Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping

TR99-022
April 21, 1999

William R. Mark

Graphics and Image Processing Laboratory
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175

POST-RENDERING 3D IMAGE WARPING: VISIBILITY, RECONSTRUCTION, AND PERFORMANCE FOR DEPTH-IMAGE WARPING

by

William R. Mark

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill

1999

Approved by:

Gary Bishop

Advisor: Gary Bishop

Frederick P. Brooks

Steven E. Molnar

Reader: Steven Molnar

Henry Fuchs

Reader: Anselmo Lastra

Turner Whitted

ABSTRACT

William R. Mark

POST-RENDERING 3D IMAGE WARPING:
VISIBILITY, RECONSTRUCTION, AND PERFORMANCE FOR
DEPTH-IMAGE WARPING

(Under the direction of Dr. Gary Bishop)

The images generated by real-time 3D graphics systems exhibit enormous frame-to-frame coherence, which is not exploited by the conventional graphics pipeline. I exploit this coherence by decoupling image rendering from image display. My system renders every Nth frame in the conventional manner, and generates the in-between frames with an image warper. The image warper modifies a rendered image so that it is approximately correct for a new viewpoint and view direction.

My image warper uses McMillan's 3D image warp. Unlike perspective image warps, the 3D image warp can correct for changes in viewpoint, even for objects at different depths. As a result, my system does not require the application programmer to segment the scene into different depth layers, as is required by systems that use a perspective image warp.

I attack three major challenges associated with using 3D warping for rendering acceleration: visibility, reconstruction, and performance. I describe how to choose pairs of rendered images so that most of the needed portions of the scene are visible in these images. I describe a framework for the 3D warp reconstruction problem, and develop reconstruction algorithms that produce good quality images. Finally, I describe properties of the 3D warp that could be used to build efficient 3D image warpers in hardware.

My technique can also compensate for rendering-system latency and network latency. I have built a real-time system that demonstrates this capability by displaying rendered images at a remote location, with low latency.

PREFACE

Thanks to

My parents, for their love and good judgment while raising me.

Pam, for much happiness and for her support during my dissertation work.

Gary Bishop, for starting me along this research path, asking questions, directing my research, and being patient even when I was frustrated.

Leonard McMillan for serving as a sort of second advisor early in my dissertation work. The numerous whiteboard discussions I had with him were one of the best parts of my graduate school experience.

DARPA, NSF, Microsoft, the Link Foundation, Intel, and the UNC Computer Science Alumni Fellowship, for funding my research.

My committee members, for agreeing to help guide my research.

Anselmo Lastra and Steve Molnar, for serving as dissertation readers.

Fred Brooks, for providing an inspiring example of how to conduct research, teach, and lead. Also, for asking insightful questions about my work, and for encouraging and supporting me during my entire graduate education.

Steve Molnar, for his encouragement, and for several discussions that helped to clarify my thinking about key concepts in my dissertation.

Dan Aliaga, for putting up with me as an officemate for over five years.

UNC's graphics research groups, for providing an open, exciting and cooperative research environment.

The many fellow grad students with whom I spent so much time, especially Dan Aliaga, Alexandra Bokinsky, Jon Cohen, Adam Duggan, Stefan Gottschalk, Aron Helser, David Luebke, Carl Mueller, Michael North, Marc Olano, and Mark Parris.

The staff of the UNC computer science department, for helping me in innumerable ways and for being a pleasure to work with.

The many teachers who helped bring me to this point, especially my high school science teachers.

The UNC Walkthrough group, for providing the kitchen model.

Electric Boat, for providing the submarine machine-room model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

FOV — field of view

HMD — head-mounted display

LDI — layered depth image

PRW — post-rendering warp

CHAPTER 1

INTRODUCTION

1.1 The problem and approach

Real-time computer-generated images contain an enormous amount of frame-to-frame coherence. Almost all surfaces that are visible in any particular frame will be visible in the next several frames. However, the conventional graphics pipeline does not attempt to take advantage of this frame-to-frame coherence. Each frame is rendered independently of previous and future frames.

I will demonstrate that it is possible to take advantage of frame-to-frame coherence by modifying the conventional graphics pipeline. The graphics system can then render a series of frames more efficiently than by computing these same frames independently. This increased efficiency allows us to render more complex 3D scenes, or to construct cheaper graphics systems.

The research presented in this dissertation explores one strategy for modifying the graphics pipeline to exploit frame-to-frame coherence. This strategy decouples the conventional rendering stages of the graphics pipeline from the display, by allowing changes in viewpoint to be processed after rendering is complete. Figure 1.1 contrasts this new pipeline with the conventional graphics pipeline. The new pipeline has a new image-warping stage between the conventional rendering stage and the display. This image warping stage modifies a rendered image so that it is approximately correct for a new viewpoint and new view direction, without re-rendering the image. I refer to this technique as *post-rendering image warping* (PRW).

The input to the new image-warping stage is a sequence of conventionally rendered images generated at a low rate. The output from this warping stage is a series of displayable frames generated at a high rate. In effect, the image-warping stage is interpolating between sparse conventionally rendered images to produce the displayed frames. The interpolation is possible because of the high degree of frame-to-frame coherence.

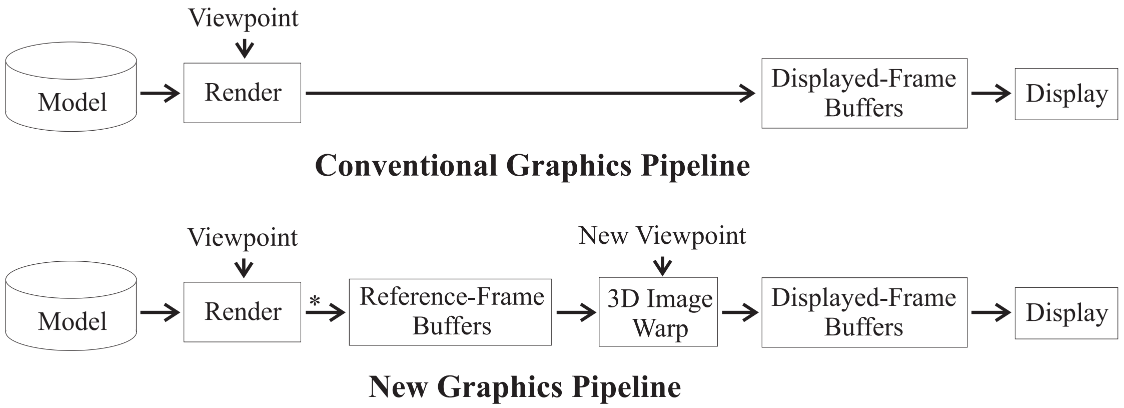


Figure 1.1: *Conventional graphics pipeline vs. new proposed pipeline. If desired, the new pipeline can be split into a client half and a server half at the point indicated by * to create a system for low-latency display of rendered images at a remote location. I refer to the images produced by the conventional renderer as* reference frames, *and the the images that are displayed as* displayed frames.

In addition to decoupling the rendering frame rate from the display frame rate, PRW can hide latency in the rendering pipeline. The effective latency (to viewpoint change) of the image generation system becomes that of the image warping stage, instead of that of the entire rendering pipeline. This latency reduction is achieved because the image-warping stage generates frames for an up-to-date viewpoint.

Latency reduction is especially important in a system that renders images at one location, then transmits them over a network for display in a second (*remote*) location. When a user at the remote location controls the viewpoint, an image warper at the remote location can compensate for the network latency.

The amount of computation performed by the image warping stage in the new graphics pipeline is approximately proportional to the number of pixels in the displayed frame. In contrast, the amount of computation performed by the conventional rendering pipeline is an increasing function of scene complexity. For sufficiently complex models, the use of the additional image warping stage is therefore more computationally efficient than using the conventional rendering pipeline alone.

Other researchers (e.g. [Regan94]) have taken this same general approach to real-time rendering acceleration and latency reduction. My work is distinguished from this earlier work by the use of an image warp that relies on per-pixel depth values (a 3D warp). I was inspired to start down this research path by Gary Bishop's suggestion that McMillan's 3D warp could be used to reduce latency in a client/server rendering system.

1.2 Thesis statement, results, and outline

The central thesis of this research is:

> By adding image-based rendering capability to the conventional graphics pipeline, we can decouple the conventional rendering pipeline from the display. This decoupling can be used to more cost-effectively display complex models at interactive frame rates, with only minor degradation in image quality.

The new image-warping stage of the pipeline uses a 3D image warp of the type developed by McMillan and Bishop [McMillan95a, McMillan97]. McMillan and Bishop's 3D warp uses images with per-pixel depth (*depth images*), so that every pixel is warped in a geometrically correct manner. A large part of this dissertation is concerned with enhancements to the basic 3D warp that increase the quality of warped images. In particular, I develop visibility, reconstruction and compositing techniques to produce a single displayed frame from the 3D warps of multiple reference frames. These results are applicable to a wide variety of image-based rendering problems besides rendering acceleration, and are important contributions in their own right.

To experiment with different algorithms for visibility and reconstruction, I built an off-line post-rendering warping test-bed. This test-bed simulates the operation of a real-time post-rendering warping system at the algorithm level. It also correctly simulates the timing of rendered and displayed frames. I drove this test-bed with actual user motion data that was previously captured from a user wearing a tracking device. I used this test-bed to generate most of the frame snapshots in this dissertation.

I also built a real-time post-rendering warping system. This system, described in Chapter 6, is split into a client half and a server half to allow low-latency display of rendered imagery at a remote location. Despite extensive optimization efforts, the software warper in this system does not achieve the performance levels required for a local post-rendering warping system. I concluded that hardware support for 3D warping would be required to obtain an acceptable price/performance ratio from a post-rendering warping system in the near future. For this reason, I investigated some of the issues involved in hardware acceleration of the 3D warp. In particular, I attempted to determine how special properties of the 3D warp that are not shared by general polygon rendering could be exploited to design a cost-effective 3D warper.

Table 1.1 provides a summary of the major issues addressed by this research. In the following sections, I discuss some of these issues further, so that the reader will understand the key elements of my post-rendering warping strategy.

Issue	Solution / Approach Followed
How many reference frames are typically needed to produce each displayed frame in order to reduce the number of visibility artifacts to an acceptably low threshold?	When reference frames are generated at 5 Hz and high-quality motion prediction is available, two reference frames are sufficient to eliminate most visibility artifacts.
How should we choose reference-frame viewpoints?	One reference frame is located at or near a previous position of the user, and one reference frame is located at or near a future position of the user.
How can we minimize the perceptual impact of any visibility artifacts that remain?	My system copies the local background color into any empty areas of the displayed frame, blurring as it does so. By making use of the epipolar geometry of the 3D warp, the system performs this blurring using only a single, constant-work pass (per reference frame) over the displayed frame.
How frequently should reference frames be generated?	If the reference frame rate is too low, the field of view of the reference frames must be huge to allow for changes in view direction (head rotation). Thus, the maximum expected rate of head rotation dictates a minimum conventional-rendering rate. Below this rate (approximately 3-8 Hz), the benefits of post-rendering warping are greatly reduced or eliminated by the need to render excessively oversized reference frames. This restriction does not apply to a system in which the only goal of post-rendering warping is to reduce latency (as opposed to a system in which the goal is also to efficiently increase the frame rate).
How do we combine the information from multiple reference frames to produce a single displayed frame of the highest possible quality?	Chapter 4 describes a conceptual framework for 3D warping reconstruction and resampling. An ideal algorithm requires access to all reference frames simultaneously.
How can we *efficiently* combine the information from multiple reference frames?	I describe a simple heuristic for determining whether or not two adjacent reference-frame pixels belong to the same surface in 3D space. Making this determination is sufficient to allow the reconstruction algorithm to work with one reference frame at a time.
How can anti-aliasing be efficiently incorporated into post-rendering warping?	I develop a REYES-inspired approach to super-sampling, which uses inexpensive flat-shaded, axis-aligned rectangles for reconstruction. I also use an A-buffer-like format to efficiently store super-samples.
How can the above techniques be efficiently implemented in hardware so as to cost-effectively use post-rendering warping as a rendering acceleration technique?	I describe a number of properties of the 3D warp that can be used to design cost-effective warping hardware. In particular, I show that the 3D warp can be implemented using fixed-point arithmetic, and that the memory-access patterns of the warp allow the design of a hardware warper with a small cache.

Table 1.1: *Summary of issues and approaches followed in this dissertation*

1.2.1 Visibility

An image is rendered using a particular center of projection, or viewpoint, in 3D space (assuming a perspective projection). When each pixel is augmented with a depth value in addition to the usual color values, this image describes a subset of the 3D geometry in the 3D world. More specifically, the image describes all objects in the 3D world that are visible from the chosen viewpoint and within the field-of-view of the image. The resolution of the image determines the precision of this sampled representation.

In post-rendering warping, our goal is to use one or more such rendered images as a partial description of the 3D world. We can then generate new images for nearby viewpoints from this partial description. It is in this manner that we interpolate between rendered frames to produce displayed frames.

If we are to interpolate between a set of rendered frames to generate images for new viewpoints, we must have some kind of assurance that our reference frames adequately describe the geometry which is visible from the new viewpoint. Unfortunately, we can not guarantee that one image tells us *anything* about geometry that is visible from viewpoints other than the one at which it was rendered.[1] However, we expect that a set of reference frames rendered at a variety of suitable viewpoints will better describe the 3D world.

This dissertation shows that if we make some assumptions about the geometry of the world and carefully choose our reference-frame viewpoints, then most geometry needed for new viewpoints will be represented in the reference frames. In particular, I demonstrate that for a post-rendering warping system, two properly chosen reference frames contain almost all of the necessary geometry in most cases. Figure 1.2 summarizes how my system chooses its reference-frame viewpoints.

Usually, some small amount of geometry will not be visible in any of the reference frames. The warping algorithm must choose some color to place in the areas of the displayed frame corresponding to this missing geometry. I characterize this problem as one of minimizing perceived error. I develop an efficient algorithm for filling these *visibility holes* in the displayed frame. This algorithm relies on the epipolar geometry of the reference and displayed frames.

Chapter 3 discusses the visibility problem in detail. The choice of reference-frame viewpoints is a crucial part of the visibility problem. Thus, Chapter 3 also discusses the use of head-motion-prediction algorithms to choose reference-frame viewpoints.

[1] As a pathological example, imagine a viewer surrounded by inward-pointing pipes.

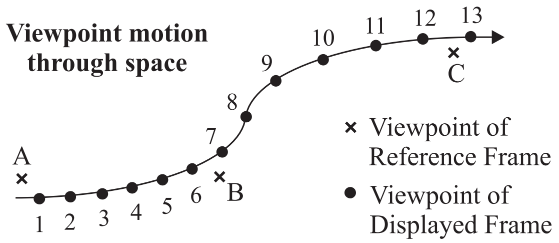


Figure 1.2: *Locations of reference frames and displayed frames. Displayed frames are computed by warping two reference frames, one near a past position of the viewer, and one near a future position of the viewer. For example, displayed frame #5 is produced by warping reference frames A and B. Ideally the reference frames lie exactly on the viewpoint path. But if future viewpoint locations are unknown, then motion prediction must be used to estimate them. As a result, the reference frames do not fall exactly on the viewpoint path.*

1.2.2 Reconstruction from multiple reference frames

Each reference frame contains a 2D array of samples of the 3D world. Although these samples are regularly spaced in the 2D reference frame, in general they are irregularly spaced in the 3D world. This irregular spacing results from the variation in per-pixel depths. Because displayed-frame viewpoints are in general different from reference-frame viewpoints, the samples will also be irregularly spaced when projected into 2D displayed-frame space.

We usually warp more than one reference frame to compute each displayed frame. Each reference frame contributes a set of irregularly-spaced samples in the 3D world. The problem is then to construct the displayed frame from these 3D samples. The irregular sample spacing makes this reconstruction and resampling problem different from most of the reconstruction problems encountered in computer graphics and image processing.

My research characterizes the 3D warp reconstruction and resampling problem as one consisting of the following steps:

1. Reconstruct 2D manifolds (surfaces) in 3-space from the irregularly-spaced 3D reference-frame samples.

2. Project the manifolds into the 2D displayed-frame space.

3. Composite the projected manifolds to produce the displayed frame.

My research uses this characterization to guide the design of practical reconstruction and resampling algorithms. Chapter 4 discusses the reconstruction and resampling problem, and describes my algorithms.

1.2.3 System summary

Figure 1.3 is a conceptual diagram of a post-rendering 3D warping system. Although the figure shows two image warpers, an actual system might share one warper to warp both reference frames. I assume such a one-warper configuration in most of this dissertation. Because the system must render reference frames at or near future viewpoints, the reference-frame viewpoints are generated by a motion predictor. Some prediction error is tolerable, since the 3D warp compensates for both position and orientation changes.

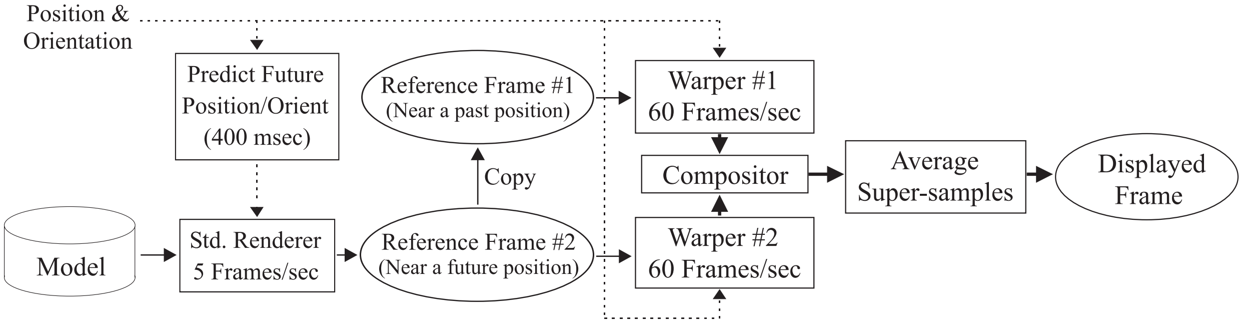


Figure 1.3: *Conceptual diagram of a post-rendering 3D warping system.*

1.2.4 Outline

The rest of the dissertation is organized in the following manner:

- The remainder of Chapter 1 describes the notation that I use for planar projections and for the 3D warp, and reviews the epipolar geometry of the 3D warp.

- Chapter 2 discusses previous work. The first part of the chapter covers general sample-based rendering work, and the second part discusses previous applications of sample-based rendering to acceleration of conventional rendering.

- Chapter 3 discusses the visibility problem for post-rendering warping. It describes and analyzes my algorithm for choosing reference-frame viewpoints. Chapter 3 also describes my algorithm for filling visibility holes.

- Chapter 4 discusses the reconstruction and resampling problem. It provides a conceptual framework for the problem, then describes my reconstruction and resampling algorithms.

Chapter 4 also describes the strengths and weaknesses of a variety of alternative reconstruction algorithms developed by others and by me.

- Chapter 5 discusses issues that are relevant to designing a hardware 3D warper. The chapter emphasizes opportunities for efficient implementation that are unique to the 3D warp (not shared by conventional polygon renderers).

- Chapter 6 describes the real-time warping system that I initially built, and that Tom Hudson (another graduate student) enhanced. The system provides low-latency display of rendered imagery in a remote location.

- Chapter 7 concludes the dissertation. It describes the characteristics of applications that are best suited to acceleration by post-rendering warping. I also provide my opinion of the future prospects for post-rendering 3D warping.

1.3 Notation

This section describes some notation and coordinate conventions which are used throughout this dissertation. In particular, it describes the conventions for image coordinates, planar projection descriptions, and 3D warp equations. It also describes the epipolar geometry of the 3D warp, and its connection to McMillan's occlusion-compatible warp. In this section, I refer to the input image for the 3D warp as the *reference image*, and the output image as the *destination image*. Throughout the dissertation, I use these terms when I am referring specifically to the input and output of a 3D warp. I use the similar terms *reference frame* and *displayed frame* when I am referring to images in the context of a post-rendering warping system. There is not always a one-to-one correspondence between a destination image and a displayed frame, since a PRW system may use two or more destination images (from the 3D warps of two or more reference images) to create a single displayed frame.

1.3.1 Planar projections

I adopt a modified version of the notation used in [McMillan97] to describe planar projections. This modified notation is illustrated in Figure 1.4. The vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$ form the basis vectors for the camera coordinate system. In general, this coordinate system is non-orthogonal, since it is representing a perspective projection. For an arbitrary point (r, s, t) in this camera coordinate system, we can calculate the corresponding world-space coordinates $\vec{x}$ of the point:

$$\vec{x} = r\vec{a} + s\vec{b} + t\vec{c} + \dot{C}. \tag{1.1}$$

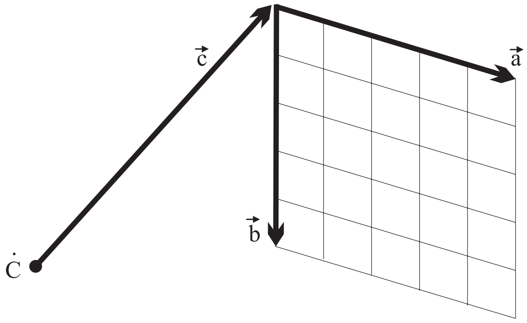


Figure 1.4: *Camera model for planar projections.*

I use an image-coordinate convention in which image coordinates are specified as (u, v), with $u, v \in [0, 1]$. Thus, we compute image coordinates from camera coordinates as follows:

$$u = \frac{r}{t} \qquad v = \frac{s}{t}. \tag{1.2}$$

Occasionally it is useful to specify the image-space coordinates in terms of pixels, rather than values between 0 and 1. I will refer to this alternate representation as *resolution-dependent* image coordinates. With this alternate convention, $u \in [0, width]$ and $v \in [0, height]$. This representation is especially useful for implementations of the 3D warp that are optimized for speed, and is used in [McMillan97]. There is a corresponding resolution-dependent planar projection model, illustrated in Figure 1.5. When both the resolution-dependent image coordinates and resolution-dependent camera model are used, equations 1.2 still hold.

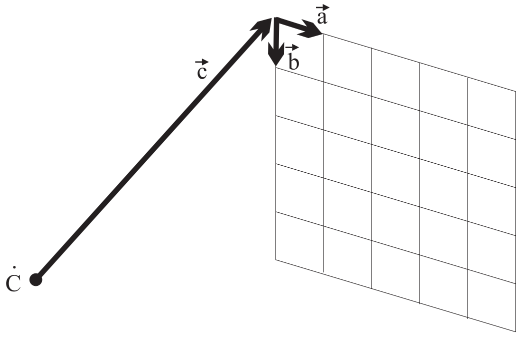


Figure 1.5: *Alternate, resolution-dependent, camera model for planar projections.*

With either variation of the camera model, the vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$ can be combined to form a single matrix, $\mathbf{P}$. This matrix (adopted from [McMillan97]) describes the intrinsic camera parameters and the camera orientation:

$$\mathbf{P} \equiv \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix}, \tag{1.3}$$

1.3.2 3D warp

We can use the notation just presented to describe the transform used in a 3D warp. For known camera parameters, the 3D warp is just an "un-projection", rotation, translation, and re-projection of a point. My approach to the 3D warp is somewhat different in emphasis from McMillan's. McMillan emphasized a non-Euclidean formulation of the 3D warp, which is useful for warping images acquired with unknown or poorly-known camera calibration. I use a strictly Euclidean formulation, because post-rendering warping always uses known "camera" parameters. I use the following 3D warp equation:

$$\frac{z_2}{S_2}\bar{u}_2 = \mathbf{P}_2^{-1}\mathbf{P}_1\frac{z_1}{S_1}\bar{u}_1 + \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2), \tag{1.4}$$

$$\text{where} \qquad S \equiv \frac{\vec{a} \times \vec{b}}{\left\|\vec{a} \times \vec{b}\right\|} \cdot \vec{c} \tag{1.5}$$

with

$$\bar{u}_1 = \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}, \qquad \bar{u}_2 = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}. \tag{1.6}$$

In this equation, (u_1, v_1) are the reference-image coordinates and (u_2, v_2) are the displayed-image coordinates. $\mathbf{P}$ and $\dot{C}$ represent the pinhole camera viewing parameters and center of projection respectively for the images, as described earlier. S is a scale factor that represents the distances of the image planes, as described by $\mathbf{P}_1$ and $\mathbf{P}_2$, from their respective centers of projection. z_1 and z_2 are the reference-image and displayed-image depth values. These depth values are defined using the usual "computer-graphics" definition of Z:

$$z = \vec{p} \cdot \hat{\vec{n}}, \tag{1.7}$$

where $\vec{p}$ is the vector from the center-of-projection to the 3D point and $\hat{\vec{n}}$ is the unit-length normal vector for the projection plane.

