar surface to which all other appropriate edges and points can be added automatically. The user would need only augment the surface data by adding coplanar points not attached to any edges.

Other types of constraints could be introduced to handle certain special cases of geometric configurations. For example, if an edge is parallel to a known plane and one of its endpoints is known, but the edge is not parallel to a principal axis, its other endpoint can still be solved. Two constraints on the unknown point come from its two projection equations, and the other comes from the fact that its distance from the known plane is equal to that of the known endpoint.

7.2.2.3 Improving Estimation Accuracy

Because points are solved mainly by a linear least-squares method that incorporates all possible constraints, they deviate from these constraints so that in general none are met exactly. For example, a point is over-constrained if it is specified to lie on edges parallel to both the x and y scene space axes. A solution will be found that is as close as possible, in a least-squares sense, to satisfying both constraints, but in general the solved point will meet neither constraint and will lie somewhere between the two.

This phenomenon produces scenes which are geometrically accurate in a gross sense, but not very precise. Coplanarity and parallelism should be stronger constraints on points than their user-specified location in the image. A point’s attached edges and shared planar surfaces should be weighted more heavily than its projection on the image plane in the geometry computations. One way to implement this is to perform a weighted least-squares solution of the constraint system.

Another problem with the existing system is that errors in point solutions are propagated to subsequently solved points. Every point is solved on the assumption that previously solved points are exactly correct, which is not the case. A confidence measure could
be used to compensate, so that points analyzed later have a lower confidence measure than those solved first. Constraints involving these points could then be weighted according to this measure.

Error measures and iteration could also be used to make both scene geometry and camera parameters more accurate. After a point is solved, its error could be defined as the sum of the deviations from each of its constraints. The point could be solved once for each individual constraint, and the error computed by taking the sum of the distances from the aggregate solved point to each of these individual solutions. Since the image location of each point is assumed to be the least reliable constraint, some amount of iteration could be performed by moving the image location of the point and examining the effects on the error metric. Points could be shifted in each iteration until the total error for each point is below a given threshold.
Appendix A
Vanishing Point Concept

A.1 Origin of Vanishing Points

Vanishing points, as introduced in Section 2.3.4.4, are the apparent convergences of parallel lines as projected onto the image plane. If a line that is not parallel to the image plane is extended out to infinity, the projection of the infinite endpoint represents the vanishing point for the direction parallel to that line.

We can represent a line parametrically as

\[ v = v_0 + v_d \cdot t \]  \hspace{1cm} (A-1)

where \( v_0 \) is a point on the line, \( v_d \) is a vector in the direction of the line, and \( t \) is the varying parameter. Expanding these vectors gives

\[ v = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^T \]

\[ v_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}^T \]  \hspace{1cm} (A-2)

\[ v_d = \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}^T . \]

Projecting the point \( v \) onto the image plane with a focal distance \( d \) yields point \( P \), whose components are
\[ P_x = d \left( \frac{x_0 + x_d t}{z_0 + y_d t} \right) \]
\[ P_y = d \left( \frac{y_0 + y_d t}{z_0 + z_d t} \right) \]
\[ P_z = d. \]  \hspace{1cm} (A-3)

Now, we let the parameter \( t \) approach infinity:

\[
\lim_{t \to \infty} P_x = d \frac{x_d}{z_d} \\
\lim_{t \to \infty} P_y = d \frac{y_d}{z_d} \\
\lim_{t \to \infty} P_z = d. \]  \hspace{1cm} (A-4)

The projection clearly approaches a finite vanishing point as the line is extended to infinity, unless the component \( z_d \) is zero. In this case, the line is parallel to the image plane and its projection does not approach a finite vanishing point. Note that the point \( P \) depends only on the direction of the 3-D line \( (v_d) \), and not its position \( (v_0) \); this implies that vanishing point locations are dependent only on the camera’s orientation and not its extrinsic location.

### A.2 Focal Point Computation

Every infinite 3-D line parallel to a given principal axis has the same vanishing point and, conversely, every 3-D line with this vanishing point is parallel to the corresponding principal axis. Consider, then, the special line which passes through both the camera’s focal point and a principal vanishing point. By the projection equations, all points on this line must project to the same point on the image, namely its vanishing point. The line must therefore be parallel to the corresponding principal axis.