We can gain some insight into the warp equation by examining its terms. The first term on the right side of Equation 1.4 is a pure projective warp. The second term on the right side of Equation 1.4 expresses the 3D warp's perturbation from a projective warp—in other words, the translational component of the 3D warp. It is the presence of this second term that distinguishes the 3D warp from other image warps. One limiting case of the 3D warp is reference-image pixels, $\bar{u}_1$, with $z_1 = \infty$. For these pixels, the second term in the warp equation becomes insignificant, so the pixel is warped projectively.

The 3D warp equation is perhaps more easily understood in a different form. This form makes the symmetry between the reference image and destination image more evident, and is more obviously tied to our conventional representation of points in space:

$$\dot{C}_2 + \frac{\mathbf{P}_2}{S_2} z_2\, \bar{u}_2 = \dot{C}_1 + \frac{\mathbf{P}_1}{S_1} z_1\, \bar{u}_1. \tag{1.8}$$

The equation can be interpreted as saying, "The 3D location of a point as specified by the destination image is the same as the 3D location of that point as specified by the input image". The similarities to a standard computer graphics transform can be seen even more clearly when I expand $\bar{u}_1$ and $\bar{u}_2$ and rearrange slightly:

$$\dot{C}_2 + \frac{\mathbf{P}_2}{S_2} \begin{bmatrix} z_2\, u_2 \\ z_2\, v_2 \\ z_2 \end{bmatrix} = \dot{C}_1 + \frac{\mathbf{P}_1}{S_1} \begin{bmatrix} z_1\, u_1 \\ z_1\, v_1 \\ z_1 \end{bmatrix}. \tag{1.9}$$

My equations describing the 3D warp differ from McMillan's usual equation ([McMillan97], Eq. 3-10) in two ways. First, I use true equality rather than equivalence to a scale factor (projective equality). In a post-rendering warping system, we know the camera parameters precisely, so there is no need to use the weaker projective equality. In fact, I require the true equality in order to compute correct displayed-image depths, which my system needs to combine multiple warped images.

The second difference between Equation 1.4 and McMillan's equation is that Equation 1.4 uses Z-depths rather than disparity values (δ). For known camera parameters, this difference is minor, since the two can be easily exchanged:

$$\delta \equiv \frac{S}{z} \tag{1.10}$$

By making this substitution in Equation 1.4, we get an equation more like McMillan's, but that still uses true equality:

$$\frac{\bar{u}_2}{\delta(\bar{u}_2)} = \mathbf{P}_2^{-1}\mathbf{P}_1 \frac{\bar{u}_1}{\delta(\bar{u}_1)} + \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2) \tag{1.11}$$

1.3.3 3D warp coefficients

We can calculate u_2 and v_2 from $\alpha\bar{u}_2$, where α is an arbitrary scale factor, by performing divisions. These divisions convert homogeneous coordinates to image coordinates. Starting with Equation 1.11, then performing these divisions, abandoning matrix notation, and multiplying by $\frac{\delta(\bar{u}_1)}{\delta(\bar{u}_1)}$, we get the following equation (described in [McMillan97]):

$$\begin{aligned} u_2 &= \frac{w_{11}u_1 + w_{12}v_1 + w_{13} + w_{14}\delta(\bar{u}_1)}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta(\bar{u}_1)} \\ v_2 &= \frac{w_{21}u_1 + w_{22}v_1 + w_{23} + w_{24}\delta(\bar{u}_1)}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta(\bar{u}_1)} \end{aligned} \tag{1.12}$$

The coefficients w_{ij} are defined as follows (note that my definition differs slightly from McMillan's, in order to preserve the correct scale factor for the computation of z_2 below):

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \equiv \mathbf{P}_2^{-1}\mathbf{P}_1 \qquad \begin{bmatrix} w_{14} \\ w_{24} \\ w_{34} \end{bmatrix} \equiv \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2) \tag{1.13}$$

In order to merge several warped images, we will need the displayed-image depth value. This value (z_2) can be computed in terms of the w_{ij} coefficients:

$$z_2 = \frac{S_2}{\delta(\bar{u}_1)}\left(w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta(\bar{u}_1)\right) \tag{1.14}$$

1.3.4 Epipolar geometry

Several algorithms in this dissertation rely on the epipolar geometry of the 3D warp. The epipolar geometry between a pair of images (in this dissertation, a reference image and the destination image), is described in [Faugeras93]. McMillan [McMillan97] applied the tools of epipolar geometry to his 3D warping work, and I have largely adopted his notation. In this section, I will briefly describe the epipolar geometry of the 3D warp and the mathematical notation that I use to describe it.

Given a pair of images, the epipole in the second image is defined as the projection of the first image's center of projection into the second image. The converse is true as well: the epipole in the first image is defined as the projection of the second image's center of projection into the first image. Figure 1.6 illustrates this definition. Mathematically, the epipole in the second image, $\vec{e}_2$, is defined in homogeneous coordinates ($\mathbb{P}^2$ coordinates) as:

$$\vec{e}_2 \equiv \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2). \tag{1.15}$$

Likewise, the epipole in the first image is defined as:

$$\vec{e}_1 \equiv \mathbf{P}_1^{-1}(\dot{C}_2 - \dot{C}_1). \tag{1.16}$$

The components of the vector $\vec{e}_1$ or $\vec{e}_2$ are defined as:

$$\text{where} \qquad \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \equiv \vec{e} \tag{1.17}$$

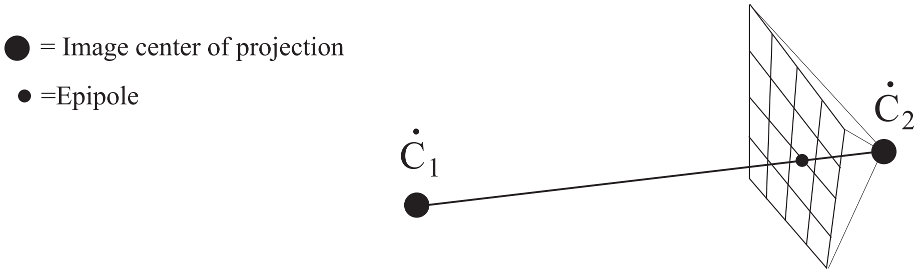


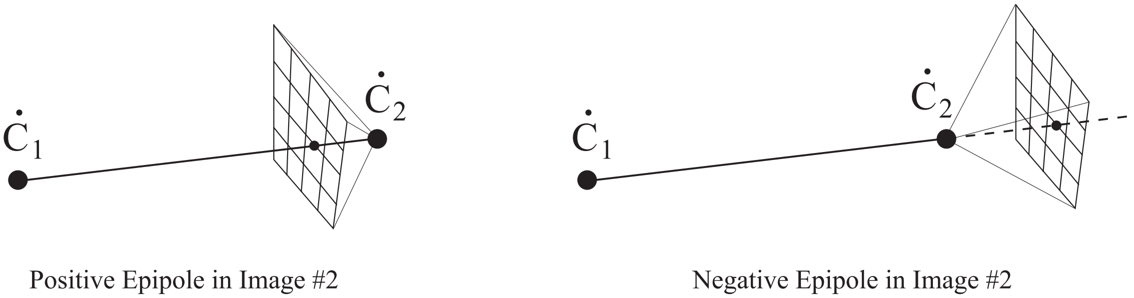
Figure 1.6: *Epipolar geometry of two images. The epipole in image #2 is the projection of image #1's center of projection (viewpoint) into image #2.*

When I wish to refer to an epipole in image-plane coordinates rather than homogeneous projective coordinates, I use the notation $\bar{e}$ (over-bar rather than over-arrow):

$$\bar{e} \equiv \begin{bmatrix} e_u \\ e_v \end{bmatrix} \equiv \begin{bmatrix} \frac{e_x}{e_z} \\ \frac{e_y}{e_z} \end{bmatrix} \tag{1.18}$$

Epipoles come in two types, distinguished by the location of the epipole's image plane with respect to the two centers of projection. Figure 1.7 illustrates the two possible types. Mathematically, the two types are distinguished by the sign of e_z. If $e_z > 0$, the epipole is referred to as a *positive* epipole, and if $e_z < 0$, the epipole is referred to as a *negative* epipole.

The image-space lines that pass through the epipole are referred to as *epipolar lines* (Figure 1.8). An important property of epipolar geometry is that there is a one-to-one mapping between epipolar lines



Positive Epipole in Image #2

Negative Epipole in Image #2

Figure 1.7: *The type of epipole (positive or negative) is determined by the location of the epipole's image plane with respect to the two centers of projection.*

in a first image and the epipolar lines in a second image. If a 3D point projects onto a particular epipolar line in the first image, then its projection in the second image is guaranteed to fall on the corresponding epipolar line in the second image.

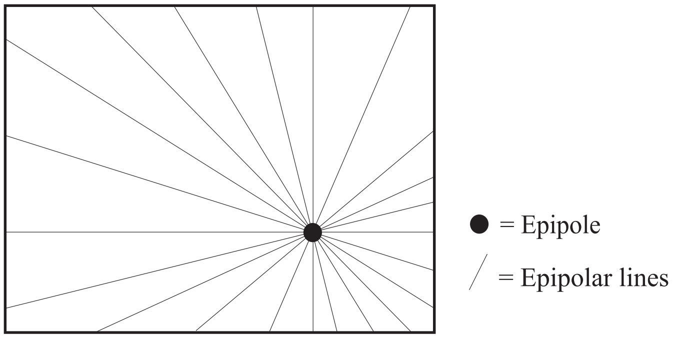


Figure 1.8: *The epipolar lines in an image pass through the epipole.*

McMillan showed that the epipolar geometry for a pair of images (reference and destination) can be used to perform a 3D warp that resolves occlusion using a painter's algorithm [McMillan97]. The algorithm requires that the reference image be traversed (and its pixels warped) in an *occlusion-compatible* order. There are two key properties of an occlusion-compatible order:

- The relative warp order for two reference-image points on *different* reference-image epipolar lines does not matter. (This property only holds strictly true for a continuous image; discrete images introduce complications.)

- The points on a *single* reference-image epipolar line must be warped in a particular order. If the epipole is positive, then the traversal must move along the epipolar line from the edge of the image towards the epipole. If the epipole is negative, then the traversal must move from the epipole towards the edge of the image. Figure 1.9 illustrates these traversals.

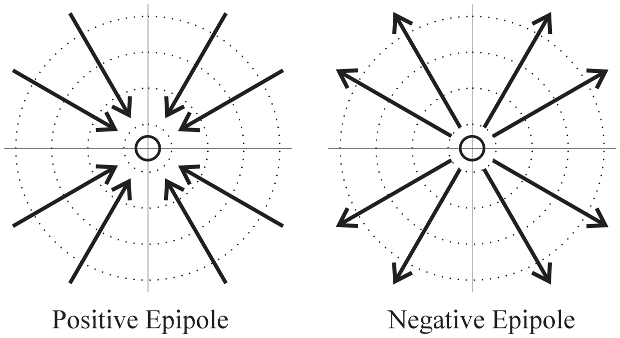


Figure 1.9: *The back-to-front occlusion compatible order moves towards a positive epipole and away from a negative epipole.*

McMillan developed a particularly simple reference-image traversal which satisfies these properties. The reference image is divided into four *sheets*. Each sheet is traversed in a raster-like order. Figure 1.10 illustrates this traversal for a negative epipole.

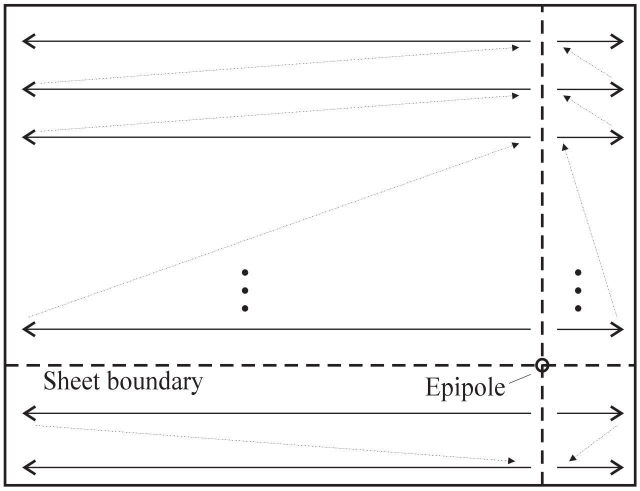


Figure 1.10: *The reference image can be divided into four occlusion-compatible sheets. Each sheet is traversed in a raster-like order.*

1.4 Summary

In this chapter, I introduced the rendering-acceleration problem that I am attacking, and described my approach at a high level. This chapter also summarized the results of the dissertation, and outlined the structure of the follow chapters. Finally, I introduced the mathematical notation and terminology for the 3D warp that I will use throughout the dissertation.

CHAPTER 2

BACKGROUND

This chapter reviews earlier work related to this dissertation. In the first part of the chapter, I discuss the basic types of image-based (or what I call sample-based) rendering. In the second part of the chapter, I discuss previous efforts to use sampled-based rendering techniques to accelerate rendering and to compensate for rendering-system latency. In other words, the chapter is divided into a part about the fundamentals of sample-based-rendering and an applications part.

2.1 Sample-based rendering

In the past few years, there has been enormous interest in the idea of replacing geometric models of scenes with image-based models of scenes. These ideas have been generally referred to using the terms *image-based modeling* and *image-based rendering*. I will instead use the terms *sample-based modeling* and *sample-based rendering*, because I believe these terms are more appropriate. The use of the term “image” is unnecessarily restrictive, since it implies that samples lie on a 2D grid.

There are two important potential advantages to the sample-based approach to 3D graphics. The first is that one can avoid the step of explicitly creating a 3D model of the scene. Instead, the model is implicitly represented by the samples acquired by cameras or other devices. This approach can be significantly less time consuming than creating an explicit 3D model. The sample-based representation can also capture subtle lighting effects that are difficult to represent in an explicit 3D model. This advantage of sample-based representations is not relevant to the work in this dissertation, since we assume the existence of an explicit 3D model that is fed to the conventional rendering pipeline.

The second potential advantage of sample-based approaches is in rendering speed. The regularity and fixed resolution of the sample-based representation are amenable to efficient rendering. It is these properties that I exploit in this dissertation. With a sample-based representation, the rendering work is bounded by the sampling density. Typically, this sampling density will be chosen to closely match the display resolution. In contrast, rendering a explicit 3D model requires work that grows with the number of geometric primitives in the model. The number of primitives is not necessarily restricted by the display resolution.

Sample-based rendering has been used in different forms for a long time. Classical texture mapping is the simplest example of sample-based rendering. In image-based texture mapping, a 2D reference image (the texture) is mapped onto a continuous surface. For a planar surface such as a polygon, the mapping function from the 2D texture-image space to the 2D screen space is a perspective transform. A perspective transform from one image space to another (sometimes referred to as a perspective image warp) takes the form shown in Equations 2.1. In these equations, (u_1, v_1) represent coordinates in one image, (u_2, v_2) represent coordinates in a second image, and w_{ij} represent constants:

$$\begin{aligned} u_2 &= \frac{w_{11}u_1 + w_{12}v_1 + w_{13}}{w_{31}u_1 + w_{32}v_1 + w_{33}} \\ v_2 &= \frac{w_{21}u_1 + w_{22}v_1 + w_{23}}{w_{31}u_1 + w_{32}v_1 + w_{33}} \end{aligned} \tag{2.1}$$

Environment mapping [Blinn76, Greene86] extends the texture mapping technique to the problem of approximately representing reflections from an object. Environment mapping for polygonal surfaces also uses a perspective transform.

I often refer to the perspective transform and the simpler affine transform as 2D transforms. Although in texture mapping the samples are mapped onto a surface that is considered to reside in 3D, the samples and image transform are still fundamentally 2D. In contrast, I consider McMillan's warp to be 3D because points transformed by it behave as true 3D points. This 3D behavior is due to the 3D warp's use of a per-pixel depth value.

The perspective and affine 2D transforms have a number of important properties. In particular, the inverses of these transforms are easily calculated. Thus, the image warps that use these transforms can be implemented using either a forward mapping or an inverse mapping. For example, texture mapping is usually implemented using an inverse mapping. Wolberg's book [Wolberg92] contains a good explanation of the mathematics of perspective and affine transforms, including the concepts of forward and inverse mapping. His book also discusses texture mapping and the related technique of image morphing.

Levoy and Whitted were the first to extend sample-based rendering from 2D samples to fully 3D samples. Their system [Levoy85] uses 3D point samples as its fundamental display primitive.

The system converts geometric primitives to point primitives prior to rendering. The point primitives generated by this conversion can be rendered in any order.

With fully 3D samples, the reconstruction problem is much more difficult than it is for 2D samples. In particular, the mapping function from source samples to destination image is no longer easily inverted, so forward mapping techniques are usually used. Levoy and Whitted use a modified A-buffer algorithm in conjunction with forward mapping to resolve visibility and to blend samples that originate from the same surface. I will discuss their approach to the reconstruction problem in more detail in Chapter 4.

2.1.1 3D image warping

Levoy and Whitted's system generates its 3D samples algorithmically from geometric primitives. But, 3D samples can also be retrieved from an enhanced form of a 2D image. To create this enhanced 2D image, one adds a depth component to each pixel of the image. Each pixel then represents an independent 3D sample. The resulting reference image is called a *depth image*. New images can be synthesized from the 3D data in the reference images.

The pinterp utility [Ward90] in Greg Ward's Radiance package uses depth images as input to synthesize new images. His system performs a full 3D transform (what I now call a 3D warp) of each pixel to generate the new images. Chen and Williams used depth images as input to their view interpolation system [Chen93], although their system is really a hybrid between a 2D and 3D image warping system. I will discuss both of these systems in more detail in the second part of this chapter.

McMillan and Bishop used 3D samples from pre-computed depth images to generate new images in real time. Their first system [McMillan95a] used a planar-image to planar-image 3D warp similar to that described in Chapter 1. Their second system [McMillan95b] used a cylindrical-image to planar-image warp. McMillan derives and describes the 3D warp equations in his dissertation [McMillan97]. The 3D warp equations given in Chapter 1 of this thesis are modified versions of McMillan's equations.

In order to achieve real-time 3D warping performance, McMillan uses an incremental evaluation of the 3D warp equations. He also developed an algorithm (described in Chapter 1) to resolve occlusion without Z-buffering. The algorithm, designed for a single reference image, employs a traversal order that resolves occlusion using a painter's algorithm.

I have used McMillan and Bishop's work as the starting point for the image-warping portion of my dissertation work. In particular, I use a planar-image to planar-image 3D warp throughout this dissertation. Planar depth-images are a particularly attractive type of reference image for post-rendering warping because they are easily generated by conventional rendering engines.

However, other researchers have developed a variety of other 3D sample-based representations. I will now discuss some of these representations.

2.1.2 Layered depth images

The planar depth-image can be extended by allowing multiple layers at each pixel. Each pixel consists of a linked list (or array) of layers, with each layer having its own color and depth. This representation has the advantage that it can represent surfaces in the world that are occluded from the reference-image viewpoint. Max and Ohsaki were the first to develop this representation, under the name *multi-layered Z-buffer* [Max95, Max96]. As this name implies, their system constructed the multi-layered Z-buffer by using a modified conventional renderer. In order to bound the size of the multi-layered Z-buffer, their system makes decisions about which layers to discard and which to keep at each pixel. In general, this decision is a difficult one to make in any sort of optimal way.

Shade *et al.* [Shade98] construct a multi-layered Z-buffer in a different manner which partially avoids this problem. They call their data structure a *layered depth image* (LDI), although it is very similar to Max's multi-layered Z-buffer. The LDI is constructed from a set of ordinary planar depth images. The system warps all of the planar images to the viewpoint of the LDI, and builds the LDI from the warped pixels. If any pair of layers at the same pixel in the LDI are within a small Z-tolerance of each other, then they are assumed to represent the same surface and are averaged. Some information can be lost in this step if a surface is better sampled in one of the planar images than it is from the viewpoint of the LDI.

By building the LDI from planar images, any surface that is visible from one of the planar depth images is represented in the LDI. The converse is true as well: Any surface that is not visible in one of these images will not be represented in the LDI. Thus, the problem of deciding which layers to store in the LDI is simplified to the problem of choosing a reasonable set of planar depth images.

I have not used LDI's for post-rendering warping for several reasons. The first reason is that generating compact LDI's directly (as Max and Ohsaki do) is difficult, because of the need to decide which layers to keep. A conventional graphics engine would need significant changes to generate LDI's directly. The alternative, generating LDI's from planar depth images, is more attractive, but adds additional complexity to a post-rendering warping system. A second warper would be required to generate the LDI's, since the primary warper is always busy generating output frames. Finally, an LDI discards some useful information that is present in the multiple planar images, unless modifications are made to the LDI data structure that would substantially increase its complexity.

An LDI representation would have some advantages. First, the output warper would not need to Z-buffer, because McMillan's occlusion-compatible order can be applied to a single LDI [Shade98]. Second, for most scenes the LDI representation would be more compact than dual planar depth images. Under less favorable circumstances (greater distance between reference-image viewpoints), [Popescu98] found that a typical LDI stored only 62% of the pixels that would be stored by two separate images. Memory bandwidth requirements (for input images) and warp computations are reduced accordingly. I have not felt that these advantages of the LDI representation outweigh its disadvantages in a post-rendering warping system.

2.1.3 Plenoptic-function representations

The sample-based representations discussed so far are closely tied to the notion of an "image". Researchers have also developed sample-based representations that are less closely related to conventional images. Most prominent among these are the approaches that are explicitly based on the plenoptic function.

The plenoptic function [Adelson91] is a five parameter function:

$$color = f(x, y, z, \theta, \phi) \tag{2.2}$$

The function provides the color seen when looking in a particular direction (θ, ϕ) from a particular point (x, y, z) in space. In a practical implementation, the allowed values of the function's independent variables are discretized, and colors are interpolated between these sample points. The plenoptic function can be extended to the time domain by adding time as a sixth parameter. Most sample-based representations can be cast as particular subsets of the plenoptic function, albeit strangely parameterized if geometric information such as per-sample depth is stored.

Levoy and Hanrahan, and Gortler *et al.* independently developed systems to represent objects in an outside-looking-in manner using a four-dimensional subset of the plenoptic function [Levoy96, Gortler96]. Gortler *et al.*'s system can optionally use additional information, in the form of per-sample depth, to bias the reconstruction process. Levoy and Hanrahan refer to the approach as *light field* rendering, while Gortler *et al.* refer to it as *lumigraph* rendering. The lumigraph/light-field representation in its pure form requires enormous amounts of storage (at least 1 GB uncompressed for complex objects [Levoy96]), which may limit its practicality in the near future.

Rademacher and Bishop have developed a sample-based representation which lies somewhere between a plenoptic-style representation and an image-style representation. They refer to their representation as a *multiple center-of-projection (MCOP) image* [Rademacher98]. This representation is image-like in that the samples are stored on a grid, and nearby samples are taken from similar centers of projection. It is different from a conventional image in that the center of projection can vary smoothly across the image, rather than remaining constant.

Neither the plenoptic-style representations nor the MCOP representation are well suited to post-rendering warping. The reason is that neither representation is readily generated at run-time by a conventional rendering engine. The plenoptic-style representations are also inappropriate because of their large storage requirements.