If three such special lines are generated, one through each of the principal vanishing points, a coordinate frame is formed whose origin is the focal point and whose axes are defined as the vectors from the focal point to the vanishing points (Figure A-1). The frame is rotationally equivalent to the scene space frame, since the vectors from the focal point to each vanishing point are parallel to each of the three principal scene space axes. These
vectors thus give the camera’s orientation with respect to the scene space, as discussed in Section 3.2.2.1.

![Coordinate Frame of Focal Point and Vanishing Points](image)

**Figure A-1: Coordinate Frame of Focal Point and Vanishing Points**

Lines are drawn from the focal point $F$ through each of the principal vanishing points $A$, $B$, and $C$. These lines form a coordinate frame whose axes are parallel to the scene space axes. Camera rotation and the image-relative focal point location can thus be determined solely from the positions of the vanishing points.

Because of the above constraints, the focal point's position can be computed directly from the three principal vanishing points. Recall from Section 3.2.1.2 that

$$FA \cdot FB = 0$$
$$FB \cdot FC = 0$$
$$FC \cdot FA = 0.$$  \hspace{1cm} (A-5)

In expanded form, these equations become

$$(A_x - F_x)(B_x - F_x) + (A_y - F_y)(B_y - F_y) + (A_z - F_z)(B_z - F_z) = 0$$
$$(B_x - F_x)(C_x - F_x) + (B_y - F_y)(C_y - F_y) + (B_z - F_z)(C_z - F_z) = 0$$  \hspace{1cm} (A-6)
$$(A_x - F_x)(C_x - F_x) + (A_x - F_y)(C_y - F_y) + (A_z - F_z)(C_z - F_z) = 0.$$

Since points $A$, $B$, and $C$ lie in the image plane, their $z$ coordinates are zero. Making this substitution and expanding further yields

$$F_x^2 + F_y^2 + F_z^2 - F_x(A_x + B_x) - F_y(A_y + B_y) = 0$$  \hspace{1cm} (A-7)
$$F_x^2 + F_y^2 + F_z^2 - F_x(B_x + C_x) - F_y(B_y + C_y) = 0$$  \hspace{1cm} (A-8)
\[ F_x^2 + F_y^2 + F_z^2 - F_x(C_x + A_x) - F_y(C_y + A_y) = 0. \] \hspace{1cm} (A-9)

The next step is to solve for \( F_x \) and \( F_y \). Two linear equations in these unknowns can be generated by subtracting Equation A-8 from Equation A-7 and Equation A-9 from Equation A-8:

\[ F_x(C_x - A_x) + F_y(C_y - A_y) = 0 \]
\[ F_x(A_x - B_x) + F_y(A_y - B_y) = 0. \] \hspace{1cm} (A-10)

An analytic solution of the two unknowns is then straightforward.

Finally, \( F_z \) is calculated by substituting the now-known parameters \( F_x \) and \( F_y \) into Equation A-7. Since this equation is quadratic in \( F_z \), both a positive and a negative solution are possible. This does not cause a problem, however, because \( F_z \) is taken in this context to be the negative of the focal length, which itself is always positive.
Appendix B

Mathematical Formulations

B.1 Solving for Camera Translation

Once camera intrinsics and rotation are known, the three translational components $T_x$, $T_y$, and $T_z$ can be solved analytically. The view transformation $V$ is given by

$$ V = SH \cdot T(F_2) \cdot R \cdot T(-VRP), $$

(B-1)

where $SH$ is a shear matrix, $T(F_2)$ represents a translation by the negative of the focal length, $R$ is a rotation into the VRC axes, and $T(-VRP)$ is a translation of the VRP to the origin. These matrices are described in Section 2.3.5.1.

Since all parameters are known except for $T(-VRP)$, the transformation can be written more simply as

$$ V = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. $$

(B-2)

Here, $T_x$, $T_y$, and $T_z$ are the negatives of the coordinates of the VRP, and the $A_{ij}$ represent the entries of the composition of matrices $SH$, $T(F_2)$, and $R$.