2.1.4 Volume rendering

Volume rendering is another form of sample-based rendering. However, its samples are of a different form—they represent the *contents* of 3-space directly, rather than the *appearance* of those contents from some particular direction. This distinction between volume rendering and other forms of sample-based rendering is not as clear-cut as it may seem at first. For example, a depth image represents the contents of spatial locations (as occupied or not occupied) as well as their appearance from a particular direction.

Research in volume rendering has traditionally emphasized the rendering of partially transparent volumes. In contrast, most image-based rendering work has emphasized the rendering of opaque surfaces.

I will not attempt to survey the volume rendering literature here. [Watt92] provides a good starting point for learning more about volume rendering.

2.2 Review of previous systems

In this second part of the chapter, I review some systems that use sample-based rendering techniques to accelerate rendering, reduce latency, or to display acquired imagery. I begin by discussing systems that both render and warp images in real time. These systems are the most directly related to the research presented in this dissertation. The second portion of this section discusses systems in which the rendering or image acquisition is performed off-line, but the rendering is still performed at run time.

Finally, I discuss a few systems that both render and warp off-line, but are nonetheless related to my work.

Throughout this part of the chapter, I will refer to different classes of image warps. The simplest warp that I discuss is an image shift. Affine warps, perspective warps, and 3D warps (per-pixel perspective warp) are progressively more complex. Wolberg's book [Wolberg92] provides a good overview of the first three types of warp. The 3D warp was discussed in Chapter 1.

2.2.1 Fully real-time systems

Image shifting for latency compensation

The simplest type of post-rendering image warping is image shifting, in which an offset is added to the X and Y image coordinates. These offsets can approximately correct for changes in the pitch and yaw rotation coordinates.[1] Optical image shifting was first proposed by Breglia *et al.* to compensate for rendering system delay in a system developed at the US Naval Training Equipment Center [Breglia81]. It was also used at about the same time by a US Air Force system developed by CAE Electronics [CAE84].

Predictive tracking can be combined with image shifting to provide even better latency compensation. The predictive tracking allows an approximately correct image to be rendered, and image shifting is used to correct for residual pitch and yaw errors. The combined technique results in less loss of field-of-view during periods of rapid head rotation and produces smaller errors for the degrees of freedom that image shifting does not correct for (translation and roll). Both the CAE Electronics system and a later version of the Naval Training Equipment Center system [Burbidge89] used predictive tracking in conjunction with image shifting. So and Griffin [So92], and Riner and Browder [Riner92] developed later systems using this same technique. Another system by Mazuryk and Gervautz [Mazuryk95] performs the image shifting in the rendering engine, before scan-out, rather than by using an optical image shifter after scan-out as the earlier systems did.

Wells and Griffin used image shifting to compensate for vibration of helmet-mounted displays [Wells84]. They shifted their output image by deflecting the CRT's display raster both vertically and horizontally during image scan-out.

[1] A quick definition of pitch and yaw: For a human head, *pitch* is the tilting motion to look up or down, and *yaw* is the turning of the head to the left or right.

Perspective warp

For a planar display, image shifts (or even affine transforms) are not sufficiently general to compensate for arbitrary rotations of the viewer. These warps are approximately valid for small rotations, but become less valid as the rotation angle increases. Full rotation compensation requires the use of a perspective image warp. Regan and Pose implemented a post-rendering perspective image warp in hardware, calling it an *Address Recalculation Pipeline* [Regan93, Regan94].

While a perspective warp can compensate for arbitrary head rotation, it can not generally compensate for head translation. The perspective warp can only compensate for head translation if all objects in the scene lie on a single plane (possibly at infinity). The objects in most scenes do not meet this criterion.

Layer-based systems

As mentioned earlier, a post-rendering image warp can be used to reduce apparent system latency and/or to increase the apparent frame rate. If the post-rendering warping system is being used only to reduce short (perhaps 100 msec) latencies, then an image warp that compensates just for rotation may be adequate. However, if the system is being used to increase the frame rate or to compensate for longer latencies, then it must use an image warp which can compensate for both translation and rotation. I conducted a simple experiment to demonstrate this point. I used a rotation-only (perspective) image warp to increase the frame rate in an architectural walkthrough from 5 frames/sec to 30 frames/sec. The displayed frames generated by this system were unacceptably jumpy.

So, any system that uses post-rendering image-based techniques to accelerate the frame rate must compensate for translation as well as rotation. The relative movement of an object in the scene due to head translation depends on its depth. Thus, any such system must implicitly or explicitly consider the depth of objects in the scene.

Previous work has approached this problem by dividing the scene into multiple layers. Typically, each layer contains objects of similar depths. The layers are independently rendered and warped. The warped layers are then composited to form the displayed image. Hofmann was the first author to suggest this approach [Hofmann88]. He proposed to use an affine warp to reduce the rate at which some parts of the scene must be re-rendered. He also discusses the conditions under which this technique provides an adequate approximation to the desired image.

Microsoft’s Talisman graphics architecture implements this idea [Torborg96]. Independent image layers are composited in front-to-back order using a per-layer affine transform at video rates. Any given image layer is re-rendered only when the residual error after applying its affine transform exceeds a desired threshold. Each layer’s affine transform is chosen to minimize error for the depth and velocity of the object(s) in the layer.

The front-to-back composition of the layers requires that a front-to-back order exist among the different layers. The existence of such an order implies that objects which reside in different layers, but overlap in screen space, do not inter-penetrate. It also implies that no occlusion cycles exist among three or more layers. If these conditions are not initially met, then the offending layers must be combined into a larger single layer. Managing the assignment of polygons to layers under these conditions is a difficult problem (although not an impossible one [Snyder98]), and forms a significant challenge when programming this type of architecture.

The requirement for a front-to-back order of layers can be relaxed if the composition stage uses Z-buffering. Schaufler’s *nailboards* technique [Schaufler97] extends the layering technique in this manner. Depth values are preserved within each layer during rendering, and used to compute an approximate depth value at each pixel during the image warp. The approximate depth value is used for Z-buffered compositing of the different layers. Note that the underlying image warp is still an affine warp—the depth value is only used to compute a new depth value, not to change the X or Y location of warped pixels. So, this technique eliminates the requirement that layers be non-overlapping, but still requires multiple layers in order to represent objects at substantially different depths in the scene.

Regan and Pose use a multi-layered technique that combines perspective warping with image composition. Objects are placed into different layers based on the minimum rate at which they must be re-rendered to avoid excessive image-plane error. Regan and Pose refer to this technique as priority rendering [Regan94]. In priority rendering, a frequently re-rendered layer will typically contain objects that are close to the viewer. However, such a layer can also contain objects that are far away, but require frequent re-rendering because they are moving rapidly. Because the layering is not based strictly on depth, the composition is Z-buffered rather than front-to-back as in Talisman.

The difference between the Talisman and priority rendering approaches to assigning objects to layers is subtle, but crucial. In Talisman, objects sharing a single layer are always at similar depths. Therefore, Talisman’s per-layer affine warp can approximately correct for viewpoint translation (using the average layer depth) as well as viewpoint rotation. In priority rendering, objects that share a layer can be at very different depths. Without the ability to assume a single depth, the perspective warp applied to each layer can not compensate for viewpoint translation at all. The warp only compensates for viewpoint rotation. As a result, priority rendering applies the *same* warp to all layers.

Recently, Shade *et al.* [Shade98] have demonstrated a layered system that uses an approximation to a 3D warp. Their "sprite" warping equations are of the form

$$u_2 = Au_1 + Bv_1 + Cz_1 + D, \tag{2.3}$$

where A,B,C, and D are constants. A similar equation is used to calculate v_2. This warp can be considered to be an affine warp which depends on z as well as u and v. The constants are chosen such that the warp is equivalent to a 3D warp at the center of the sprite; elsewhere on the sprite it is somewhat in error. Shade *et al.*'s sprite system only warps one reference image for each sprite, so it is only appropriate for smoothly changing surfaces which will not undergo substantial changes in visibility with movement.

Imposter systems

The systems discussed above apply image-based acceleration techniques to the entire scene, although some of them divide the scene into layers first. A different type of system uses image based techniques to accelerate the rendering of only certain parts of the scene. The remaining parts of the scene are conventionally rendered every frame. Typically, distant portions of the scene are represented in an image format, while nearby portions are conventionally rendered. The image-based representations of distant portions of the scene are often referred to as *impostors* [Maciel95].

These imposter systems can be classified into two categories. The first class of systems dynamically generates the image-based representations at run-time. The second class of systems generates the image-based representations in a pre-processing step. This second class of systems was developed first, but I will defer its discussion to the next subsection of this chapter.

Schaufler and Stürzlinger [Schaufler96a, Schaufler96b], Shade *et al.* [Shade96], and Aliaga [Aliaga96] have developed imposter systems that generate image-based representations at run-time. All of these systems combine their image-based representations with standard geometric primitives by texture mapping the images onto large polygons, then rendering the standard primitives. Thus, they are using a perspective image warp of cached, previously rendered images to display portions of the scene.

The systems developed by Shade [Shade96] and by Schaufler and Stürzlinger [Schaufler96b] automatically choose and generate their image-based impostors. The complete scene is represented in a spatially organized hierarchical data structure, and impostors can exist at multiple levels in the hierarchy. Impostors are re-generated when their image-space error exceeds a threshold. They are re-generated by rendering geometry and/or lower-level impostors as seen from the current viewpoint.

If the image-space error threshold for impostors is set too high, significant misalignment will occur between abutting geometry and impostors as the viewpoint moves. Aliaga's system [Aliaga96] eliminates this misalignment by morphing geometry vertices that are near an imposter to match the error in the imposter.

2.2.2 Real-time warping from stored images

My research develops an approach that is designed to both render and warp images at run-time. The previous subsection of this chapter discussed earlier work of this same type. Sample-based rendering can also be used in a different form, in which images are rendered (or acquired) in a pre-processing step. The pre-processed imagery is then warped at run-time. Many images must be stored in order to produce high quality output for a large range of viewpoints.

Panoramic systems

Lippman's movie-maps system [Lippman80] allows virtual movement through a city along certain routes. His system plays back images from a video-disc player as the user moves. He suggests, but does not implement, the use of both image scaling and projective transforms to interpolate between stored images.

Chen's QuickTime VR panoramic viewer [Chen95] allows discrete changes in viewpoint along with arbitrary rotation and zooming. The user can only translate to pre-defined points at which images have been acquired. This system stores its reference images (panoramas) as cylindrical manifolds. QuickTime VR uses a cylindrical-to-planar perspective-like warp to provide the rotation and zooming for each reference image.

Using per-pixel depth

By adding some form of explicit or implicit per-pixel depth information to the reference images, restrictions on translation can be removed. Systems developed by Greene and Kass, Chen and Williams, and McMillan and Bishop take this approach. I will describe each of these systems in turn.

Greene and Kass [Greene94] develop an approach for walkthroughs of static scenes which is a hybrid between geometric rendering and image-based rendering. Because image-based representations are prone to occlusion errors caused by objects close to the viewpoint, they separate the geometry for a given reference viewpoint into near and far regions. Only the far geometry is represented in image form. At run-time, the near geometry (presumably a small number of polygons) is rendered in the usual manner into the same Z-buffer as the image-based representation.

The "image-based" representation is itself a hybrid between a geometric and completely image-based representation. It is geometric in nature—it consists of a subset of the polygons in the far region—but the subset is chosen using an image. The technique retains only those far polygons which are visible in a Z-buffered image from the reference viewpoint. To minimize cracking, polygons which occupy a pixel in this image but whose true projected screen-space area is smaller than a pixel are enlarged to fill the pixel.

To represent an entire geometric model, many of these image-based representations are arranged on a 3D grid. At display time, the eight image-based representations with viewpoints closest to the desired final viewpoint are rendered into a Z-buffer to produce an image of the far geometry. The near geometry is then rendered into this same Z-buffer to produce the displayed image.

Chen and Williams take a more purely image-based approach in their view interpolation system [Chen93]. Their system uses pre-rendered reference images with per-pixel depth. But, rather than perform a per-pixel perspective transform (3D warp) at run-time, their system linearly interpolates the results of pre-computed 3D warps.

The pre-computed 3D warps are stored in what Chen and Williams refer to as morph maps. A morph map is associated with a particular reference image (with associated viewpoint) and a second, fiducial, viewpoint. The morph map holds the per-pixel, image-space movement vectors for a 3D warp of the reference image to this fiducial viewpoint. At run-time, a new image can be produced for any viewpoint along the line between the reference image viewpoint and the fiducial viewpoint. The new image is produced by moving each reference-image pixel in the direction specified by its morph-map entry. The movement distance is calculated by linearly weighting the morph-map entry according to the position of the new viewpoint along the line between the reference image viewpoint and the fiducial viewpoint. The depth value for the new pixel can be similarly interpolated, if a depth interpolation value is included in the morph map.

The new pixel locations are not perfectly correct except under certain special conditions, since a perspective transform is not linear with respect to the position along the line between source and destination viewpoints. However, if the reference image viewpoint and the fiducial viewpoint are close to each other (compared to the distance to the nearest object in the scene), the linear approximation is quite good.

Because this type of warp uses per-pixel depth, albeit indirectly, warping only a single reference image can introduce occlusion artifacts. The view interpolation system addresses this problem by always warping two reference images. Each reference image has a single morph map. The fiducial viewpoint for the first reference image's morph map is chosen to be the viewpoint for the second reference image, and vice-versa. Thus, a new image can be produced for any viewpoint along the line between the two reference image viewpoints.

Chen and Williams discuss, but do not implement, the possibility of extending their technique to four reference images whose viewpoints are arranged in a tetrahedron. By associating three morph maps with each reference image—the fiducial viewpoints are the other three reference image viewpoints—a new image can be produced for any viewpoint within the tetrahedron. The translation vector for a pixel from a particular reference image would be computed using barycentric interpolation of the corresponding translation vectors from the reference image's three morph maps.

For greater efficiency, the view interpolation system groups blocks of reference image pixels together using a quad-tree decomposition of the reference image. A block contains pixels with similar depth values, and thus requires only a single morph-map entry.

When combining multiple reference images, visibility can be resolved by Z-buffering of the interpolated Z values. But for greater efficiency, the view interpolation system uses a view-independent visibility order. This visibility order is computed for a pair of reference images in a pre-processing step. The Z-values for all of the pixel blocks are transformed to the first image's coordinate system, and then the blocks are sorted in back-to-front order. The visibility order remains valid as along as the view angle remains within 90 degrees of the first reference image's view angle.

Chen and Williams were the first to discuss the reconstruction problems caused by image warps that use per-pixel depth values. Their system uses a one-pixel reconstruction kernel, which resulted in frequent holes. Some holes are filled incorrectly by background objects that showed through the holes. Any pixels that remain unfilled are filled in a post-process step by interpolating colors from neighboring non-hole pixels.

McMillan and Bishop use a full 3D warp in their plenoptic modeling system [McMillan95b]. Thus, their system correctly handles arbitrary viewpoint translation (except for occlusion artifacts). The system uses cylindrical reference images, which are generated in a pre-processing step from many planar images. The planar images are acquired from the real world. McMillan and Bishop's earlier stereo-display system [McMillan95a] directly warps planar reference images generated by a ray tracer. Both of these systems warp only one reference image at a time, so occlusion artifacts become severe as the viewer moves away from the reference-image viewpoint.

McMillan and Bishop introduce a view-independent rendering order that resolves occlusion relationships between pixels from a single reference image using a painter's algorithm. This technique avoids the expense of Z-buffering, but is not easily generalized to resolving occlusion relationships between pixels from *different* reference images.

McMillan's dissertation [McMillan97] discusses the problem of reconstructing a new image from the warped reference-image samples. I will discuss this work in more detail in the reconstruction chapter of this dissertation.

Mesh-warping systems

In a typical reference image, significant portions of the image are planar or approximately planar. Several researchers have used this property to avoid a full 3D warp of every pixel in the image. Instead, they triangulate the image into approximately planar regions in a pre-processing step. The result of the triangulation is a texture-mapped triangle mesh with discontinuities at most object boundaries. The meshing algorithm must decide where to insert discontinuities in the mesh. This problem is essentially the same one encountered in 3D warp reconstruction. To produce a new (warped) image, the triangle mesh is conventionally rendering using texture-mapping hardware.

In work concurrent with mine, Darsa *et al.* [Darsa97] use this approach to generate images of a model originally represented in geometric form. In a pre-processing step, the model is ray-traced to produce reference images with per-pixel depth. Then, each reference image is triangulated along depth discontinuities to produce a triangle mesh with discontinuities. At run-time, two such reference-meshes are rendered to produce the displayed image. Darsa *et al.* explore several different approaches for blending the contributions from two reference meshes when they both sample the same surface.

Sillion *et al.* [Sillion97] use a similar approach as part of an imposter system for viewing urban environments. Near geometry is represented as geometry, and far geometry is represented as a triangle mesh. This system uses only one reference mesh at a time, so occlusion artifacts can appear fairly easily. Aliaga *et al.* [Aliaga98] also use this approach to image warping as part of their massive model rendering system. Although their system uses depth discontinuities to guide the meshing, it does not insert breaks in the mesh at these discontinuities. Occlusion artifacts thus appear in the form of false “skins”.

Imposter systems

The three imposter-based systems just described use reference meshes computed in a pre-processing step. Several researchers have also built imposter-based systems that use more strictly image-based representations. Some of these systems compute the images at run time, and were discussed earlier in this chapter. Other systems compute the images in a pre-processing step. I will briefly describe these systems next.

Maciel and Shirley [Maciel95] represent objects and clusters of objects with texture maps. There may be many different texture maps for the same object or cluster. Each such map is valid for a different range of viewing angles. The geometry and texture database is organized as an octree. Impostors may exist at multiple levels of the hierarchy. When a portion of the octree is far away, an imposter representing a large volume will be used. As the viewer moves closer, the system uses more than one imposter—or even the actual geometry—to represent this same volume.

Aliaga and Lastra [Aliaga97] adapt this technique to the specific case of architectural models. They use perspective-mapped textures to represent nearby cells (rooms) in an architectural model. Each view of a room through a portal (inter-room opening) is represented by many pre-computed textures. Each of these textures is valid for a small range of viewing angles. At run-time, the texture with the closest match to the current viewing angle is chosen to represent the geometry visible through the portal.

By using 3D image warping instead of perspective image warping, the number of pre-computed reference images needed by this type of portal-warping system is greatly reduced. Rafferty *et al.* describe this approach [Rafferty98]. They use a variant of the two-reference-image approach developed in this dissertation to reduce occlusion artifacts. Their system is optimized for speed, so they use a fixed splat size for reconstruction. They also do not Z-buffer, which can cause incorrect visibility under some conditions. In particular, if a surface is visible in the first reference image, but not visible in the second reference image, the surface can be incorrectly overwritten in the displayed image by a surface that is farther away.

Other systems

Shade *et al.* developed a system that uses layered depth images as an intermediate representation of a scene [Shade98]. Their system takes as input a set of ray-traced planar depth images. At run-time, the system builds an LDI from a changing subset of these images in a low-priority thread. A high-priority thread warps the LDI to produce output images. The system produces 300x300 pixel output images at 8-10 frames/sec on a 300 MHz Pentium II.

Pulli *et al.* have developed an outside-looking-in system for displaying images of acquired objects [Pulli97]. A preprocessing step generates a textured depth mesh from depth images. At run-time, the system generates an output view using the three closest (by viewing angle) textured depth meshes. The meshes are rendered into independent images, then composited to form the output image. The compositing step performs Z-buffering, but blends close surfaces rather than making a binary decision. The weighting factor for the blending is based on viewing angle differences, sampling density, and distance from the edge of the reference image.

Dally *et al.* developed an outside-looking-in system that uses a hierarchical structure to efficiently represent a large number of pre-acquired reference images [Dally96]. The lower levels of the hierarchy contain only the information that can not be obtained by warping the reference images represented at higher levels of the hierarchy. This system does not run in real-time, but the technique is intended to eventually run in real-time by using hardware acceleration. The system uses Z-buffering to resolve occlusion between different warped reference images. It employs a frequency-based representation of reference image tiles to facilitate filtering and compression.

The Lumigraph [Gortler96] and Light Field [Levoy96] systems, discussed earlier, perform their rendering in real-time. These systems require a very large number of images for input, in order to build a discretization of the 4D free-space plenoptic function.

2.2.3 Off-line warping

A wide variety of systems have been built that acquire (or render) and warp images off-line. In this subsection, I will review a few of these that are most closely related to post-rendering warping. Generally I have chosen to discuss those systems that generate their reference images by rendering, rather than acquiring them from the real world. However, the computer vision literature contains many examples of systems that acquire their reference images from the real world (e.g. [Szeliski96]).

Several authors have used a 3D warp (pixel re-projection) to accelerate stereoscopic ray-tracing and ray-tracing of animations. This problem is similar in several respects to the one addressed in this thesis, but there are several differences. These systems operate off-line, so the viewpoint path is always known. Also, these systems produce their reference images using ray-tracing rather than polygon rasterization hardware. As a result, holes in the warped images can be filled by ray-tracing the hole pixels. Polygon hardware does not provide an analogous method for easily filling just a few pixels without generating the whole image.

Badt [Badt88] used pixel re-projection to speed up ray tracing of animations. His technique preserves the first-level ray's hit point for each pixel in the current frame, and projects that hit point into the next frame. Any 5 by 5 pixel areas of the next frame that are significantly under-sampled or over-sampled with respect to the current frame are considered to be suspicious (often they represent a hole) and are re-ray-traced. Isolated missing pixels are either ray-traced or interpolated from nearby pixels.

Adelson and Hodges used a 3D warp to inexpensively perform stereoscopic ray-tracing [Adelson93]. Left-eye pixels are re-projected into the right-eye image. Their paper derives conditions under which re-projected pixels can be guaranteed to be correct (in terms of visibility). The remaining pixels are ray-traced.

Adelson and Hodges later extended their re-projection work to accelerating the ray tracing of animations [Adelson95]. In this work, the intersection point of each pixel's ray with a surface is saved, along with normal vector, diffuse color, and an object ID. To generate a subsequent frame, each such intersection point is re-projected, with the re-projection incorporating both object and camera motion information. A verification step checks that a ray from the new eyepoint to the new intersection-point location does not intersect any other objects (which might not have been visible in the previous frame). An enhancement phase calculates view-dependent shading, including the casting of reflective and/or refractive rays when necessary. Shadow rays may also need to be recast. The time savings of the algorithm results from potential elimination of the final, precise, object-intersection test, and by eliminating the need to test for ray intersections with objects behind the re-projection point. The algorithm fills the holes that are left after re-projection by ray tracing in the standard manner.

Greg Ward's *pinterp* program [Ward90] uses a 3D warp to generate new images from one or more reference images. It is typically used to generate short animations from a few ray-traced key frames that have associated per-pixel range values. The user of the program must choose the locations of the key frames. The program is capable of warping more than one reference image to produce a destination image. *Pinterp* resolves conflicts between multiple warped pixels using Z-buffering. Holes can be filled by ray tracing, or they can be filled with the color of the farthest-away (in Z) pixel at the boundary of the hole. Only the boundary pixels exactly in the $\pm u$ and $\pm v$ image-plane directions are considered as candidates to contribute the fill color.

Max and Ohsaki's system, mentioned earlier, uses an off-line 3D warp to generate new images of trees [Max95, Max96]. The system's reference images contain multiple layers at each pixel (i.e. they are layered depth images). The reference images are parallel projections, taken from different directions, rather than the perspective projections used by most other systems. The system uses deferred shading to avoid problems with view-dependent lighting, so normals are stored along with colors in the layered depth images. An A-buffer-style anti-aliasing scheme is used, with a constant color, depth, and normal associated with each subpixel coverage mask.

2.3 Summary

This chapter has described some of the previous research in sample-based rendering. It has also described other systems that use sample-based rendering to accelerate conventional rendering, or to reduce rendering-system latency.