Once a point is transformed by the matrix $V$, it is in the camera coordinate frame. From this frame, the projection equations (Equation 2-13) can be applied to the point to obtain its $(x, y)$ position on the view plane.
The two known feature points \( P_1 \) and \( P_2 \), when transformed to this new space, become \( C_1 \) and \( C_2 \). The coordinates of \( C_1 \) are given by
\[
\begin{align*}
C_{1x} &= A_{11}(P_{1x} + T_x) + A_{12}(P_{1y} + T_y) + A_{13}(P_{1z} + T_z) + A_{14} \\
C_{1y} &= A_{21}(P_{1x} + T_x) + A_{22}(P_{1y} + T_y) + A_{23}(P_{1z} + T_z) + A_{24} \\
C_{1z} &= A_{31}(P_{1x} + T_x) + A_{32}(P_{1y} + T_y) + A_{33}(P_{1z} + T_z) + A_{34}.
\end{align*}
\] (B-3)

Those of \( C_2 \) are obtained similarly.

The information actually available, however, is not the scene point in camera space, but rather its projection onto the image, \( S_1 \). From the projection equations, we have
\[
\begin{align*}
S_{1x} &= d \cdot \frac{C_{1x}}{C_{1z}} \\
S_{1y} &= d \cdot \frac{C_{1y}}{C_{1z}}.
\end{align*}
\] (B-4)

Substituting for \( C_{1x}, C_{1y}, \) and \( C_{1z} \),
\[
\begin{align*}
S_{1x} &= d \cdot \frac{A_{11}(P_{1x} + T_x) + A_{12}(P_{1y} + T_y) + A_{13}(P_{1z} + T_z) + A_{14}}{A_{31}(P_{1x} + T_x) + A_{32}(P_{1y} + T_y) + A_{33}(P_{1z} + T_z) + A_{34}} \\
S_{1y} &= d \cdot \frac{A_{21}(P_{1x} + T_x) + A_{22}(P_{1y} + T_y) + A_{23}(P_{1z} + T_z) + A_{24}}{A_{31}(P_{1x} + T_x) + A_{32}(P_{1y} + T_y) + A_{33}(P_{1z} + T_z) + A_{34}}.
\end{align*}
\] (B-5)

Deriving similar equations for \( S_{2x} \) and \( S_{2y} \), and cross-multiplying yields four linear equations in the three unknowns \( T_x, T_y, \) and \( T_z \), which can be solved directly by a least-squares method since the parameters \( d, A_{ij}, P_1, P_2, S_1, \) and \( S_2 \) are all known.

### B.2 Projection Equations

If the image coordinates of an unknown point \( P_1 \) are given and the image's camera has been completely solved, then two linear equations can be generated in the point's three unknowns \( P_{1x}, P_{1y}, \) and \( P_{1z} \).

The transformation \( V \) which takes points in scene space to camera space is given in Equation B-1. Applying \( V \) to an unknown point \( P_1 \) yields
\[ C_{1x} = V_{11}P_{1x} + V_{12}P_{1y} + V_{13}P_{1z} + V_{14} \]
\[ C_{1y} = V_{21}P_{1x} + V_{22}P_{1y} + V_{23}P_{1z} + V_{24} \]
\[ C_{1z} = V_{31}P_{1x} + V_{32}P_{1y} + V_{33}P_{1z} + V_{34}. \]  

(B-6)

Here, the point \( C_1 \) is the unknown point in camera space, and \( V_{ij} \) represents the entry in the \( i \)th row and \( j \)th column of the matrix \( V \).

Now, since the image coordinates of \( P_1 \) are known, the two projection equations become

\[ S_{1x} = d \cdot \frac{V_{11}P_{1x} + V_{12}P_{1y} + V_{13}P_{1z} + V_{14}}{V_{31}P_{1x} + V_{32}P_{1y} + V_{33}P_{1z} + V_{34}} \]
\[ S_{1y} = d \cdot \frac{V_{21}P_{1x} + V_{22}P_{1y} + V_{23}P_{1z} + V_{24}}{V_{31}P_{1x} + V_{32}P_{1y} + V_{33}P_{1z} + V_{34}}. \]  

(B-7)

Cross-multiplying these last two equations and gathering terms results in two linear equations in the three unknowns \( P_{1x}, P_{1y}, \) and \( P_{1z} \).

### B.3 Frame Rectification

If we define matrix \( A \) as taking points in frame \( F_1 \) to points in frame \( F_2 \), and define \( P_1 \) as a point in \( F_1 \) and \( P_2 \) as the same point in \( F_2 \), then we have the simple equation

\[ P_2 = A \cdot P_1. \]  

(B-8)

Matrix \( A \) is a composition of affine transformations and is shown in expanded form in Equation 4-1. There are thus twelve unknown parameters to be solved. Expanding Equation B-8 for the solved point pair \( P_1 \) and \( P_2 \) yields three equations in these twelve unknowns:

\[ P_{2x} = A_{11}P_{1x} + A_{12}P_{1y} + A_{13}P_{1z} + A_{14} \]
\[ P_{2y} = A_{21}P_{1x} + A_{22}P_{1y} + A_{23}P_{1z} + A_{24} \]
\[ P_{2z} = A_{31}P_{1x} + A_{32}P_{1y} + A_{33}P_{1z} + A_{34}. \]  

(B-9)

Every additional point pair gives another set of three equations. All entries of \( A \) can therefore be solved if there are at least four such pairs that are not all coplanar, since they provide exactly twelve independent constraints.
Appendix C
Data Structures

C.1 Viewports

Camera and view information is combined in the ViewPort data structure. Figure C-1 shows the file format of a ViewPort.

```
# A ViewPort example
Camera SampleCamera
ProjectionType PERSPECTIVE
VRP 175 175 175
VPN -1 -1 -1
VUP 0 1 0
PrincipalPoint 0 10 -200
WindowSize 320 240
ClipDepths 0 10000
ScreenSize 320 240 256
EndCamera
```

Figure C-1: Sample ViewPort File
The ViewPort data structure as stored in a text file. Comments are added after the '# ' symbol, and the parameters can be specified in any order.

The first two lines give the name of the viewport and its projection type (ORTHOGRAPHIC or PERSPECTIVE). The next three lines specify the VRC frame by supplying the scene-relative view reference point, view plane normal, and view up vectors. The two lines that follow give the position of the focal point in VRC space and the size of the virtual window on the image plane. Front and back clip distances, respectively, are specified by ClipDepths, and ScreenSize gives width, height, and depth scalings for the final
display window. Finally, the **EndCamera** keyword signifies the end of the camera data, so that a ViewPort structure can be embedded within a larger file. An example of such an embedding is shown in the next section.

## C.2 Image Data Elements

Point, edge, and surface information is stored in a large **ImageData** structure. The order in which the three main element types are stored is crucial, since edges have references to points, and surfaces have references to both points and edges. Figure C-2 shows the overall structure of an ImageData file.

```
# An ImageData example
ImageData SampleImageData
ImageSize 320 240
ImageFile /data/Image1.dat
Sense X -
Sense Y -
Sense Z +

# Camera structure can be inserted here

# Points go here
# Edges follow
# Surfaces are last

EndImageData
```

**Figure C-2: Sample ImageData File**

A skeleton ImageData structure, stored as a text file. The commented lines indicate where sub-structures, contained in the ImageData, are inserted.

Each ImageData set can have a name, and also specifies the dimensions and file location of the original image on which it builds. Positive and negative senses of each edge direction are also specified. The image's camera parameters, if the camera has been solved, is stored according to the specification presented in the previous section.

Points are each tagged with an identification number and contain the \((x, y)\) coordinates of their original image positions. Solved points also have a scene-relative position. Anchor points are marked as such, and features have an associated name.
Point 3
   Image 0 11
   Actual 150 150 150
   Anchor
      FeatureName UpperLeftCorner
Point 4
   Image -1 93
   Actual -158.961 -158.778 145.753
Point 5

Figure C-3: Point Structures
Several example points are shown within an ImageData text file. Each point has an ID number, so that other image elements can refer to it, and an image position. The 3-D coordinates of the point are listed if the point has been solved, as are tags for anchors and features.