None of these previous systems combines all of the following properties that characterize my system:

- The system is real-time, or intended to be real-time.
- The image warp is a 3D warp (requires depth images).
- The depth images are computed at run-time, rather than in a pre-processing step.
- The image warp is applied to the *entire* visible scene (represented by the conventionally rendered images), rather than to components of the scene.

The combination of these properties presents challenges not encountered by previous researchers.

CHAPTER 3

VISIBILITY

A single image generally samples only a subset of the surfaces in a scene. It samples those surfaces which are visible from the image's center of projection, and are within the field of view of the image. From a different center of projection, an entirely different set of surfaces may be visible. How then do we construct new images from a finite set of reference frames? And how do we choose the viewpoints for these reference frames?

In this chapter, I explore this question. The first section of the chapter discusses a variety of approaches to choosing reference-frame viewpoints. I then argue that, for post-rendering warping, two appropriately chosen reference frames contain most of the geometry needed to generate displayed frames.

Even with well-chosen reference frames, the visibility problem is not completely solved. There is often some geometry which should be visible in the displayed frame but is not visible in any of the reference frames. In such cases, the post-rendering warping system must choose some color to place in these displayed-frame *holes*. In the second section of this chapter, I develop an efficient algorithm for filling holes with estimated colors. This algorithm attempts to minimize *perceived* error.

In the third and final section of this chapter, I show how the method by which my system chooses reference-frame viewpoints is related to the epipolar geometry of the 3D warp. Understanding this relationship leads to a better understanding of how holes form. I also use this relationship to derive an expression for a bound on hole size.

3.1 Reference-frame viewpoints

3.1.1 Introduction to occlusion artifacts

In most circumstances, a single reference image is insufficient to avoid unacceptable occlusion artifacts. Figure 3.1 shows such a case. The rear surface is not visible in the reference image, but should appear in the destination image. As a result, if we warp the reference image to the destination viewpoint, we will get a destination image that does not contain the rear surface. Figure 3.2 shows this phenomenon for a real reference image and the corresponding destination image.

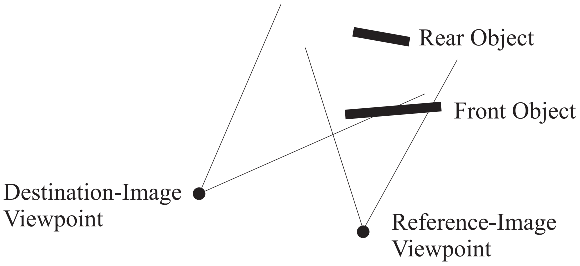


Figure 3.1: *A simple visibility example. The rear object is visible in the destination image, but occluded in the reference image.*

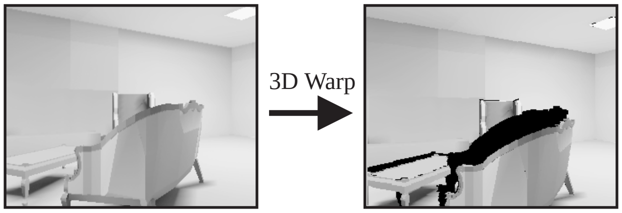


Figure 3.2: *The 3D warp can expose areas of the scene for which the reference frame has no information (shown here in black). For illustration purposes, the exposure shown here is exaggerated compared to the typical exposure from a single-reference-image post-rendering warping system.*

The type of geometric configuration shown in figure 3.1—an occluder in front of a background object—is the simplest scene configuration capable of causing occlusion artifacts. We can calculate the width of the occluded region using the 3D warp equation (Equation 1.4). We let $\bar{x}_1^a$ and $\bar{x}_2^a$ represent the location of point a in the source and destination images respectively. Likewise, $\bar{x}_1^b$ and $\bar{x}_2^b$ represent the location of point b in the source and destination images. The destination-image locations $\bar{x}_2^a$ and

$\bar{x}_2^b$ are calculated from the source image locations using Equation 1.4. Then, the width of the visibility hole in the destination image is:[1]

$$holewidth = \left\| \bar{x}_2^b - \bar{x}_2^a \right\| \tag{3.1}$$

One might ask, "What if we restrict ourselves to situations in which the hole size is insignificant?" Such a restriction would allow us to ignore the possibility of visibility artifacts. However, with such a strong restriction, there is no need to use a 3D warp, since a perspective warp would serve just as well. The very property of the 3D warp that makes it attractive—points move differently depending on their depth—is exactly what causes visibility artifacts.

Calculating the hole width using Equation 3.1 is not particularly insightful. By considering a particular case and approximating slightly, we can gain a better understanding of the behavior of visibility holes. Figure 3.3 illustrates such a case. In this particular case, the movement of the center of projection (from source to destination) is perpendicular to the view direction.

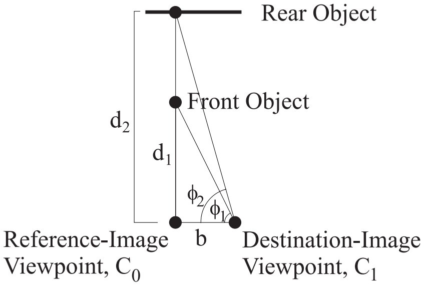

Rear Object
Front Object
d$_2$
d$_1$
ϕ_2
ϕ_1
Reference-Image Viewpoint, C$_0$
b
Destination-Image Viewpoint, C$_1$

Figure 3.3: *Dimensions for approximate calculation of visibility-hole size.*

From the figure,

$$\tan(\frac{\pi}{2} - \phi_1) = \frac{b}{d_1} \qquad \tan(\frac{\pi}{2} - \phi_2) = \frac{b}{d_2} \tag{3.2}$$

By making the small angle approximation $\tan(\theta) \approx \theta$, we can show that

$$\phi_2 - \phi_1 \approx b\left(\frac{d_2 - d_1}{d_1 d_2}\right) = \frac{b}{d_1}\left(\frac{1 - d_1}{d_2}\right) \tag{3.3}$$

[1] I neglect here the question of the screen-space orientation of the foreground object's edge, which is discussed in part three of this chapter. This equation is equivalent to a worst-case edge orientation, a perfectly reasonable possibility.

We can calculate the gap size in pixels from the gap size in radians, by using information about the display's field-of-view and resolution. With the (reasonably good) approximation that all pixels subtend the same solid angle, we get

$$pixelmovement \approx (\phi_2 - \phi_1) \cdot \frac{horizontalpixels}{horizontalFOV} \tag{3.4}$$

So, the hole size in pixels grows approximately linearly with movement distance. The hole size also grows as the viewpoint moves closer to the front surface, and as the distance from the front surface to the rear surface increases. This result is exactly what one expects intuitively.

By considering a particular example, we can get a feel for the magnitude of the hole size as well. For head movement of 0.1 m/sec[2], an inter-reference-frame time of 0.2 seconds, a distance of 2 m to the front object, and an infinite distance to the rear object, $\phi_2 - \phi_1 = .01$ radians. If the display is 640 x 480 with a 60° horizontal field of view ($\frac{\pi}{3}$ radians), then the hole will be six pixels wide.

If the scene consists only of a surface representable as a height field, then visibility artifacts need not necessarily result from a 3D warp. The reason is that a height field can be completely represented by a single (parallel-projection) reference image. However, in a general purpose post-rendering warping system we must assume that the scene is composed of more than just a single height field.

I have found that one source image is not sufficient to avoid serious visibility artifacts. In order to get acceptable quality warped output, we need additional information about the scene. A layered depth image ("LDI", discussed in Chapter 2) could provide this information, but standard renderers would require extensive modifications to generate LDI's. It is also difficult to choose which layers to keep when generating an LDI. So, I have chosen to use additional single-layered images. The post-rendering warping system renders these additional images using viewpoints that are different from those used to render the first image. Figure 3.4 illustrates this approach.

3.1.2 How many source images?

How does one choose the source-image viewpoints, and how many source images are necessary? Suppose we are given a volume of space containing all potential destination-image viewpoints (the destination-viewpoint volume). We would like to choose a set of source images such that all surfaces visible from the destination-viewpoint volume are represented in a source image. This problem is hard, in part because it is an inverse problem. Rather than asking which surfaces are visible from a particular

[2] The relatively slow head velocity of 0.1 m/sec is appropriate, since the movement direction is perpendicular to the view direction.

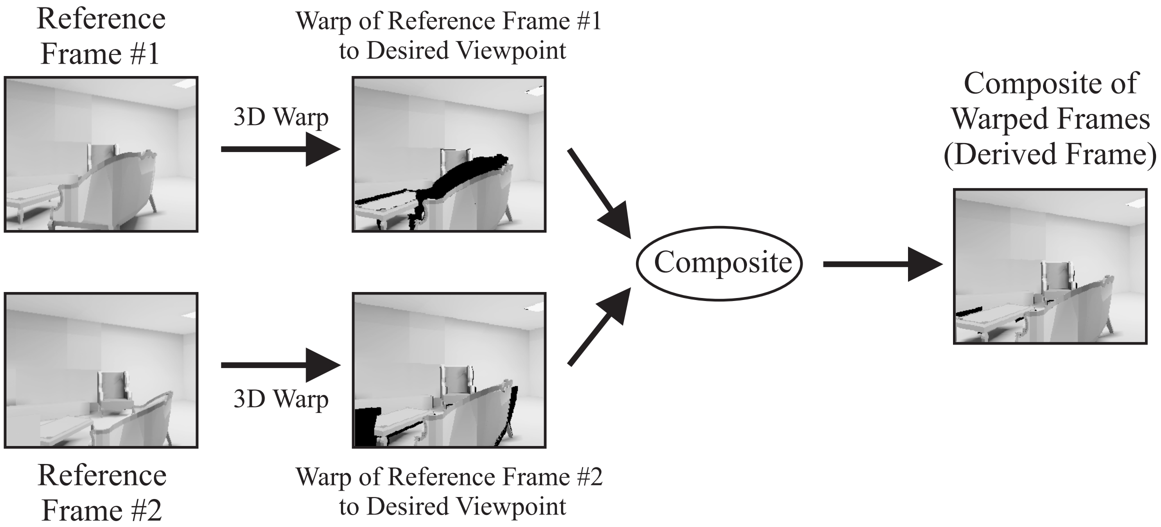

Reference Frame #1
Warp of Reference Frame #1 to Desired Viewpoint
3D Warp
Composite of Warped Frames (Derived Frame)
Composite
3D Warp
Reference Frame #2
Warp of Reference Frame #2 to Desired Viewpoint

Figure 3.4: *A single warped frame will lack information about areas occluded in its reference frame. Multiple reference frames can be composited to produce a more complete displayed frame. (For illustration purposes, the inter-viewpoint distances are much larger than normal, resulting in larger-than-normal-sized visibility holes).*

viewpoint, we are asking which viewpoints should be chosen such that certain surfaces are visible. In the worst case (for an arbitrarily complex scene), we would need an infinite number of source images.

Even typical scenes require extensive analysis to solve this problem, and can require a large number of reference images to guarantee that there will not be any visibility artifacts. Since my system must determine viewpoints in real time, such analysis is not feasible. Such analysis would also require very tight integration with the level of software that manages the geometric model as a whole (the scene-graph software layer), an imposition I consistently avoid in my system. Instead, my system needs a simple heuristic for choosing reference images that performs well for typical scenes and user motions. Additionally, in order to guarantee a constant output frame rate from my system, there must be a bound on the number of reference images that are warped to produce any particular displayed frame.

There are several tradeoffs involved in choosing the number of reference images that are warped to produce each displayed frame. First, using more reference images increases the computational expense (with my general approach), because all of the reference images must be warped every frame. Second, as we increase the number of reference images, we also increase the lifetime of each reference image. The reason for this tradeoff is that the conventional rendering engine only produces one reference image at a time. So, if we use four reference images, each reference image remains in use four times longer than it would if we were using only one reference image.

I considered using one, two, or more reference images. I quickly ruled out the use of one reference image, because occlusion artifacts are too severe with only one reference image. After considering both a two image approach and a greater-than-two image approach, I settled on the two-image approach for my test-bed system. Using two images eliminates most visibility artifacts, while limiting both the cost of warping and the length of time that reference images remain in use. I will first discuss the two-image approach, then the alternative approaches.

When using two reference images, there is a particular case under which we can *guarantee* that there will be no occlusion artifacts. This case, illustrated in Figure 3.5a, consists of two conditions. The first condition is that the scene contain only a single, non-moving, convex occluder and a background object. The second condition is that the destination-image viewpoint must lie on the 3-space line segment between the two reference-image viewpoints. I refer to this second condition as the *viewpoint-on-line* condition.

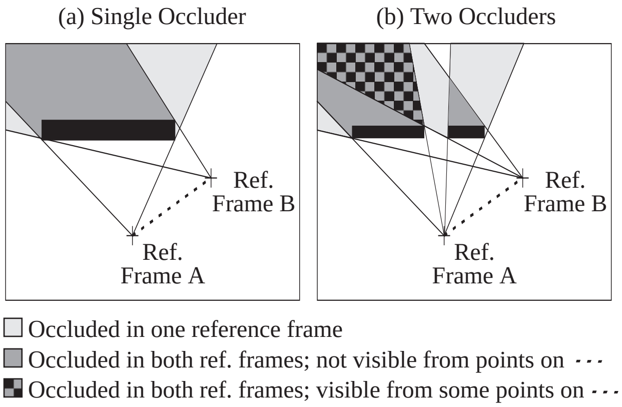

(a) Single Occluder
(b) Two Occluders
Ref. Frame B
Ref. Frame A
Ref. Frame B
Ref. Frame A
Occluded in one reference frame
Occluded in both ref. frames; not visible from points on
Occluded in both ref. frames; visible from some points on

Figure 3.5: *Point-on-line condition for a single occluder. For a single, non-moving, convex occluder, if a point is visible from a viewpoint on the line between A and B, then it is guaranteed to be visible from point A or point B. For multiple occluders no such guarantee can be made.*

When both of these conditions are satisfied, we know that any portion of the scene that is occluded in both of the reference images will also be occluded from the destination-image viewpoint.[3] Thus, no occlusion artifacts will be produced in the destination image.

[3] The following is a brief proof by contradiction. Let C be a point on line segment AB. Suppose there is a point P which is visible from C, but not from A or B. Then there are points G on line segment AP and H on line segment BP which reside within the occluder. Points A, B, C, P, G, and H are coplanar, so line segment GH must intersect line segment CP. This intersection point is labeled J, and must be outside the occluder, since line segment CP never intersects the occluder (by the given conditions). Thus, line segment GH is inside the occluder at G, then leaves it to pass through J, then re-enters it to pass through H. But, a line may not exit then re-enter a convex object. QED.

In real scenes there are of course many occluders, but we have found in practice that very few occlusion artifacts will appear if the viewpoint-on-line condition is satisfied. Intuitively, the reason is that with only a small viewpoint change, it is unlikely that the same portion of the scene will be occluded, then visible, then occluded again by a *different* occluder. Figure 3.5b illustrates this unlikely case. This figure depicts an extreme example, because the viewpoint-to-viewpoint distance is large relative to the viewpoint-to-object distance. In a local post-rendering warping system, the change in viewpoint results from the user's movement during a fraction of a second. As a result, the viewpoint change is generally quite small relative to the distance to the nearest object in the scene. For example, in 200 msec the viewpoint could move 0.2 m, but the distance to the nearest object is not usually less than 1.0 m.

I have just argued that satisfying the viewpoint-on-line condition is very effective at reducing occlusion artifacts. Thus, the problem of choosing reference-frame viewpoints for a two-image system reduces to attempting to satisfy this condition. It is also important to choose the reference-frame viewpoints such that the system gets maximum use (as many useful warps as possible) out of each reference frame.

My system meets these conditions by attempting to pick reference-frame viewpoints that lie on the user's path through space (Figure 3.6). Any particular displayed frame is generated from one reference frame with a viewpoint near a previous viewer position, and a second reference frame with a viewpoint near a future viewer position. The future viewer position is determined using a motion prediction algorithm. When the viewer passes this "future" viewpoint, the system starts using a new "future" reference frame, and discards the old "past" reference frame. I will discuss the timing of this system's image rendering and position prediction later in this chapter.

This viewpoint-selection algorithm gets maximum use out of the reference frames, because the future reference frame eventually becomes the past reference frame. It is this "reuse" of the reference frames that allows us to always warp and composite two reference frames, without having to render them any more often than if we were only warping a single reference frame.

In order to perfectly satisfy the viewpoint-on-line condition, the user's path would have to be exactly straight, and perfectly predictable. Obviously, these conditions do not always hold. But, both of these conditions are approximately true over short intervals of time (i.e. a first-order Taylor approximation to the movement is reasonably accurate). It is for this reason that using only two reference frames works so well in practice.

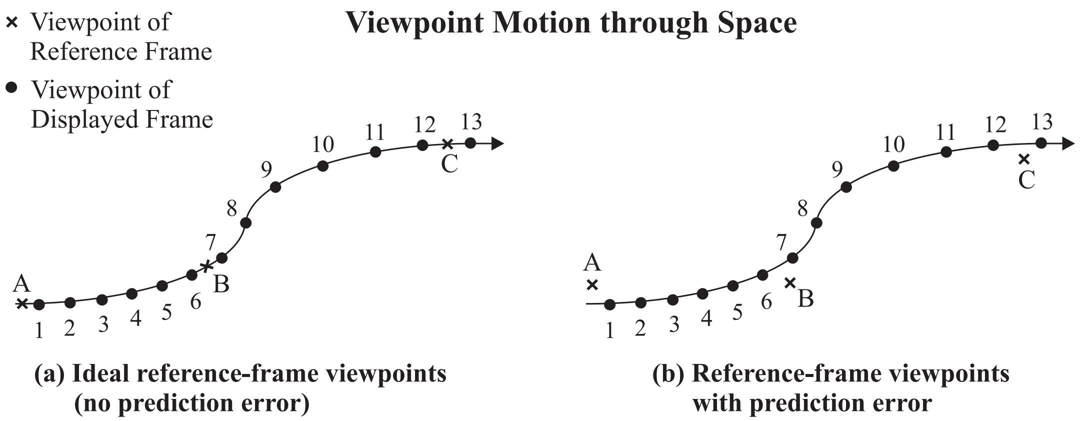
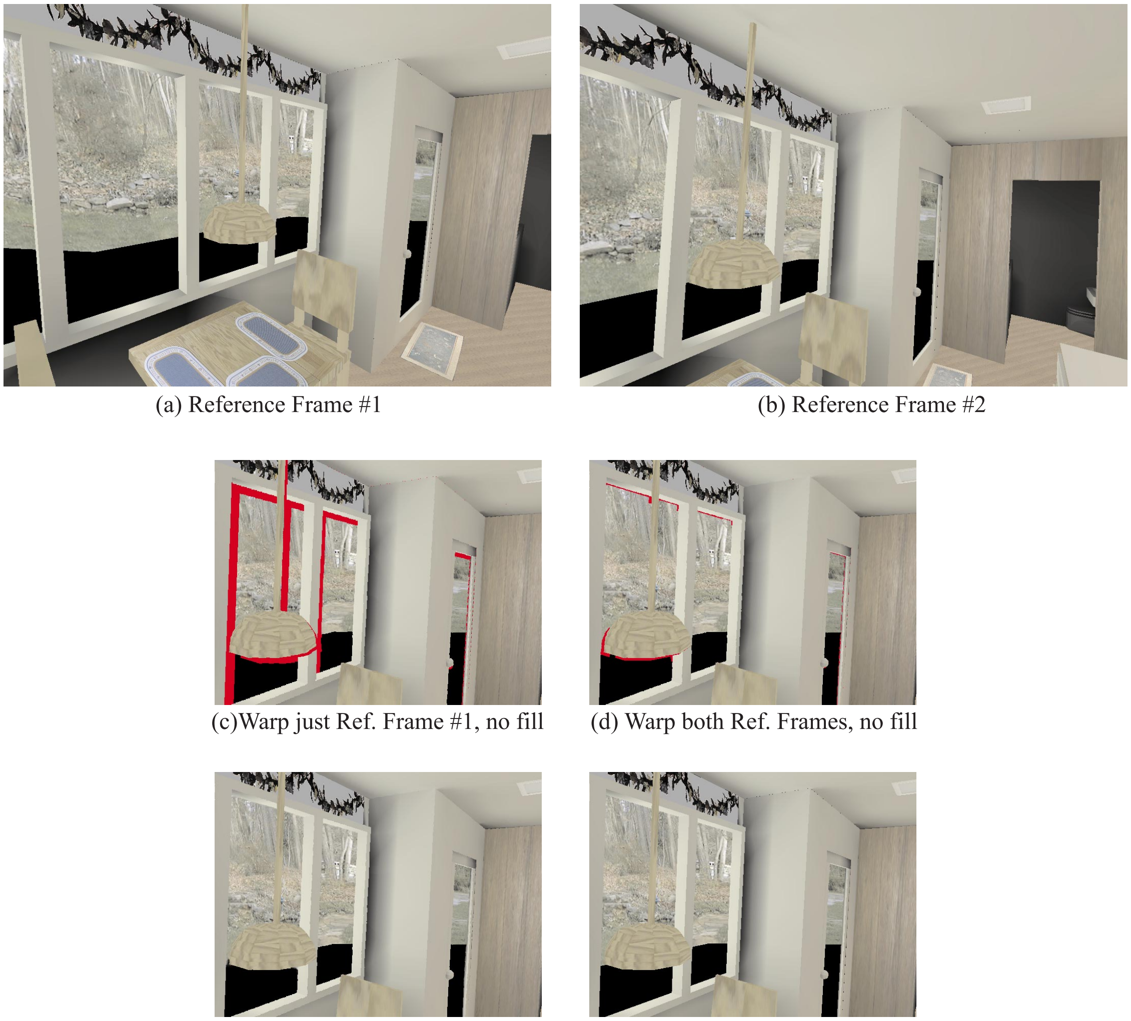

× Viewpoint of Reference Frame
● Viewpoint of Displayed Frame
Viewpoint Motion through Space
A
B
C
1
2
3
4
5
6
7
8
9
10
11
12
13
(a) Ideal reference-frame viewpoints (no prediction error)
(b) Reference-frame viewpoints with prediction error

Figure 3.6: *Displayed frames are computed by warping two reference frames, one near a past position of the viewer, and one near a future position of the viewer. For example, derived frame #5 is produced by warping reference frames A and B. Ideally the reference frames lie exactly on the viewpoint path, as shown in (a). But if future viewpoint locations are unknown, then viewpoint-motion prediction must be used to estimate them. As a result, the reference frames do not fall exactly on the viewpoint path, as shown in (b).*

Table 3.1 shows that two reference images are in fact better than one, especially when motion-prediction error is small. For a variety of system/model/user-path configurations, the table gives the percentage of pixels in each frame that are hole pixels. A hole pixel is a pixel at which the correct surface was not visible in any of the reference images. The color of hole pixels must therefore be estimated, using a hole-filling algorithm described later in this chapter. Figure 3.7 illustrates the effectiveness of using two reference images rather than one, and also illustrates the results of the hole-filling algorithm.

	Kitchen Path				AMR Path			
	1 Ref. Image		2 Ref. Images		1 Ref. Image		2 Ref. Images	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max
Actual prediction	0.74%	6.5%	0.39%	6.9%	—	—	—	—
Position-only pred.	—	—	0.39%	6.9%	—	—	—	—
1/2 error pred.	—	—	0.15%	3.9%	—	—	—	—
Perfect prediction	0.16%	3.0%	0.008%	0.4%	0.69%	2.43%	0.13%	0.61%

Table 3.1: *Severity of visibility holes under different conditions. The table entries indicate the percentage of the total pixels in each displayed frame that are part of a visibility hole. I obtained the kitchen path by walking through the kitchen model while wearing an HMD. Table 3.3 provides characteristics of this path. The path through the AMR is a spline path rather than a captured path, so position prediction is not relevant for it. For each combination of path and reference-image conditions, the table provides both the average percentage of hole pixels and the peak percentage (calculated from the worst frame). "Perfect prediction" gives the results that would be obtained with a perfect motion prediction algorithm (I generate these results by letting the simulator look into the "future"). "Actual prediction" gives the results that I obtained using simple velocity-based motion prediction. The velocities are obtained from a Kalman filter rather than being measured directly. "Position-only" prediction uses actual prediction for positions, and perfect prediction for orientations. The fact that these results are very similar to the "actual prediction" results indicates that field-of-view clipping is not a major cause of holes for these configurations. "1/2 error prediction" is like actual prediction, but with each prediction error exactly halved, by averaging with the known future values. The fact that these results are substantially better than the results from "actual prediction" indicates that an improved prediction algorithm would increase the quality of displayed frames. There are two anomalies that deserve discussion. First, the maximum value for kitchen-path "actual prediction" is slightly worse for two reference images than one. I attribute this anomaly to chance — two poorly predicted reference images are not necessarily better than one. Second, note that under equivalent conditions, the AMR sequence produces more visibility holes than the kitchen sequence. This difference is due to the greater occlusion complexity in the AMR model.*



(a) Reference Frame #1

(b) Reference Frame #2

(c)Warp just Ref. Frame #1, no fill

(d) Warp both Ref. Frames, no fill

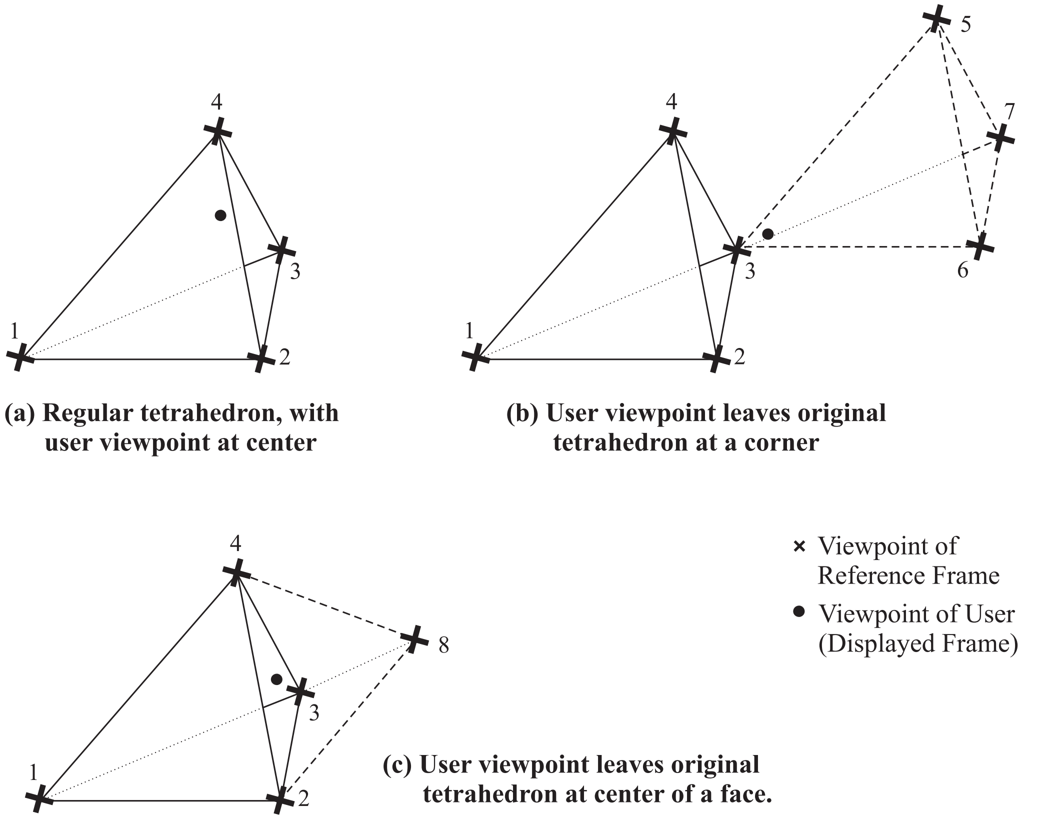
(e) Warp both Ref. Frames, with fill

(f) Conventional Rendering (the ideal)

Figure 3.7: *Different warping options. The two top images are adjacent reference frames from the kitchen walkthrough, generated at viewpoints specified by motion prediction. The four bottom images show various types of displayed frames. (c) was generated by warping only one reference frame. Visibility holes are shown in red. (d) was generated by warping both reference frames. Visibility holes are again shown in red. (e) was generated by warping both reference frames, then using the hole-filling algorithm described later in this chapter. (f) was generated using conventional rendering, and is presented for comparison purposes. Note that the same pixel–to–page-area scaling was used for all six images. The reference frames are larger because they have a larger field of view.*

3.1.3 Alternatives to two reference frames

There are a number of possible alternatives to using two reference images. One such alternative is to use four reference images. With four reference images, the single-occluder property can be extended to allow destination-image viewpoints anywhere in a particular *volume*. Specifically, we can guarantee (for a single convex occluder) that there will be no occlusion artifacts if the destination-image viewpoint lies anywhere within the tetrahedron formed by the four reference-image viewpoints. A system employing this strategy would probably attempt to choose the reference-image viewpoints so that they are maintained in an approximately regular tetrahedral configuration (Figure 3.8a).


4
3
1
2
5
4
7
3
6
1
2
4
8
3
1
2
× Viewpoint of Reference Frame
● Viewpoint of User (Displayed Frame)

(a) Regular tetrahedron, with user viewpoint at center

(b) User viewpoint leaves original tetrahedron at a corner

(c) User viewpoint leaves original tetrahedron at center of a face.

Figure 3.8: *Working with four reference-frame viewpoints. (a) Reference viewpoints are arranged in a regular tetrahedron. (b) If the user's viewpoint leaves the tetrahedron near a vertex, three new reference images (#5, #6, and #7) must be generated almost simultaneously. (c) If the user's viewpoint leaves the tetrahedron near the middle of a face, only one new reference image (#8) must be generated.*

The two-reference-image approach had a very elegant method for choosing new reference-image viewpoints. Only one reference image was updated at a time, but the bracketing of the user's position by the reference-image viewpoints could always be approximately maintained. Unfortunately, I have been unable to devise such an elegant approach for the four-image approach. The problem occurs when the user's viewpoint leaves the tetrahedron by passing through (or close to) one of the reference-image viewpoints. To maintain the desired regular tetrahedral configuration, three of the four reference-image viewpoints need to be nearly simultaneously replaced (Figure 3.8b). Such a simultaneous replacement is impossible with a conventional renderer that only produces one reference image at a time. Note that this problem does not occur if the user's viewpoint leaves the tetrahedron through the middle of a face. Then, only a single reference image needs to be replaced to maintain the regular tetrahedral configuration (Figure 3.8c).

Several solutions to this problem exist. One is to locate the reference-image viewpoints farther away from the user viewpoint. The goal is to prevent the user's viewpoint from ever approaching the boundary of the tetrahedron, or at least from approaching a vertex of the tetrahedron. But, in real scenes with multiple occluders, placing the reference-image viewpoints further away increases occlusion artifacts. These artifacts might be reduced by adding a fifth reference image that is maintained close to the user's viewpoint. Another solution to the four reference-image update problem is to handle the case shown in Figure 3.8b by moving the vertex near the user's viewpoint further away, thus stretching the tetrahedron.

These problems with choosing new reference-image viewpoints are closely related to the fact that we can only update each reference image half as often as we can when we are using only two reference images. Intuitively, one expects that the average distance from any particular reference image to the user's viewpoint will be greater than it is in the two-reference-image approach.

When the user's viewpoint is moving slowly or not at all, the four-reference-image approach becomes more attractive. The reason is that reference-image viewpoints can be located close to the user viewpoint without becoming rapidly obsolete. A promising approach that I have not implemented would be to adaptively vary the algorithm for choosing reference-image viewpoints based on the user's speed. When the user is moving rapidly, the system would use an algorithm similar to that discussed for the two-reference-image approach. When the user slows down, the system switches to a four-reference-image approach.

So far, I have discussed techniques for choosing reference-image viewpoints that are scene-independent—they rely only on information about the user's motion. I argued earlier that an automatic scene-dependent technique for general scenes would be too computationally expensive for a real-time system. However, for certain types of highly structured environments, it would certainly be possible to optimize reference-image viewpoint locations. In particular, the cells-and-portals structure of architectural models would be amenable to a scene-dependent approach. For example, extra reference image viewpoints could be located in the doors (portals) of the current room, where they would capture information about the adjoining rooms. These special reference-image viewpoints could be chosen by automatic means, manual means, or a combination of the two. Once again though, this sort of optimization requires a tighter coupling between the higher-level model-management software and the image warping system. I have not attempted to implement this technique.

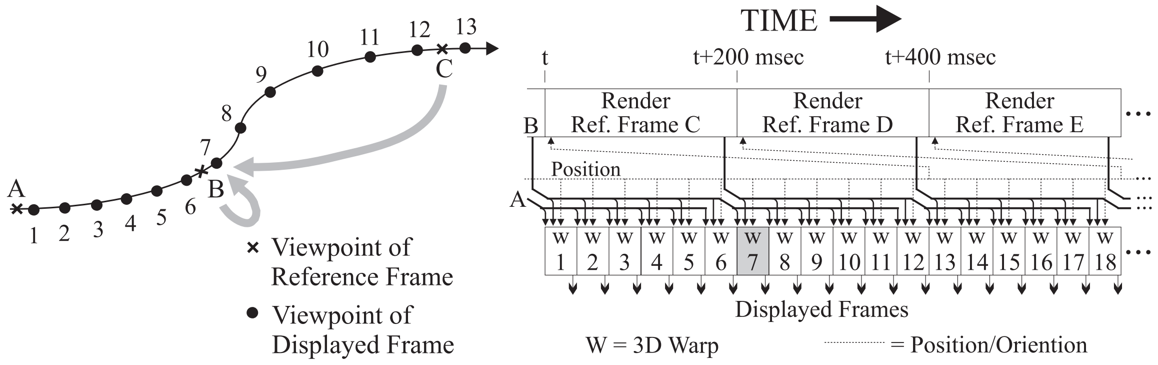
3.1.4 Motion prediction and system timing

The previous subsection noted that when we use two reference frames, we typically use motion prediction to help choose the reference-frame viewpoints. The motion prediction allows reference frames to be placed close to the point where they will be most useful. In particular, we can place reference images near a future viewpoint location.

The prediction interval—how far into the future we must predict the viewpoint position—is determined by the post-rendering-warping system's timing. Figure 3.9 illustrates this timing for a system that generates reference frames at 5 frames/sec, and displayed frames at 30 frames/sec. The left side of this figure shows that a reference frame must be available one reference-frame-time before the user reaches that reference frame's viewpoint. For example, reference frame C must be ready as soon as the user reaches derived-frame viewpoint #7.

This analysis might lead one to conclude that the prediction interval in this case is $1/5$th of a second. However, this conclusion would be incorrect. The reason is that conventional rendering of a reference frame is not an instantaneous process—the rendering must commence one reference-frame-time *before* the reference frame is actually needed. So, the prediction interval is approximately $1/5 \text{ sec} + 1/5 \text{ sec} = 2/5 \text{ sec}$.

The reference frames are staggered with respect to the displayed frames. For example, in Figure 3.9a, reference frame B is midway between displayed frames #6 and #7. Thus, the prediction interval for a reference-frame viewpoint is actually $1/2$ of a displayed-frame-time less than the two reference-frame-times previously used. In the case shown in Figure 3.9 the prediction interval is $23/60$ sec, not $24/60 = 2/5$ sec.



(a) Viewer's path through space **(b) System Timeline**

Figure 3.9: *System timing with reference frames rendered at 5 frames/sec and displayed frames generated at 30 frames/sec. (a) User's path through space. The grey arrows show that reference frames B and C are warped to produce displayed frame #7. (b) Rendering and warping time-line. The rendering of a reference frame must be complete before the reference frame is needed to produce a displayed frame. Note that position information is shown moving backwards in time in order to render the reference frames. This backwards movement is achieved using motion prediction.*

3.1.5 Prediction error

Generally, motion prediction is not perfect. As the prediction interval grows longer, the expected magnitude of the error in the prediction also grows. The characteristics of the prediction error will vary depending on how viewpoint motion is controlled. They will also change from application to application, since different applications require different types of motion.

Some common types of viewpoint control in computer graphics include precomputed paths, track-ball or joystick control, and head motion (for head-mounted displays). In considering motion prediction for post-rendering warping, I have concentrated on head motion. Unlike track-ball or joystick control, it is reasonable to discuss head motion in an application-independent manner, because the application-to-application variations are somewhat constrained by human anatomy.

There is a considerable literature on head-motion prediction (see [Azuma94, Azuma95b, Foxlin98] for examples and references to other work). I will summarize two important points. First, prediction error (at least for polynomial-type predictors) grows approximately as the square of the prediction interval [Azuma95b]. Second, peak errors are much larger than average errors. For one set of motion data, Azuma's 100 msec predictor [Azuma95b] had an average error of 3.6 mm and a peak error of 100.1 mm. These error measurements are per-axis measurements for one of the three translational axes.

For the "kitchen" walkthrough sequence used to generate Table 3.1, I used the UNC Tracker group's Kalman-filter based system [Welch97], without inertial sensors. This system provides estimates of pose (position and orientation) and pose first derivatives from which future poses can be predicted. I measured the prediction error for this walkthrough sequence in a PRW system that used a two-reference-frame post-rendering warp with a 383 msec prediction interval. The average position error (measured as a 3D distance) was 44 mm, and the peak error was 154 mm. The average angular prediction error was $5.9°$, and the peak error was $24°$. Table 3.2 provides a more in-depth analysis of the prediction errors from the kitchen motion path.

Error Type	Average	Maximum
Translation (mm)	44 (26, 20, 21)	154 (127, 114, 106)
Rotation (deg)	5.9 (3.4, 3.5, 2.0)	24 (20, 23, 12)

Table 3.2: *Prediction errors for the kitchen path, for a 383 msec prediction interval without inertial sensors. Each measurement is made for the worst-case direction, and also made on a per-axis basis. The per-axis measurements are given in parentheses. The three axes, in order, are the image-plane u, the image-plane v, and the view direction (perpendicular to image plane). Translations are along the specified axis, and rotations are about the specified axis.*

Prediction error is the biggest cause of visibility holes from our system, as Table 3.1 showed. The prediction error causes the reference frames to be incorrectly placed, so that the desired-frame viewpoints are no longer on the line between the reference-frame viewpoints. When perfect prediction is used, very few visibility holes are produced. These remaining holes are caused by either violations of the single-occluder principle, or by violations of the point-on-line condition caused by curvature of the true viewpoint path.

Better motion prediction would significantly improve the quality of the imagery generated by our system. Hans Weber, another researcher in our laboratory, believes that a 10-fold improvement in prediction accuracy (compared to Azuma's results) is possible, by incorporating knowledge of human anatomy into the prediction model [Weber97]. Better quality imagery should be achievable even without such advances in prediction models. The tracking and prediction system that I used did not incorporate inertial sensors, even though these sensors are already known to improve prediction accuracy [Azuma94]. I did not use a system that incorporated inertial sensors because such a system was not available to me when I gathered my motion data.

There are several properties unique to post-rendering warping which can be used to tune a motion prediction system. First, post-rendering warping is insensitive to high-frequency, low-magnitude prediction errors (jitter). Such errors are extremely bothersome when they are directly visible to users, as in systems which use only prediction (no warping) to compensate for latency [Azuma94]. In my system, the image warping hides these low-magnitude errors. As a result, the motion predictor can be tuned for greatest accuracy, even at the cost of some increase in jitter.

My system is most sensitive to position prediction errors in the direction parallel to the image plane. Such errors cause larger image-space holes than those perpendicular to the image plane. It is possible that a motion prediction system could take advantage of this property of post-rendering warping.

If latency is not a concern to the user of a post-rendering warping system, then the system can decrease prediction error by increasing latency. Such a system generates reference and displayed frames using delayed viewpoints. In the extreme case, the system can get “perfect” prediction by increasing the delay until it matches the normal prediction interval of the system. A completely pre-determined viewpoint path also allows a PRW system to use perfect prediction.

3.1.6 Field-of-view and rotation

Up until this point, I have not discussed the field-of-view (FOV) needed for the reference frames. Typically, the reference-frame FOV must be larger than the displayed-frame FOV, in order to allow for changes in the view direction (i.e. head rotation). If future head orientations can not be predicted with perfect accuracy, an additional reference-frame FOV margin must be added to allow for the orientation prediction error. Some additional FOV is also needed to allow for backward motion by the user, but this extra FOV is so small in comparison to that required by changes in view direction that I ignore it in my analysis.

Theory and results

The extra FOV (Δf degrees) required for the reference frames depends on three application-dependent and system-dependent constants. The constants are the maximum rate of view-rotation (R degrees/sec), the rate at which reference frames are generated (Q reference-frames/sec), and the size of orientation prediction errors (E degrees). R is actually defined more precisely as the maximum *average* rate of view rotation over the time period that a reference frame is used ($2Q$). For both horizontal (Δf_x) and vertical (Δf_y) directions,

$$\Delta f = E + \frac{2R}{Q} \tag{3.5}$$

This equation neglects rotation about the viewing direction, since I have observed that for an HMD this rotation has a negligible effect on the worst-case Δf. The equation also ignores the effect of translation, which requires slight further enlargement of the FOV to capture both foreground and background objects that move different distances across the FOV during the warp.

If we are willing to accept the possibility that one of the two reference frames might not cover the entire displayed-image FOV, then we can choose:

$$\Delta f = E + \frac{R}{Q} \tag{3.6}$$

We also might be willing to occasionally tolerate the situation in which neither of the reference frames completely covers the entire displayed-image FOV (I refer to this problem as *FOV clipping*). If we can occasionally tolerate FOV clipping, then Δf can be even smaller.

For the kitchen walkthrough sequence, the actual display is $60°$ horizontal by $47°$ vertical, and our system used reference frames of $88°$ by $68°$. Thus, $\Delta f_x = 28°$, and $\Delta f_y = 21°$. Table 3.3 summarizes the rotational and translational movements in the kitchen walkthrough sequence. Given the maximum 400 msec rotation rate of $53°$/sec from this table and the maximum prediction error in the kitchen sequence of $24°$ from Table 3.2 (earlier), Equation 3.5 recommends $\Delta f = 45.2°$. The less conservative Equation 3.6 recommends that $\Delta f = 34.6°$. The fact that there is very little FOV clipping in the kitchen sequence with $\Delta f_x = 28°$ indicates that the worst-case prediction error does not usually coincide with the worst-case rotation rate for both reference images.

For the auxiliary machine room (AMR) sequence, the actual display was again $60°$ horizontal by $47°$ vertical. The reference images were $75°$ by $52°$ ($\Delta f_x = 15°$ and $\Delta f_y = 5°$). The AMR sequence was able to use smaller Δf's because it used a known path (perfect prediction), and the rotation was smooth and slow.

A large reference-frame FOV is expensive to store and render. An increase in reference-frame FOV moves more geometry into the view frustum of the reference frames, requiring a greater geometry transform rate from the conventional renderer. Moreover, as the reference-frame FOV increases, the number of pixels in the reference frame must also increase to maintain a $1:1$ sampling ratio between the reference frames and displayed frames. At the end of this section, I consider relaxing the $1:1$

Movement Type	Average	Largest 400 msec avg	Maximum
Translation (m/sec)	0.29 (.11,.04,.25)	0.60 (.34,.16,.59)	0.75 (.47,.24,.72)
Rotation (deg/sec)	19 (16,5.1,4.7)	53 (53,19,26)	64 (63,31,30)

Table 3.3: *Rotation and translation characteristics of the kitchen walkthrough path. Each measurement is made for the worst-case direction, and also made on a per-axis basis. The per-axis measurements are given in parentheses. The three axes, in order, are the image-plane u, the image-plane v, and the view direction (perpendicular to image plane). Translations are along the specified axis, and rotations are about the specified axis. I calculated three types of statistics. The "average" is the average over the entire movement path. The "largest 400 msec avg" is calculated from the maximum-rate movement over a 400 msec interval. This interval corresponds to the two-reference-frame time interval ($2Q$) for a five reference-frame/sec system. The "maximum" is the maximum rate anywhere on the path (using a $\frac{1}{60}$ sec window).*

sampling restriction by adapting the reference-frame FOV as the head rotation rate changes, but the following analysis assumes that the $1:1$ restriction is in place.

Because $\tan(\theta) \neq \theta$, the increase in number of pixels is more than linear (in each dimension) with the increase in FOV. The increased number of pixels requires a greater pixel-fill rate from the conventional renderer and more memory for reference-frame storage. Note, however, that with appropriate clipping of the reference image (discussed in Chapter 5), the cost of the 3D image warp does *not* grow indefinitely with Δf_x and Δf_y. When the reference-frame FOV becomes large enough that the reference frame's edges are discarded by the clip test, further increases in the reference-frame FOV do not increase the cost of warping.

We can derive several equations that describe the cost of an increase in reference-frame FOV. We assume in these equations that both the displayed-frame and reference-frame image-planes are perpendicular to, and centered on, the view direction. In each dimension, the ratio of reference-frame pixels to displayed-frame pixels is:

$$\frac{axisRefPix}{axisDisPix} = \frac{\tan\left(\frac{1}{2}(f + \Delta f)\right)}{\tan\left(\frac{1}{2}f\right)} \tag{3.7}$$

Thus, the ratio of the total number of reference-frame pixels to displayed-frame pixels is:

$$\frac{refPix}{disPix} = \frac{\tan\left(\frac{1}{2}(f_x + \Delta f_x)\right)\tan\left(\frac{1}{2}(f_y + \Delta f_y)\right)}{\tan\left(\frac{1}{2}f_x\right)\tan\left(\frac{1}{2}f_y\right)} \tag{3.8}$$

One interesting result from Equation 3.8 is that when the horizontal FOV becomes too wide, the pixel count can be significantly reduced by splitting the reference frame into two smaller planar images sharing the same center of projection. This strategy halves the number of pixels when the horizontal FOV reaches 141°.

We can calculate the increase in the solid angle (in steradians) that falls within the view frustum due to an increase in FOV. We use the formula $solidAngle = 4 \arcsin\left(\sin\left(\frac{1}{2}f_x\right)\sin\left(\frac{1}{2}f_y\right)\right)$ for this calculation. This formula is derived in Appendix A. The increase in solid angle is:

$$\frac{refSolidAngle}{disSolidAngle} = \frac{\arcsin\left(\sin\left(\frac{1}{2}f_x + \Delta f_x\right)\sin\left(\frac{1}{2}f_y + \Delta f_y\right)\right)}{\arcsin\left(\sin\left(\frac{1}{2}f_x\right)\sin\left(\frac{1}{2}f_y\right)\right)} \tag{3.9}$$

Table 3.4 summarizes the FOV’s, pixel counts, and solid angles of the reference frames used for the kitchen and AMR walkthrough sequences.

	Kitchen Path	AMR Path
Disp FOV (f_x x f_y)	60° x 47°	60° x 47°
Ref FOV ($f_x + \Delta f_x$ x $f_y + \Delta f_y$)	88° x 68°	75° x 52°
Δf_x x Δf_y	28° x 21°	15° x 5°
Disp resolution	640 x 480	640 x 480
Disp pixels	307,200	307,200
Ref resolution	1070 x 745	851 x 538
Ref pixels (rel. factor)	797,150 (x2.59)	457,838 (x1.49)
Disp solid angle	0.80	0.80
Ref solid angle (rel. factor)	1.60 (x2.0)	1.08 (x1.35)

Table 3.4: *Displayed-frame and reference-frame field-of-view statistics. Angles are measured in degrees, solid angles are measured in steradians. The ratio between reference-frame and displayed-frame values is given in parenthesis for some measurements.*

Discussion

The need to enlarge reference-frames to cope with the possibility of field-of-view clipping places constraints on the types of systems in which post-rendering warping is most useful. The reason is that rendering oversized reference frames is expensive, so the net benefit from using post-rendering warping depends strongly on how much the reference frames are enlarged relative to the displayed frames.