Every edge contains references to its two endpoints. These references are listed simply as the identification numbers of the points, so all points must be listed before any edges. A given edge also has an identification number and a direction, which can be X, Y, Z, or ?. The last denotes the null direction.

Edge 5
   Points 2 1
   Direction Z
Edge 6
   Points 0 5
   Direction ?
Edge 7

Figure C-4: Edge Structures
Edges as they appear in the ImageData text file. Each edge has an ID number for references from other objects, and contains the ID numbers of its two endpoints. User-specified scene space directions are also listed.

Finally, each surface is named rather than tagged with a number, and has a list of references to the points and edges that belong to it. An example is shown in Figure C-5.
Figure C-5: Surface Structure
In the ImageData file, a surface contains only a unique name and a list of the points and edges associated with it. These elements are referred to by their ID numbers.

In a given Builder project file, data from many images is merged into a single scene database. Each set of image data is stored in the ImageData structures described above, and multiple sets are stored sequentially in the file. The EndImageData tag allows these structures to be embedded cleanly within larger files.
Glossary

**affine transformation**: 3-D coordinate transformation which preserves parallelism of lines. Translation, rotation, scale, and shear are all affine transformations.

**camera space**: Coordinate space in which the focal point is in the x-y plane and the vector from the focal point to the center of the view window is parallel to the z axis.

**clip planes**: The front and back bounds of the view volume.

**coordinate frame**: 3-D Cartesian space.

**coordinate transformation**: 4-by-4 matrix that takes points from one frame to another.

**depth buffer**: Same as z-buffer.

**drag vector**: Vector formed between initial and final mouse positions.

**dynamic texture rendering**: Texture rendering process that does not use texture maps.

**extrinsic camera parameters**: Six values denoting camera’s position and orientation in scene space.

**focal distance**: Same as focal length.

**focal length**: Perpendicular distance from the focal point to the image plane.

**focal point**: Point at which all light rays converge in the pinhole camera model.

**foreshortening**: Effect which makes distant objects appear smaller than close objects.

**GUI**: Graphical User Interface.

**image plane**: Plane onto which a 3-D scene is projected.

**intrinsic camera parameters**: 3-D coordinates of the focal point’s position in VRC space.

**MMSE**: Minimum Mean-Square Error.

**normal**: Perpendicular.

**orthographic projection**: Projection in which parallel lines remain parallel.

**parallel projection**: Same as orthographic projection.

**perspective projection**: Projection in which light rays converge at a single focal point.

**pinhole camera**: Ideal camera model in which all light rays are focused through a focal
point and project onto an image plane.

**principal vanishing point:** Vanishing point formed by lines parallel to a principal axis.

**rotation:** Affine transformation that rotates points about the origin.

**scale:** Affine transformation that stretches along the coordinate axes.

**scan conversion:** Process by which polygons are filled to a pixel-based buffer.

**scene:** Collection of 3-D objects in their own coordinate frame.

**scene space:** Coordinate frame of objects in a scene.

**shear:** Affine transformation that shifts points as a function of their distance along the shear axis.

**texture map:** 2-D buffer that contains a texture image to be mapped onto a 3-D object.

**translation:** Affine transformation that performs a rigid shift.

**up vector:** Vector which, when projected on the image plane, forms the y axis of VRC space.

**vanishing point:** Point to which parallel lines appear to converge when projected onto the image plane.

**view plane:** Same as image plane.

**view plane normal:** Vector normal to the view plane specifying the z axis in VRC space.

**view reference coordinates:** Coordinate frame whose origin is the center of the view window and whose z axis is normal to the view plane.

**view reference point:** Origin in VRC space.

**view volume:** Volume of space visible to the camera.

**VPN:** Same as view plane normal.

**VRC:** Same as view reference coordinates.

**VRP:** Same as view reference point.

**VUP:** Same as up vector.

**world space:** Same as scene space.

**z-buffer:** Buffer that contains the depths of all visible pixels.
References