The ideal conditions for minimizing this enlargement are:

1. A high reference-frame rate.
2. A high ratio between the displayed-frame rate and the reference-frame rate.
3. Small head rotation rates. In particular, a low maximum head rotation rate.
4. Accurate prediction of future head orientation.
5. A large displayed-frame FOV, so that a particular Δf is small in proportion to f.

Most of these conditions are self-explanatory given the discussion earlier in the chapter, but I will elaborate on #1 and #3.

As the reference-frame rate increases, the extra FOV required for reference frames decreases. Thus, a post-rendering warping system generating 60 Hz displayed frames from 10 Hz reference frames is more efficient than a system generating 12 Hz displayed frames from 2 Hz reference frames. The 60 Hz/10 Hz system will also produce fewer occlusion artifacts. Both of these advantages are manifestations of the fact that post-rendering warping is fundamentally a technique that exploits temporal coherence. Temporal coherence is greatest at high frame rates.

The disadvantage of lowering the reference-frame rate eventually ends. If the reference frame is low enough, and/or the expected maximum head rotation rate is high enough, then it becomes necessary to represent a full $360°$ (more precisely 4π steradian) view for each reference-frame viewpoint. For example, such a representation becomes necessary for a maximum rotation rate of $100°$/sec and a reference frame rate of 0.7 Hz. Typically the representation consists of six sides of a cube (a complete environment map [Blinn76, Greene86]), since the cube faces can be generated by a conventional rendering engine. Once we reach this point, further decreases in the reference frame rate do not result in an increase in the reference-frame FOV. In fact, if the displayed-frame rate is held constant, then further decreases in the reference-frame rate actually improve the overall efficiency of the system. The reason for this improvement is the increased ratio between displayed-frame rate and reference-frame rate. Note however, that the quality of the displayed frames will slowly degrade due to increased visibility artifacts—the viewpoint-on-line condition fails more frequently and more severely as the reference-frame rate decreases.

Head rotation rates also strongly affect the net efficiency improvement from a post-rendering warping system. Studies of head rotation rates published in the literature provide some guidance as to the range of head rotation rates that are possible. Maximum possible human head rotation rates, especially about the yaw axis, are quite high. Foxlin [Foxlin93] found peak rates of 1000°/sec yaw, 500°/sec pitch, and 575°/sec roll. Zangemeister *et al.* [Zangemeister81] found a peak yaw rate of 580°/sec when subjects were changing their gaze (using head rotation) from one point to another. However, this peak rate was only attained for a very brief period of time, at the midpoint of the head movement.

From a theoretical point of view, we expect that the maximum rate of head rotation will be only briefly attained. Consider a head movement from one at-rest orientation to another at-rest orientation. Assume that the acceleration used in the movement is always $\pm a$, where a is a constant ([Zangemeister81]'s data indicates that this assumption is not unreasonable for high-speed rotations). Then, the average rate of head rotation (over the whole movement) will be exactly one half of the maximum rate of head rotation, which is reached in the middle of the movement.

Zangemeister found that the average speed during a worst-case 120° head rotation was approximately 300°/sec. This speed is almost exactly one half of the peak rate. The period over which this average was taken was 400 msec. In a five reference-frames/sec post-rendering warping system, this corresponds to the time period $2Q$ (twice the reference-frame interval). Thus, for this system, the worst case value of R (average rotation rate over the time period $2Q$) is only 300°/sec even though the very-short term peak rate is 580°/sec.

Such high rates of head rotation (values of R) would be difficult to support in a post-rendering warping system that uses a single image at each reference-frame viewpoint. Fortunately, users do not rotate their heads this quickly in most applications. Azuma studied head motion by several naive users in a head-mounted display demo [Azuma95a], and found that the maximum angular head velocity was 120°/sec. In one of his two sets of data, the maximum velocity was considerably less, 70°/sec. Holloway studied a surgeon's head movement during a surgical planning session [Holloway95], and found that head rotation velocities were generally below 50°/sec, but occasionally reached as high as 100°/sec. As shown earlier in Table 3.3, the maximum head rotation rate in the kitchen walkthrough data that I gathered was 70°/sec.

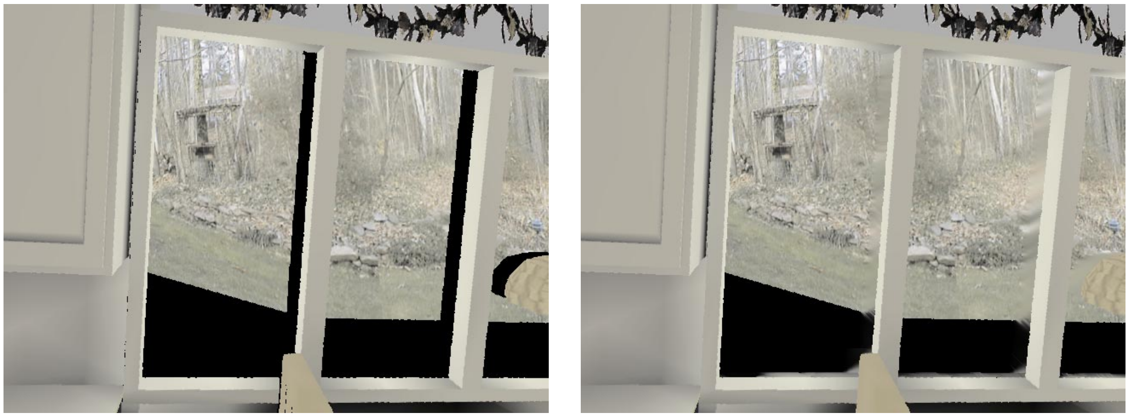
Thus, I conclude that for many applications field-of-view clipping could be entirely eliminated by supporting a yaw rate of 100–120°/sec. Table 3.3 and some of the previous research indicate that supporting a lower rate for pitch, perhaps 70–80°/sec, would be sufficient. For applications in which occasionally brief clipping of the FOV is acceptable, the system could be designed for much lower rotation rates, perhaps 45°/sec. Ultimately, the decision as to the maximum head rotation rate to support is an application-dependent one, unless the system designer decides to support the maximum possible human head rotation rates of around 600°/sec in each axis. For some applications, it may be possible to simply inform the user that rapid head rotation will result in clipped images (the “don’t do that” approach).

When a user is rotating his head quickly, the importance of high-resolution imagery is reduced. This observation leads to the possibility of an adaptive strategy in generating reference frames. Rather than generating reference frames with a constant FOV, the field of view could be adjusted to accommodate the user’s predicted head rotation rate. The reference-frame size (in pixels) would be held constant, even as the FOV changes. Thus, the *angular* resolution of the reference frame would decrease as the FOV increased.

I have not explored this adaptive approach in detail. Its feasibility depends in large part on whether or not future high rotation *rates* can be accurately predicted. This technique would increase reference-frame rendering speed if the renderer is fill-rate limited, but not if it is transform-rate limited. The technique could introduce difficulties in the 3D warp reconstruction process if the reconstruction process relies on a fixed angular resolution of the reference frames.

3.2 Hole filling

The first section of this chapter discussed strategies for choosing reference frame viewpoints. But even with well chosen reference frames, the visibility problem is usually not completely solved. After warping the reference frames, there are typically a few visibility holes in the displayed image.[4] These holes represent regions of the scene for which there is no information available in the reference frames. The post-rendering warping system must choose a color with which to fill each of the hole pixels. Ideally, the system would use its conventional renderer to selectively render these hole pixels. This approach works well if the conventional renderer is a ray tracer, and has been successfully employed in systems that accelerate ray-tracing of animations [Badt88, Adelson95, Ward90], as mentioned in Chapter 2. However, selectively rendering hole pixels is hopelessly inefficient with a polygon-based conventional renderer. We need a technique that fills hole pixels using the information available in the reference images. Figure 3.10 illustrates the technique that I developed.



Holes filled with the color black

Holes filled with algorithm described in this section

Figure 3.10: *Good-quality hole filling is important. On the left is a frame in which holes are filled with the color black. On the right is the same frame in which the holes are filled with the algorithm described in this section. Later in this section, Figure 3.20 provides a more detailed comparison of hole-filling algorithms.*

[4] Not all visibility errors necessarily manifest themselves as holes. Consider a background surface that falls in the same portion of the displayed frame as a missing portion of a medium-depth object. The displayed frame will be incorrect, yet there is no hole (there actually is a hole in the medium-depth object, but it is incorrectly filled by the background). McMillan refers to this type of error as an *exposure error* [McMillan97]. This type of error requires a complex interaction of two occluders and a background surface at different depths. First, the missing portion of the medium-depth object must be hidden by a foreground object in the reference frame. Second, the troublesome portion of the background surface must appear on one side of the foreground object in the reference frame, but appear on the *opposite* side of the foreground object in the displayed frame. Fortunately, the small viewpoint changes in PRW make this type of violation rare.

In developing my technique for filling holes, I had three goals:

1. Fill the holes with a good estimate of the correct color (i.e. the color that would appear there in a conventionally rendered image)

2. Avoid attracting the user's attention to the holes—Fill holes in a manner that minimizes their perceiveability.

3. Fill the holes in a computationally efficient manner.

I will begin by discussing my approach to the first goal, then discuss the second. Discussions related to the third goal, efficiency, are intermingled with the discussions of the first two.

3.2.1 Estimating the correct hole-fill color

Holes form at the boundary between a foreground object and a background object (Figure 3.11). Since the reference frames do not provide any information about the portion of the scene covered by the hole, a warping system must make some kind of assumption about the scene in order to fill the hole. My system assumes that the background object extends into the hole, and estimates the hole color accordingly. Typically, the background object does in fact extend into the hole, and thus this approach produces reasonable results.

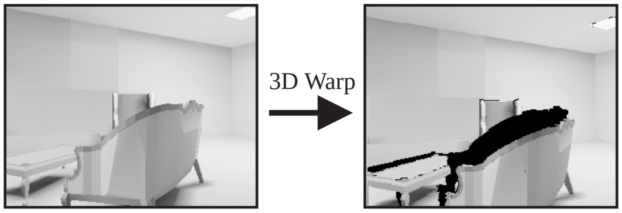


Figure 3.11: *Visibility holes left by a 3D warp (shown here in black) are always located at the boundary between a foreground object and the object(s) behind it. In this case, holes form at some of the boundaries of the sofa and table. Note that holes do not appear at all foreground/background boundaries.*

The simplest version of this approach would just copy the background-object color to each of the hole pixels. But, how does the system decide which pixels bordering the hole belong to the background object? The system could examine all of the pixels bordering the hole, and pick the one that is furthest away to determine the hole-fill color. Ward used a variant of this approach in his *pinterp* program [Ward90]. Picking the hole-fill color in this manner presents several problems. First, although the technique is conceptually simple, in practice it is very complicated and inefficient to implement in a manner that meets the goal of having the cost of the hole-fill operation grow linearly with the number of hole pixels. Second, this technique chooses only a single background pixel to determine the fill color for the entire hole. If the background object is textured, more than one background-pixel color should be used to fill the hole. I will return to this point later.

To overcome these problems, I have developed a technique for hole filling that is based on the epipolar geometry of the 3D warp. I will begin by describing how this technique is used to fill the holes left after warping a single reference frame. Later, I will describe how the technique is extended to warping two or more reference frames.

Hole filling based on epipolar geometry

The key realization behind the development of my hole-filling technique is the following: The 3D warp's epipolar geometry indicates which pixels at the edge of a hole are background pixels. Figure 3.12 illustrates this point. The figure shows the epipolar geometry in the vicinity of a hole created by the movement of a convex foreground object. For any particular hole pixel, we can trace in the reverse direction along the epipolar lines to find a background-object pixel. It is no longer necessary to compare the depths of different pixels at the edge of the hole to determine which ones are background pixels.

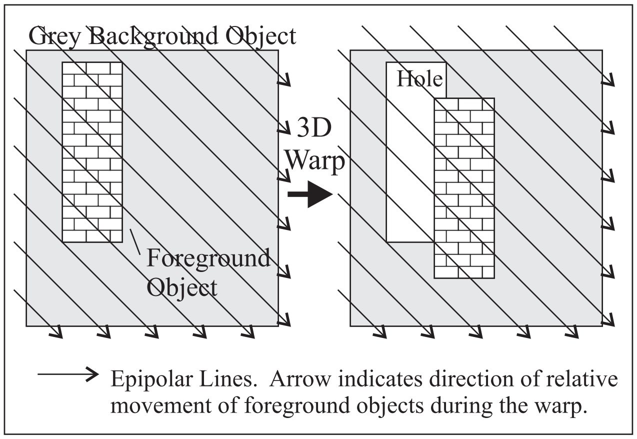

Grey Background Object
Hole
3D
Warp
Foreground
Object
Epipolar Lines. Arrow indicates direction of relative
movement of foreground objects during the warp.

Figure 3.12: *Epipolar geometry near a visibility hole left by a convex foreground object. The 3D warp causes the foreground object to move relative to the background object. For any particular pixel in the hole, the edge of the background object can be found by moving along the epipolar lines in the direction opposite to object movement. Note that although the epipolar lines in this figure and several subsequent ones are mutually parallel, in general the epipolar lines will be converging or diverging slightly.*

The simplest hole-filling algorithm that made use of the epipolar geometry of the 3D warp would be the following:

```
Warp source image, to create destination image
for each pixel in destination image {
  if the pixel is a hole pixel then {
    Search backward along epipolar line to find background pixel
    Copy background pixel color to hole pixel
  }
}
```

Unfortunately, the cost of this algorithm grows not just with the number of hole pixels, but also with the average hole size. The problem is the search step in the inner loop. This search step costs more as the average hole size grows. To address this problem, I have developed a single-pass hole-filling algorithm that does not require the search step. The algorithm traverses the destination image in an order that allows background pixel colors to be “wiped” across each hole.

Figure 3.13 illustrates this traversal order. For each pixel of this traversal that is a hole pixel, the algorithm looks backward by one pixel along the epipolar line to determine the hole pixel’s color. Because this reverse-direction pixel has already been touched by the algorithm, it is either a true background pixel, or an already-filled hole pixel. In either case, this reverse-direction pixel contains the background-object color that the algorithm needs to fill the current hole pixel. Figure 3.14 illustrates the gradual filling of a hole by this algorithm.

The traversal depicted in Figure 3.13 is actually a variant of McMillan and Bishop’s occlusion-compatible rendering order [McMillan95a]. McMillan and Bishop’s rendering order moves inward towards the epipole or outward from the epipole. However, their traversal is in the source image, while ours is in the destination image.

An important feature of McMillan and Bishop’s implementation of occlusion-compatible rendering is the division of the source image into up to four *sheets*. Each sheet is traversed in a raster-scan order, while still maintaining the occlusion-compatible property for the entire image. I extend this idea by dividing the image into as many as eight sheet-like regions, rather than just four. By using eight regions rather than four, our traversal direction remains closer to the ideal direction (perpendicular to the epipolar lines). I explain the importance of this improvement later in this chapter. Figure 3.15 illustrates the eight-sheet traversal.

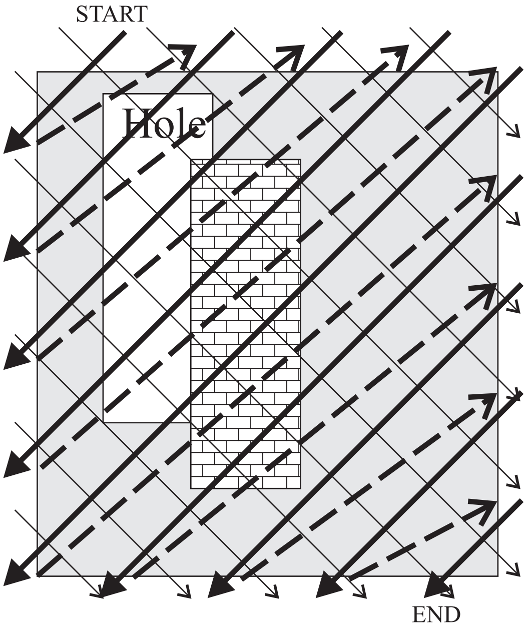


Figure 3.13: *Image traversal for hole filling. To fill the holes, the destination image is traversed in the order indicated by the dark black lines. The background-object color is thus propagated into the hole, one pixel at a time. The result is equivalent to that which would be obtained, in a continuous-domain image, by traversing each epipolar line (thin lines) and copying the background color. The traversal which is depicted here has fewer problems for a discrete-domain image. Note that in this particular image, the epipole is located at infinity, and thus the epipolar lines are parallel. In general, the epipolar lines are not perfectly parallel.*

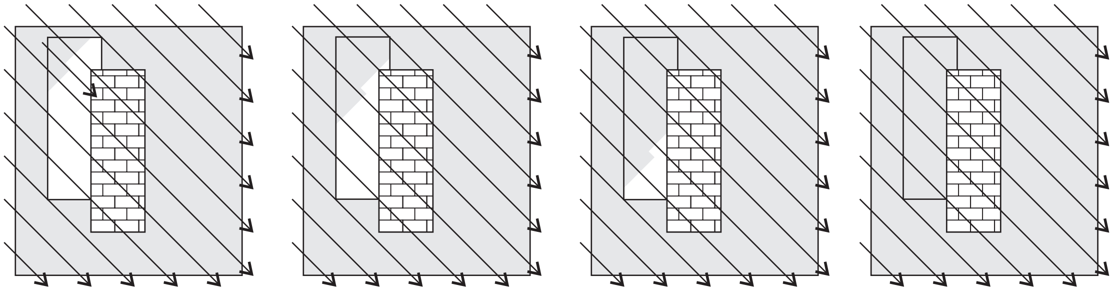


Figure 3.14: *The hole-fill algorithm gradually fills the hole by "wiping" across it. These images illustrate the process of gradual filling.*

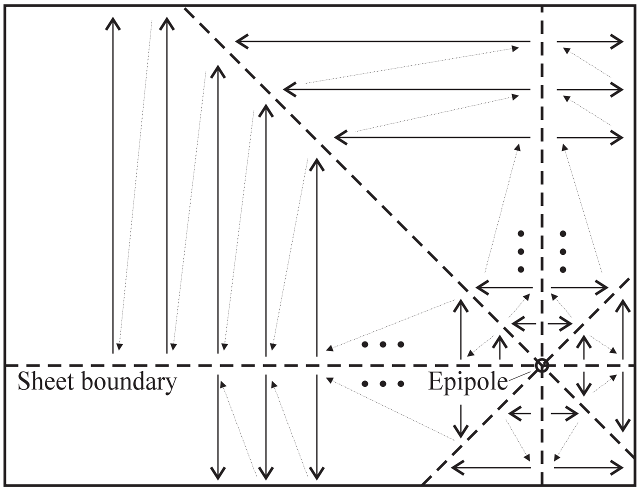


Figure 3.15: *Eight-sheet occlusion-compatible image traversal. The image is divided into up to eight sheets. Each sheet is traversed in a raster-like order to perform hole filling. By using eight sheets, we can guarantee that the local traversal direction is always within* $45°$ *of the ideal direction specified by the epipolar geometry.*

Filling holes left by concave objects

The hole-filling algorithm that I have just described fails unnecessarily when a hole is created by a concave foreground object. Figure 3.16 illustrates this case, which occurs in regions of the hole near concavities in the foreground object. When the filling algorithm looks in the reverse direction along the epipolar lines, it picks up the color of the concave foreground object rather than the color of the background object.

A minor modification of the hole-filling algorithm cures this problem. In order to describe the modification, I need to first define the *forward edge* of an object. The forward edge(s) of a foreground object are determined in the source image, with respect to the epipolar geometry of a particular 3D warp. Consider a source-image pixel X, and the pixel Y that is adjacent to it in the direction of my epipolar arrows. More precisely, the “direction of my epipolar arrows” is the direction of the [positive/negative] epipole in the source image. Pixel X is part of a forward edge if pixel Y is at a greater depth than X. Figure 3.16 shows the forward edges as thick lines.

In order to properly handle concave foreground objects, the hole-fill algorithm and the 3D warp itself are modified to treat pixels belonging to a forward edge specially. As each pixel X is warped, the warp algorithm checks to see if it is a forward-edge pixel. If X is a forward-edge pixel (and X passes the Z-buffer test in the destination image), then the warp algorithm sets a flag bit at X’s location in the destination image. Next, the algorithm determines the color, C, of the background-pixel Y (defined

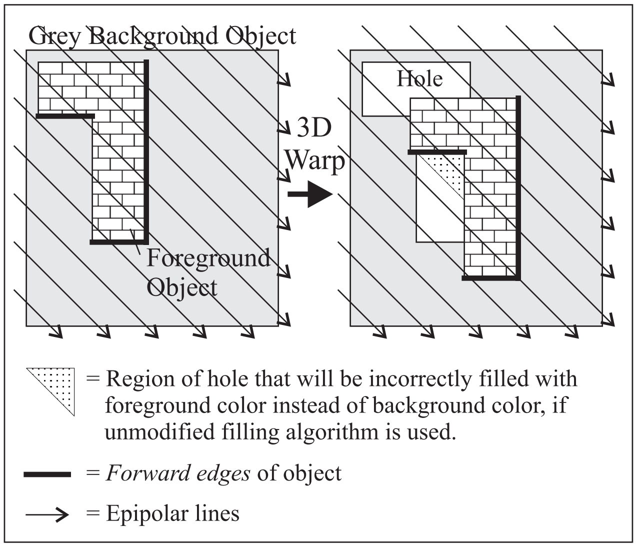

Grey Background Object
Hole
3D
Warp
Foreground
Object
= Region of hole that will be incorrectly filled with foreground color instead of background color, if unmodified filling algorithm is used.
= *Forward edges* of object
= Epipolar lines

Figure 3.16: *The simple version of the hole-filling algorithm will incorrectly fill some areas when the foreground object is concave. The dotted region on the right side of this figure is an example of such an instance. By recognizing and specially treating the forward edges of an object, we avoid this problem.*

earlier), which is adjacent to X in the source image. This color C is stored at X's location in the destination image, but in an auxiliary *hole-fill color buffer* rather than the main destination-image color buffer. X's location in the main destination-image color buffer is set to X's color, as usual.

As discussed earlier, the hole fill algorithm copies background colors into hole pixels as it traverses the destination image. The hole-filling algorithm is modified to use the information from the hole-fill color buffer as follows: When copying a color from a pixel tagged as a forward-edge pixel, the copying uses that pixel's hole-fill color rather than its main color. Thus, the hole is filled with the background color that was adjacent to the forward edge, rather than the foreground color from the forward edge. This modification to the algorithm eliminates the incorrect results that the unmodified algorithm produces near the concavities of foreground objects.

I believe that it would be possible to eliminate the auxiliary hole-fill color buffer by changing the algorithm slightly. Instead of storing the color C with the forward-edge pixel (in the hole-fill buffer created for this purpose), the color C could be stored with a nearby hole pixel, in the main color buffer. The nearby hole pixel chosen for this storage is the one closest to the warped location of the forward edge. Since the main color buffer is otherwise unused at a hole pixel, this change increases the storage efficiency of the algorithm. I have not attempted to implement this strategy.

One might suspect that the forward-edge modification to the hole-filling algorithm is a hack that just re-shuffles the conditions under which artifacts occur. Artifacts might be eliminated under one set of conditions, but be introduced under a different set of conditions. However, this suspicion is wrong — There is no general case under which the original color of a forward-edge pixel ought to be used to fill a hole. The relative motion of surfaces as determined by the epipolar geometry guarantees that there is no such case. Thus, the results of the hole-fill algorithm can only be improved by using the color from the hole-fill buffer rather than from the main color buffer.

Implementing the forward-edge modification to the hole-filling algorithm is tricky, due to the discrete nature of the reference and displayed frames. In particular, when the 3D warp detects that a reference-frame pixel is on a forward edge, it has to be concerned with the behavior of the reconstruction algorithm. It can not allow the forward edge to disappear in the destination image as a result of over-sampling in the reference frame.

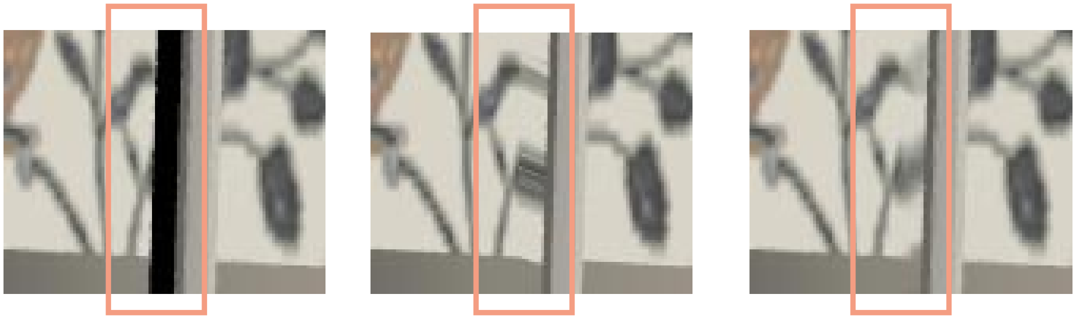
3.2.2 Minimizing the perceptual impact of holes

My hole-filling technique uses only the information from the reference frames when filling holes. As a result, it, or any similar technique, can only estimate the correct hole-fill colors. When an estimate is incorrect, this technique fills hole pixels with incorrect colors. Because hole pixels may be the wrong color, it is undesirable to attract the user's attention to these pixels. For this reason, I have designed my hole-filling technique to attempt to avoid attracting the user's attention to hole pixels.

How is a user's attention drawn to an area of an image? One of the most important capabilities of the lower levels of the human visual system is the capability to detect contrast between nearby areas of the visual field. A user's attention is likely to be drawn to such areas of high contrast. Thus, to minimize the perceptual impact of holes, the hole-filling algorithm should avoid unnecessarily introducing regions of high contrast within a hole.

Figure 3.17 illustrates the difference between the simplest variant of my hole-filling algorithm and a variant that attempts to minimize contrast. The simplest variant of the algorithm introduces high-contrast "stripe" artifacts that attract attention to the hole. The low-contrast variant eliminates most of these artifacts.

The high-contrast stripes are caused by propagating the colors from the background object along epipolar lines. If the colors along the edge between the background object and the hole vary, then the variations become stripes within the hole. The low-contrast variant of my algorithm reduces contrast by blurring the colors used to fill holes. Rather than copying a single pixel to fill each hole pixel, the



No hole filling

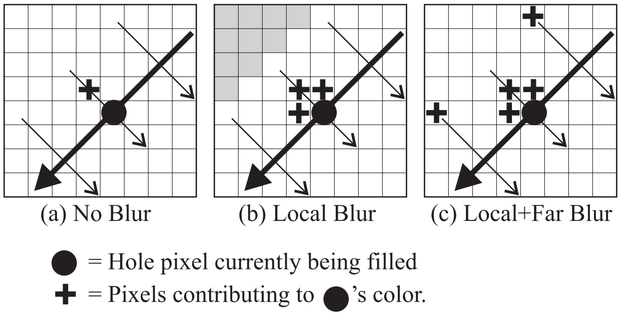
Hole filling without blur

Hole filling with blur

Figure 3.17: *Hole filling with and without blurring. The left image shows the unfilled hole. The hole is the black area inside the pink box. The middle image shows the the results of my epipolar-geometry–based hole-filling algorithm, but without blurring. The right image shows the results of my algorithm with blurring. The blurring reduces the perceiveability of the hole.*

algorithm averages several pixels to compute each new hole pixel. Figure 3.18 illustrates the blurring algorithm, and compares it to the non-blurring algorithm.

If a hole pixel is close to a background surface, then the hole-filling algorithm averages three nearby pixels to compute the fill color (Figure 3.18b). The blurring caused by averaging three close pixels levels off as the image-space distance from the true background surface increases. So, when the distance from the true surface becomes greater than four pixels, two additional further-away pixels are used in the averaging, to broaden the spatial extent of the blurring (Figure 3.18c). This algorithm could be generalized to use pixels that are even further away as the distance from the true surface increases further, but I did not find this generalization to be necessary. Once holes get too big, they tend to attract attention no matter what the filling algorithm does.



(a) No Blur

(b) Local Blur

(c) Local+Far Blur

● = Hole pixel currently being filled
✚ = Pixels contributing to ●'s color.

Figure 3.18: *Blurring technique for hole filling. In (a), only a single already-filled pixel is examined to determine the hole-fill color for a new pixel. We do not use this approach, because it causes stripe-like artifacts. If our hole-filling algorithm is filling a pixel near a known background surface (b), then it averages three adjacent already-filled pixels to calculate the newly-filled pixel's color. If the known background surface is far away (c), then the algorithm averages five already-filled pixels, two of which are four pixels away.*

The blurring technique that I use is not very sophisticated, but it is inexpensive and succeeds in greatly reducing the stripe effect, as Figure 3.17 showed. I believe that a more sophisticated blurring algorithm would be considerably more expensive, but would not substantially improve the visual quality of the output.

The blurring technique imposes some additional demands on the filling algorithm's traversal of the image. Without blurring, the traversal order must insure the following: When a hole pixel is filled, the pixel in the reverse direction along the epipolar line must have already been filled. Figure 3.18a illustrated this case—The pixel marked with a $+$ must have been filled before the pixel marked with a solid dot.

With blurring, the traversal order must insure that, for any particular hole pixel, either three (Figure 3.18b) or five (Figure 3.18c) *precursor* pixels have already been filled. Since the direction of epipolar lines generally varies throughout the image, the direction in which the precursor pixels lie relative to the hole pixel varies also. My algorithm quantizes precursor-pixel directions to one of the eight cardinal/diagonal directions. Figure 3.19 shows how these directions vary throughout the image.

A four-sheet occlusion-compatible traversal is not sufficient to insure that precursor pixels will be ready when needed. Under certain conditions, a hole pixel will be filled before all of its precursor pixels have been filled. However, the eight-sheet traversal is sufficient to avoid this problem (except at sheet boundaries, which I will discuss later). It is for this reason that I use the eight-sheet traversal. The eight-sheet traversal performs better that the four-sheet traversal because the maximum divergence between the actual traversal direction and the ideal traversal direction (as defined by the epipolar lines) is less. The maximum divergence for the eight-sheet traversal is $45°$, while the maximum divergence for the four-sheet traversal is $90°$.

The choice of precursor pixels depicted in Figures 3.18b and 3.18c was influenced in part by the eight-sheet traversal order. One cannot arbitrarily change the relative location of these precursor pixels and expect that the one-pass eight-sheet traversal order will guarantee that the precursor pixels are always ready when needed. More broadly, it is tricky to specify a one-pass hole-filling algorithm that uses background colors for filling and that incorporates blurring.

Even the eight-sheet traversal order does not work perfectly at the boundary between sheets, if the sheets are traversed one at a time. At a sheet boundary, it is possible for one of the three (or two of the five) precursor pixels to still be unfilled when it is needed. My current implementation of the hole-filling algorithm detects this case and ignores the offending precursor pixel(s). However, this

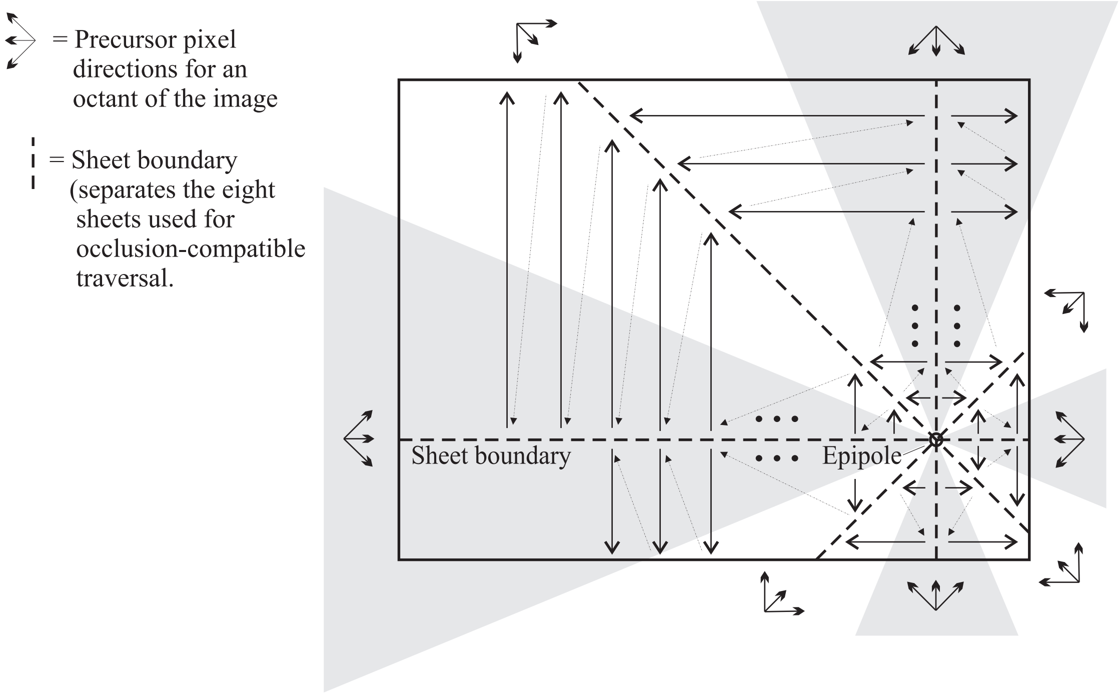

= Precursor pixel directions for an octant of the image
= Sheet boundary (separates the eight sheets used for occlusion-compatible traversal.
Sheet boundary
Epipole

Figure 3.19: *Variation of precursor-pixel directions throughout the image. The precursor-pixel directions are quantized to one of the eight cardinal/diagonal directions. Filling a hole pixel requires looking in up to three directions,* $45°$ *apart. Due to the direction-quantization, there are eight possible sets of these three directions. Each set is used in one of eight regions in the image. The eight regions converge on the epipole. The figure shows four of the eight regions in grey. The other four regions are represented by the intervening white areas. Note that the eight precursor-pixel regions do not correspond to the eight regions used for the eight-sheet occlusion-compatible traversal.*

means that blurring only occurs across sheet boundaries in one direction, so that a sheet boundary which passes through a large hole could conceivably be visible in the displayed image. I have not observed this artifact in practice.

An alternate implementation is possible, in which the eight sheets are traversed approximately simultaneously, by visiting them in round-robin fashion. At each visit to a sheet, one row/column of the sheet is traversed. This traversal gradually zeroes in on (or away from) the epipole from all directions simultaneously. By using this traversal, precursor pixels will always be ready when needed, even at sheet boundaries.

3.2.3 Hole filling for multiple reference frames

The hole-filling algorithm that I have just finished describing works only for warping a single reference frame. Why can't the algorithm be used to fill the holes left after warping two or more reference images? The problem is that the algorithm relies on the epipolar geometry of the 3D warp. This epipolar geometry is different for each reference frame, even for a single destination image. Once two reference frames have been warped into a single destination image, there are two different sets of epipolar lines. At any particular hole pixel, there is no longer a single direction in which the hole-filling algorithm can look to find a background-object pixel.

The solution to this problem is to independently warp each reference frame into its own destination image. The hole-filling algorithm is run separately on each destination image. Finally, the destination images are composited to form the displayed frame. The compositing process always discards hole-filled pixels in favor of non-hole-filled pixels. Thus, hole-filled pixels only appear in the displayed frame at locations where there were hole-fill pixels in *all* of the individual destination images. These locations correspond to parts of the scene that were not visible in *any* of the reference frames (i.e. true displayed-frame hole pixels).

If our hole-fill algorithm always guessed hole-pixel colors correctly, then the different destination images would always agree on the color to be stored at a particular displayed-frame hole pixel. But, the hole-fill algorithm is not perfect, and so the different destination images can disagree as to the correct fill color. The compositor determines the hole-fill color that will be displayed by averaging the colors contributed by the different destination images. Each contribution is weighted by the reciprocal of that hole-pixel's distance (along its epipolar line) from a background pixel. This weighting strategy is motivated by the fact that the closer a hole-fill pixel is to a background pixel, the more likely it is to have correctly estimated the "true" color for the hole.

This compositing algorithm requires that each destination image contain two extra fields at each pixel. The first field is a bit which indicates whether or not the pixel is a hole pixel. If the pixel is a hole pixel, a second field contains the distance of the hole pixel from the nearest background pixel along its epipolar line. The hole-fill algorithm computes these values as it traverses the destination image.

In practice, the multi-reference-image hole-filling algorithm does not have to be implemented by warping each source image to its own destination image. Instead, the entire process can be performed incrementally using just a single destination image (the displayed frame) that contains some additional per-pixel variables. In this incremental approach, the compositing and hole-filling steps for each reference frame are combined into a single hole-fill/compositing pass. Conceptually however, the algorithm is unchanged. The per-pixel variables required for the incremental approach are described in Appendix C.

3.2.4 Alternate approaches and discussion

The technique that I have developed for hole filling is not the only possible one. In this section, I discuss a variety of alternative approaches to the hole-filling problem. I quantitatively compare some of these approaches to my technique.

Human visual system's approach

One of my committee members, Dr. Fred Brooks, suggested that one can draw an analogy between the problem of hole filling and the human brain's need to cope with the blind spot in the eye. Humans are not generally conscious of the existence of the blind spot, because the lower levels of the human visual system fill it in. It is instructive to consider how this process works. Careful experiments have shown that the visual system will fill in solid colors, 2D textures, and straight lines when they pass through the blind spot [Ramachandran95].

If this same approach could be applied to the 3D warp hole-filling problem, it would reduce or eliminate the need to explicitly determine background colors. The reason is that the visual system's approach would automatically preserve the edge between a foreground and background object (a line of sorts). In order to preserve this edge, the hole filling would have to use the color that is in fact the background color. To explicitly preserve edges or lines, they must first be detected. Parts of the human visual system are devoted to the task of detecting oriented line segments. This approach is intriguing, but does not appear to be immediately feasible for a low-cost PRW system.

3.2.5 Filling with texture

My algorithm fills holes with blurred background colors, but does not attempt to reproduce the background texture. [Malzbender93] shows that holes in textured regions can be realistically filled by analyzing and reproducing the second order statistics of the texture. This type of approach is currently too expensive for a real-time warping system.

Combining reconstruction with hole filling

My approach to hole filling is clearly separated from the reconstruction process (discussed in the next chapter). It is possible to combine the two tasks, but generally with a penalty in either efficiency or image quality. I initially used a combined approach, as described in [Mark97b] and discussed briefly in the next chapter, but discarded it in favor of the approach described in this chapter. The cost of my combined approach was not bounded by the number of hole pixels, and the approach did not allow for blurring to eliminate “stripe” artifacts.

Chen and Williams [Chen93] used a post-processing approach to reconstruction which had the side effect of filling holes. Their approach does not distinguish between foreground and background objects. They average pixels on the border of a hole in order to determine the fill color. Their paper does not make it clear how the border pixels are selected.

Quantitative comparisons

I have quantitatively compared the displayed frames generated by my hole-filling technique and by several alternative techniques. To measure the quality of a technique, I calculate the average and root-mean-squared errors between pixels generated using the technique and the same pixels generated using conventional rendering. Note that these quality metrics are quite crude, because they do not accurately measure differences as perceived by the human visual system.

I compare four different hole-filling techniques. With the exception of the first technique, they are implemented in a single pass after both reference images have been warped. The four techniques are:

1. The epipolar-geometry–based technique described earlier in this chapter. This technique requires a separate pass after warping each reference image.

2. A technique that searches locally for background surfaces and uses the color of the furthest-away background surface to fill a hole. This technique is described below in more detail.

3. A technique that fills hole pixels by searching to the left along the current scan-line for the first non-hole pixel. This technique could be efficiently implemented at scan-out time without any searching by keeping track of the color of the last non-hole pixel. This color would be used to fill any hole pixels.

4. A “do-nothing” approach – holes are filled with the color black.

The technique that searches locally for background surfaces requires some more explanation. In this approach, the hole-filling algorithm examines every pixel in the destination image. If the pixel is a hole pixel, the algorithm begins a search for the farthest-away (in Z) surface that borders the hole. The algorithm constrains its search to the eight cardinal/diagonal image-space directions. For each of these directions, the algorithm searches outward from the hole pixel until it finds a non-hole pixel, or has reached a maximum distance from the hole pixel. This maximum distance is 20 pixels in my implementation. The eight cardinal/diagonal directions thus yield up to eight non-hole pixels that border the hole. The algorithm chooses the pixel that is farthest away in Z, and uses its color to fill the hole. The assumption is that this color represents the local background surface, and thus is the correct color with which to fill the hole. This algorithm is similar to that used by [Ward90]. Note that the cost of this algorithm grows with hole size.

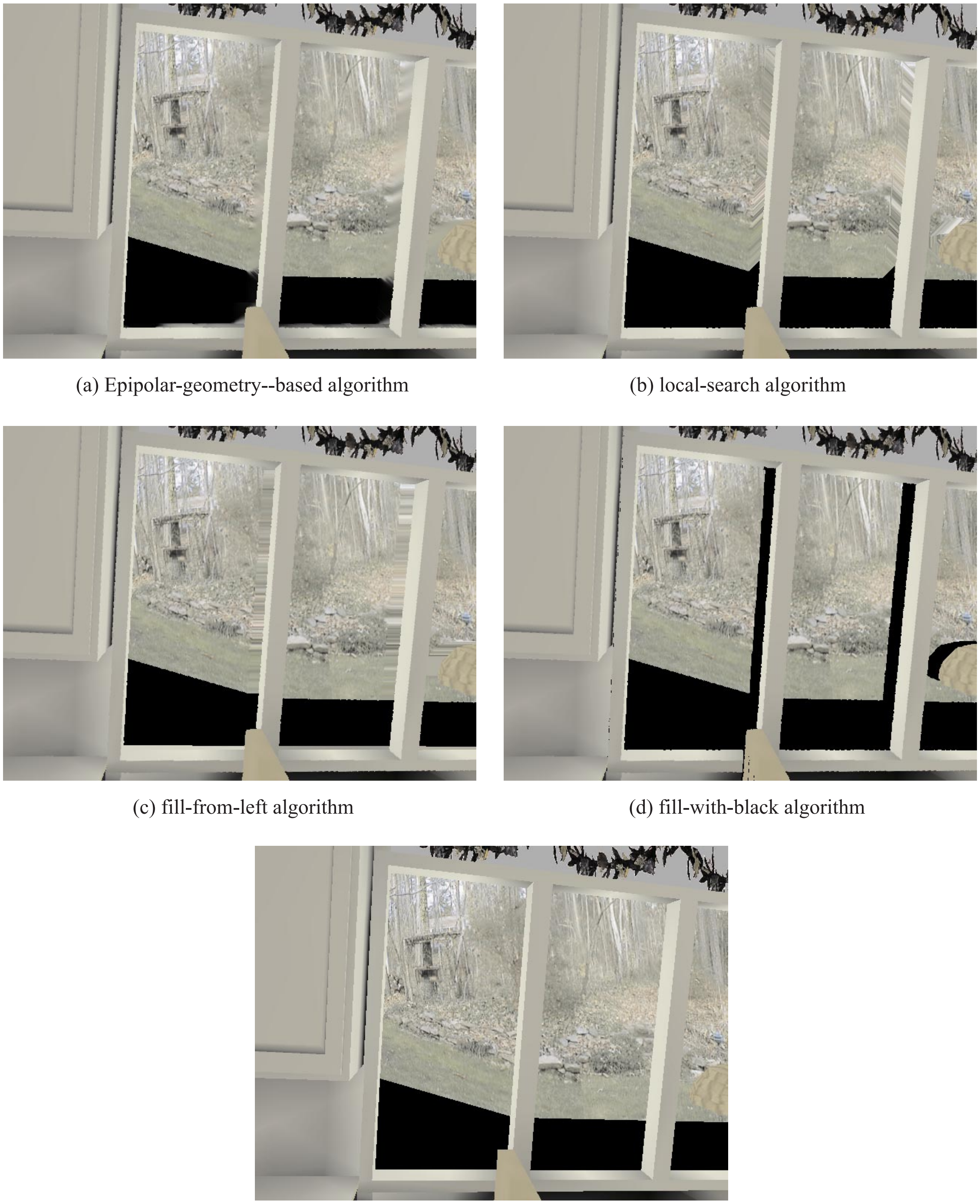
Table 3.5 provides the results of my quantitative comparison of the different hole-filling algorithms. I made the comparison for both the entire kitchen sequence and for a single frame of this sequence. The data show that the epipolar algorithm and the local-search algorithm are clearly better than the others, as expected. However, the comparison does not indicate any clear difference between the epipolar and local-search algorithms.

	Entire Sequence		Single Frame (#430)	
	Avg. Error	RMS Error	Avg. Error	RMS Error
Epipolar geometry	1.55	6.65	3.44	11.51
Local search	1.54	6.72	3.17	10.97
Copy from left	1.72	8.30	4.00	15.94
Fill with black	1.86	9.59	6.48	24.94

Table 3.5: *Comparison of different hole-filling algorithms. The average and RMS per-pixel, per-color-component error is computed for the entire Kitchen sequence, and for frame #430. Frame #430 has several particularly large holes. Note that even a perfect hole-filling algorithm would not produce zero error in this table, because these error measurements also include resampling error from non-hole regions of the displayed frames.*

Figure 3.20 shows that the results produced by the epipolar-based algorithm are perceptually better than those produced by the local search algorithm (for frame #430). The average and RMS per-pixel error metrics are not sufficiently tied to human perceptual characteristics to accurately measure this difference. In particular, the blurring provided by the epipolar-based algorithm improves the visual quality of the results, but is not properly measured by the simple error metrics.

One might ask, “Can blurring be added to the local-search algorithm?” Such a modified algorithm would produce images comparable in quality to those produced by my epipolar-based algorithm, yet not require a fill pass after warping each reference image. I do not believe that such a



(a) Epipolar-geometry--based algorithm

(b) local-search algorithm

(c) fill-from-left algorithm

(d) fill-with-black algorithm

(e) conventional rendering

Figure 3.20: *Comparison of different hole-filling algorithms, for kitchen frame #430. This frame has particularly large holes (caused by large prediction error). The epipolar-geometry–based hole-filling algorithm produces the least noticeable artifacts, as is typical. The fill-from-left algorithm produces results similar to the local-search algorithm. This result is atypical, and is caused by the particular configuration of geometry in this frame. Typically, these two algorithms produce quite different results—the fill-from-left algorithm is generally inferior to the local search algorithm.*

change can be made without a substantial penalty in computational expense. The epipolar algorithm's blurring can be performed in a single pass (per reference frame) because it takes advantage of the constraints on foreground and background object placement imposed by the epipolar geometry of the 3D warp. Any algorithm that operates in the final image, after all reference frames have been warped, can not take advantage of these geometric constraints. A more general, and more expensive, blurring operation would be required.

3.2.6 Discussion

The human-perceived quality of the output from the hole-filling algorithm that I have developed is better than the quality of the output produced by the competing algorithms that I have examined. By taking advantage of the epipolar geometry of the 3D warp, the algorithm executes in a single pass over a warped image, with cost essentially independent of hole size. Its major drawback is that the algorithm must be executed once for each reference frame, rather than just a single time after all reference frames have been warped. In a two reference-frame system, the need to make two passes over the displayed frame to fill holes contributes significantly to the cost of PRW. In some types of PRW systems, it may make sense to use a cheaper hole-filling algorithm even though it produces lower-quality results.

3.3 Relationship of hole size to epipolar geometry

Our current post-rendering warping system always warps two reference frames to produce a derived frame. In order to minimize occlusion artifacts, the relative positions of the reference-image centers of projection and the derived image center of projection must satisfy the point-on-line condition.

The point-on-line condition (discussed earlier in this chapter) requires that the center of projection for the derived image lie on the 3-space line segment between the centers of projection for the reference frames.

If this condition is maintained, we can guarantee that there will be no occlusion artifacts for a single convex occluder. Multiple convex occluders will not produce artifacts so long as the multiple occluders remain independent of each other (relatively far away in image space).

In practice, the point-on-line condition is usually not perfectly met. This section mathematically characterizes this deviation from the ideal condition, and shows how it affects hole size.

This section has the following structure:

1. I show that the point-on-line condition is equivalent to coincidence of the destination-image epipoles generated by the two reference-image viewpoints.

2. If the destination-image center-of-projection is perturbed away from the point-on-line condition, the previously coincident epipoles begin to diverge. I derive expressions for the locations of the diverging epipoles. I also derive a small-perturbation approximation for the locations of the diverging epipoles.

3. Next, I work with the expression for the translational movement of arbitrary warped points (taken from the warp equation). I linearize this expression for small translations. Then, I express this approximation in terms of the perturbed reference-image viewpoints.

4. I characterize the holes created by failure of the point-on-line condition. Such holes are formed at foreground-background edges. I describe the worst-case orientation for such edges, and derive an expression for the resulting hole size.

5. Finally, I tie everything together to derive an expression for the largest possible hole size anywhere in the destination image. I evaluate this expression for an example case, and show that this result, computed using 2D image-space analysis, matches the result obtained from a 3D analysis.

I believe that the derivations in this section are more interesting for the insight they provide into the image-space behavior of the 3D warp than they are for the bounds computations that they enable. It is for this reason that I have included the derivations in such detail in this dissertation. There is already an example of the utility of the insight provided by this work. The intuition I developed from the figures and equations in this section provided important guidance in the development of the hole-filling algorithm described in the previous section of this chapter.

3.3.1 The point-on-line condition's meaning in image space

The point-on-line condition has an interesting meaning in image space. If this condition is satisfied, the epipoles from reference image A and reference image B will be coincident in the destination image.

A brief explanation will make this property clear. When warping a single reference image to a destination image, the destination-image epipole is the projection of the reference-image center of projection onto the destination image. When warping two reference images, if the two reference-image centers of projection and the destination-image center of projection are all collinear, then the epipole from the first reference image will be at the same location in the destination image as the epipole from the second reference image.

One of the coincident epipoles will be a positive epipole, and the other will be a negative epipole (see [McMillan97] for an explanation of epipole types).

We can show all of this mathematically. A point $\dot{C}_2'$ on the line segment between reference-image centers of projection $\dot{C}_A$ and $\dot{C}_B$ can be expressed as:

$$\dot{C}_2' = (1-t)\dot{C}_A + t\dot{C}_B. \qquad t \in [0,1] \tag{3.10}$$

The viewpoint-on-line condition can be expressed as

$$\dot{C}_2 = \dot{C}_2', \qquad \text{(Viewpoint-on-line)} \tag{3.11}$$

where $\dot{C}_2$ represents the destination-image center of projection.

As discussed in Chapter 1, if $\dot{C}_1$ represents the center of projection of a reference image (either A or B), and $\vec{e}_2$ is the location (in $\mathbb{P}^2$ homogeneous coordinates) of the epipole in the destination image, then

$$\vec{e}_2 \equiv \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2), \qquad \text{where} \qquad \vec{e}_2 = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}, \tag{3.12}$$

With two reference images, A and B, we have two $\mathbb{P}^2$ epipoles in the destination image (and thus two pairs of potential $\mathbb{R}^2$ epipoles):

$$\vec{e}_{2A} \equiv \mathbf{P}_2^{-1}(\dot{C}_A - \dot{C}_2) \qquad \text{and} \qquad \vec{e}_{2B} \equiv \mathbf{P}_2^{-1}(\dot{C}_B - \dot{C}_2) \tag{3.13}$$

Substituting for $\dot{C}_2$ using equations 3.11 and 3.10 enforces the viewpoint-on-line condition. The following equations give the resulting $\mathbb{P}^2$ epipole coordinates. The * on the epipole variables indicates that the expressions are valid only for the viewpoint-on-line condition. In the next subsection this condition will be relaxed, and the * will be removed.

$$\vec{e}^{\,*}_{2A} = \mathbf{P}_2^{-1}(\dot{C}_A - ((1-t)\dot{C}_A + t\dot{C}_B)) \qquad \text{and} \qquad \vec{e}^{\,*}_{2B} = \mathbf{P}_2^{-1}(\dot{C}_B - ((1-t)\dot{C}_A + t\dot{C}_B)) \tag{3.14}$$

Simplifying:

$$\vec{e}^{\,*}_{2A} = \mathbf{P}_2^{-1}(-t(\dot{C}_B - \dot{C}_A)) \qquad \text{and} \qquad \vec{e}^{\,*}_{2B} = \mathbf{P}_2^{-1}((1-t)(\dot{C}_B - \dot{C}_A)) \tag{3.15}$$

Since $\vec{e}_{2A}$ and $\vec{e}_{2B}$ are expressed in homogeneous coordinates, the multiplicative constants $-t$ and $(1-t)$ can be eliminated, showing that the two epipoles coincide. Alternatively, we can express the locations of the epipoles in image space to show that they coincide. Since we will need this image-space representation later in this section anyway, we will describe it now. First, we need two definitions:

$$\vec{b} \equiv \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \equiv \dot{C}_B - \dot{C}_A \qquad \text{and} \qquad \vec{b}' \equiv \begin{bmatrix} b'_x \\ b'_y \\ b'_z \end{bmatrix} \equiv \mathbf{P}_2^{-1}\vec{b} \tag{3.16}$$

Then,

$$\vec{e}^{\,*}_{2A} = -t\vec{b}' \qquad \text{and} \qquad \vec{e}^{\,*}_{2B} = (1-t)\vec{b}' \tag{3.17}$$

We can compute the image-space coordinates of both epipoles, using the definition of image-space epipole coordinates e_u and e_v from Equation 1.18. As expected, the image-space coordinates of the two epipoles coincide:

$$\begin{aligned} e^*_{u,2A} &= e^*_{u,2B} = \frac{b'_x}{b'_z} \\ e^*_{v,2A} &= e^*_{v,2B} = \frac{b'_y}{b'_z} \end{aligned} \tag{3.18}$$

Henceforth in this section, this shared epipole location that results from the point-on-line condition will be described using the variables $e^*_{u,2}$ and $e^*_{v,2}$, which lack the A or B subscript:

$$\begin{aligned} e^*_{u,2} &= \frac{b'_x}{b'_z} \\ e^*_{v,2} &= \frac{b'_y}{b'_z} \end{aligned} \tag{3.19}$$

3.3.2 Diverging epipoles

If the destination-image center of projection begins to stray from the line segment between the two reference-image centers of projection, then the coincident epipoles will begin to diverge as well.

Let $\vec{d}$ represent the deviation of the destination-image center of projection from the line segment $\overline{C_A C_B}$. We choose $\vec{d}$ so that it is always perpendicular to $\overline{C_A C_B}$. Figure 3.21 illustrates this configuration, in which

$$\dot{C}_2 = \dot{C}'_2 + \vec{d}, \qquad \text{(Viewpoint-off-line)} \tag{3.20}$$

with $\dot{C}'_2$ satisfying the point-on-line condition of equation 3.10, but $\dot{C}_2$ no longer satisfying this same condition.

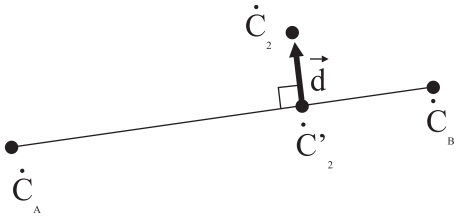


Figure 3.21: *Perturbation of the destination-image center of projection from the line segment between the two reference-image centers of projection.*

If the direction of the deviation $\vec{d}$ is given by $\hat{\vec{d}} \equiv \frac{\vec{d}}{\left\|\vec{d}\right\|}$, then we can define the magnitude of the deviation with a scale factor $\beta \equiv \frac{\vec{d}}{\hat{\vec{d}}}$, so that:

$$\vec{d} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \beta\hat{\vec{d}}. \tag{3.21}$$

The following related definition, for $\vec{d}$ transformed into the destination image's $\mathbb{P}^2$ coordinate system, will be useful later:

$$\hat{\vec{d}'} \equiv \mathbf{P}_2^{-1}\hat{\vec{d}} \qquad \text{with} \qquad \hat{\vec{d}'} = \begin{bmatrix} \hat{d'}_x \\ \hat{d'}_y \\ \hat{d'}_z \end{bmatrix} \tag{3.22}$$

Using substitutions from equations 3.20 and 3.10, we have the following expression for a destination-image epipole:

$$\begin{aligned} \vec{e}_2 &= \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2) \\ &= \mathbf{P}_2^{-1}(\dot{C}_1 - (\dot{C}'_2 + \vec{d})) \\ &= \mathbf{P}_2^{-1}(\dot{C}_1 - ((1-t)\dot{C}_A + t\dot{C}_B + \beta\hat{\vec{d}})) \\ &= \mathbf{P}_2^{-1}(\dot{C}_1 - (1-t)\dot{C}_A - t\dot{C}_B - \beta\hat{\vec{d}}) \end{aligned} \tag{3.23}$$

So, substituting $\dot{C}_A$ or $\dot{C}_B$ for $\dot{C}_1$, we get expressions for both of the (diverging) epipoles:

$$\begin{aligned} \vec{e}_{2A} &= \mathbf{P}_2^{-1}(-t(\dot{C}_B - \dot{C}_A) - \beta\hat{\vec{d}}), \qquad \text{and} \\ \vec{e}_{2B} &= \mathbf{P}_2^{-1}((1-t)(\dot{C}_B - \dot{C}_A) - \beta\hat{\vec{d}}). \end{aligned} \tag{3.24}$$

By using Equation 3.16 and multiplying by scale factors, we can write Equation 3.24 as

$$\begin{aligned} \vec{e}_{2A} &\doteq \mathbf{P}_2^{-1}(\vec{b} + \tfrac{\beta}{t}\hat{\vec{d}}), \qquad \text{and} \\ \vec{e}_{2B} &\doteq \mathbf{P}_2^{-1}(\vec{b} - \tfrac{\beta}{1-t}\hat{\vec{d}}). \end{aligned} \tag{3.25}$$

where the $\doteq$ symbol represents homogeneous coordinate equality (equality within a scale factor).

The presence of the $1/t$ factor in the first of Equations 3.25 shows that if $\dot{C}'_2$ is close to $\dot{C}_A$ (i.e. $t \approx 0$), then the location of $\vec{e}_{2A}$ is very sensitive to even small violations of the viewpoint-on-line condition (i.e. small $\vec{d}$). A similar observation holds for $\vec{e}_{2B}$.

Now, we can re-express Equations 3.25 in image-space coordinates:

$$\begin{aligned} e_{u,2A} &= \frac{b'_x + \frac{\beta}{t}\hat{d}'_x}{b'_z + \frac{\beta}{t}\hat{d}'_z} \qquad & e_{v,2A} &= \frac{b'_y + \frac{\beta}{t}\hat{d}'_y}{b'_z + \frac{\beta}{t}\hat{d}'_z} \\ e_{u,2B} &= \frac{b'_x - \frac{\beta}{1-t}\hat{d}'_x}{b'_z - \frac{\beta}{1-t}\hat{d}'_z} \qquad & e_{v,2B} &= \frac{b'_y - \frac{\beta}{1-t}\hat{d}'_y}{b'_z - \frac{\beta}{1-t}\hat{d}'_z} \end{aligned} \tag{3.26}$$

These equations can be linearized for small perturbations of the epipole (i.e. small magnitudes of $\vec{d}$). The details of this linearization are in Appendix B. To express the linearization, we first define

$\Delta\bar{e}_{2A} = (\Delta e_{u,2A}\ ,\ \Delta e_{v,2A})$, and $\Delta\bar{e}_{2B} = (\Delta e_{u,2B}\ ,\ \Delta e_{v,2B})$ to represent the perturbation of the epipoles away from the original location $\bar{e}_2^* = \left(e_{u,2}^*\ ,\ e_{v,2}^*\right)$, as follows:

$$\begin{aligned} \bar{e}_{2A} &= \bar{e}_2^* + \Delta\bar{e}_{2A} \\ \bar{e}_{2B} &= \bar{e}_2^* + \Delta\bar{e}_{2B} \end{aligned} \tag{3.27}$$

Using these definitions, the result of linearizing Equation 3.26 is:

$$\begin{aligned} \Delta e_{2A} &= \left[\frac{1}{t}\cdot\frac{d'_z}{b'_z}\right] \left(\tfrac{d'_x}{d'_z} - \tfrac{b'_x}{b'_z}\quad,\quad \tfrac{d'_y}{d'_z} - \tfrac{b'_y}{b'_z}\right) \\ \Delta e_{2B} &= -\left[\frac{1}{1-t}\cdot\frac{d'_z}{b'_z}\right] \left(\tfrac{d'_x}{d'_z} - \tfrac{b'_x}{b'_z}\quad,\quad \tfrac{d'_y}{d'_z} - \tfrac{b'_y}{b'_z}\right). \end{aligned} \tag{B.9}$$

Figure 3.22 illustrates this linear approximation of epipole movement in the destination image, as a result of small deviations from the viewpoint-on-line condition. The slope of the line representing the direction of perturbation is:

$$\frac{dy}{dx} = \frac{b'_z\hat{d}'_y - b'_y\hat{d}'_z}{b'_z\hat{d}'_x - b'_x\hat{d}'_z} \tag{3.28}$$

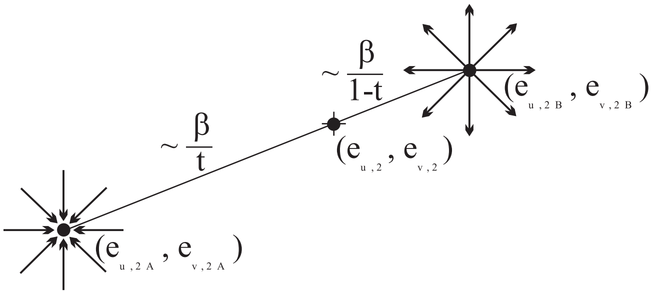


Figure 3.22: *For small deviations from the viewpoint-on-line condition, the two destination-image epipoles are perturbed in opposite directions from their initial common location. One of the perturbed epipoles is a positive epipole, and the other is a negative epipole.*

3.3.3 Behavior of warped points due to viewpoint translation

The eventual goal of this section (3.3) is to describe the occlusion errors that can result when the point-on-line condition is violated. The next step towards this goal is to derive a linearized expression which describes the image-space movement of warped points due to viewpoint translation.

From the 3D warp equation defined in Chapter 1, we can derive the following symmetric warp equation:

$$\dot{C}_2 + \frac{\mathbf{P}_2}{S_2} z_2 \, \bar{u}_2 = \dot{C}_1 + \frac{\mathbf{P}_1}{S_1} z_1 \, \bar{u}_1, \tag{3.29}$$

where

$$\bar{u}_1 = \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}, \qquad \bar{u}_2 = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}. \tag{3.30}$$

Rearranging, we get:

$$\bar{u}_2 = \frac{S_2}{z_2} \mathbf{P}_2^{-1} (\dot{C}_1 - \dot{C}_2) + \frac{S_2}{z_2} \mathbf{P}_2^{-1} \frac{\mathbf{P}_1}{S_1} z_1 \, \bar{u}_1 \tag{3.31}$$

If we ignore the translation part of the warp, we get a new image-space location that we call $\vec{u}_2'$:

$$\vec{u}_2' = \frac{S_2}{z_2} \mathbf{P}_2^{-1} \frac{\mathbf{P}_1}{S_1} z_1 \, \bar{u}_1, \qquad \vec{u}_2' \equiv \begin{bmatrix} u_{2,x}' \\ u_{2,y}' \\ u_{2,z}' \end{bmatrix} \equiv \begin{bmatrix} w_2' \, u_2' \\ w_2' \, v_2' \\ w_2' \end{bmatrix} \tag{3.32}$$

Note that the third coordinate of $\vec{u}_2'$ is not guaranteed to be 1. We use the notation $\bar{u}_2'$ (over-bar instead of over-arrow) to refer to this same point with a third coordinate of 1:

$$\bar{u}_2' \equiv \frac{1}{w_2'} \vec{u}_2' = \begin{bmatrix} u_2' \\ v_2' \\ 1 \end{bmatrix} \tag{3.33}$$

So, we can describe the destination-image movement due only to translation as:

$$\bar{u}_2 - \vec{u}_2' = \frac{S_2}{z_2} \mathbf{P}_2^{-1} (\dot{C}_1 - \dot{C}_2) \tag{3.34}$$

When warping two reference images, we are interested in how the destination-image translation of a particular object differs for the two reference images. Figure 3.23 illustrates this situation.

In Appendix B, I compute linearized expressions for the translation vectors $\Delta\bar{u}_{2A}$ and $\Delta\bar{u}_{2A}$ shown in Figure 3.23. These expressions show that the direction of the vectors is exactly towards or away from the appropriate epipole:

$$\begin{aligned} \Delta\bar{u}_{2A} &\approx \tfrac{S_2 e_{2A,z}}{z_2} (\bar{u}_2 - \bar{e}_{2A}) \\ \Delta\bar{u}_{2B} &\approx \tfrac{S_2 e_{2B,z}}{z_2} (\bar{u}_2 - \bar{e}_{2B}) \end{aligned} \tag{B.20}$$

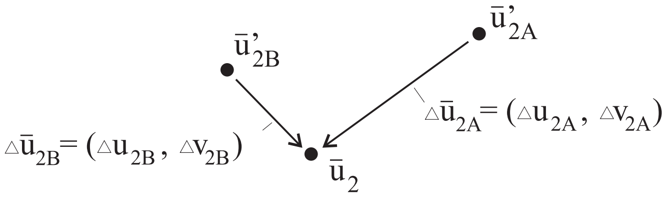


Figure 3.23: *Destination-image movement of a single object, due to translation. The same object is represented in two reference images. The pre-translation (post-rotation) destination-image location for the object's representation coming from reference image A is $\bar{u}'_{2A}$. The corresponding location for reference image B is $\bar{u}'_{2B}$. The final, post-translation, location of the object is $\bar{u}_2$. The arrows indicate the destination-image movement caused by translation, which is described by equation 3.34.*

The vectors $\Delta\bar{u}_{2A}$ and $\Delta\bar{u}_{2B}$, whose behavior is described by these expressions, are illustrated in Figure 3.23. Here are their formal definitions:

$$\Delta\bar{u}_{2A} \equiv \bar{u}_2 - \bar{u}'_{2A}, \qquad \text{and} \qquad \Delta\bar{u}_{2B} \equiv \bar{u}_2 - \bar{u}'_{2B} \tag{3.35}$$

If we are working with perturbed epipoles, then we can express $\bar{e}_{2A}$ and $\bar{e}_{2B}$ in terms of $\vec{b}$ and $\beta\hat{\vec{d}}$, using equations 3.24, 3.16, and 3.22. First, we express the $\mathbb{P}^2$ vectors $\vec{e}_{2A}$ and $\vec{e}_{2B}$ in the desired form:

$$\vec{e}_{2A} = -t\vec{b}' - \beta\hat{\vec{d}}', \quad \text{and} \quad \vec{e}_{2B} = (1-t)\vec{b}' - \beta\hat{\vec{d}}'. \tag{3.36}$$

Then, we switch to the image-space vectors $\bar{e}_{2A}$ and $\bar{e}_{2B}$ that we need for the substitution:

$$\begin{aligned} \bar{e}_{2A} &= \left(\frac{-tb'_x - \beta\hat{d}'_x}{-tb'_z - \beta\hat{d}'_z} \quad , \quad \frac{-tb'_y - \beta\hat{d}'_y}{-tb'_z - \beta\hat{d}'_z}\right) \\ \bar{e}_{2B} &= \left(\frac{(1-t)b'_x - \beta\hat{d}'_x}{(1-t)b'_z - \beta\hat{d}'_z} \quad , \quad \frac{(1-t)b'_y - \beta\hat{d}'_y}{(1-t)b'_z - \beta\hat{d}'_z}\right) \end{aligned} \tag{3.37}$$

We can substitute these equations into Equation B.20 to get the destination-image translation vectors for the case of perturbed epipoles:

$$\begin{aligned} \Delta\bar{u}_{2A} &\approx \frac{S_2\left(-tb'_z - \beta\hat{d}'_z\right)}{z_2}\left(\frac{-tb'_x - \beta\hat{d}'_x}{-tb'_z - \beta\hat{d}'_z} - u_2 \quad , \quad \frac{-tb'_y - \beta\hat{d}'_y}{-tb'_z - \beta\hat{d}'_z} - v_2\right) \\ \Delta\bar{u}_{2B} &\approx \frac{S_2\left((1-t)b'_z - \beta\hat{d}'_z\right)}{z_2}\left(\frac{(1-t)b'_x - \beta\hat{d}'_x}{(1-t)b'_z - \beta\hat{d}'_z} - u_2 \quad , \quad \frac{(1-t)b'_y - \beta\hat{d}'_y}{(1-t)b'_z - \beta\hat{d}'_z} - v_2\right) \end{aligned} \tag{3.38}$$

Note that it may sometimes be useful to further approximate by eliminating the $-\beta\hat{d}'_z$ term from the *common* factor. This approximation would affect the magnitude of the translation, but not the direction.

3.3.4 *Holes due to perturbation from point-on-line condition*

Holes are regions of the destination image which represent portions of the scene that are occluded in *both* reference images. For a single convex occluder (or equivalent), no holes will result if the point-on-line condition is honored. However, as the destination-image viewpoint deviates from the point-on-line condition, holes will appear.

We want to determine the size of these holes for perturbations from the point-on-line condition. To study this question, I consider the straight edge of an occluder, with the edge oriented such that a hole is produced next to it. The hole will extend along most of the length of the edge. The appropriate measure of hole size is thus the *width* of the hole (Figure 3.24).

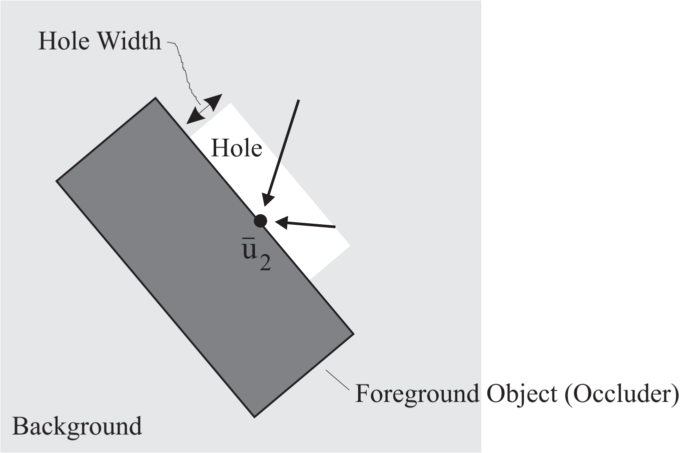


Figure 3.24: *Holes form along the edge of an occluder. The hole's severity is most appropriately measured by its width. We measure in this manner because the width of the hole depends only on the orientation of the edge, and is independent of the foreground object's size (once some minimum size is exceeded). In contrast, the length of the hole depends on the length of the foreground object.*

If we are given a particular $\bar{u}_2$, $\Delta\bar{u}_{2A}$, $\Delta\bar{u}_{2B}$, and occluder-edge orientation, then we can compute the size of the hole. Figure 3.25 shows geometrically how the hole size is determined from this information.

Typically, the edges of occluders in a scene have a variety of orientations. Since we would like to determine the worst-case hole size, we need to compute the edge orientation that yields the largest hole. Given a particular $\bar{u}_2$, $\Delta\bar{u}_{2A}$, and $\Delta\bar{u}_{2B}$, we can determine this worst-case edge orientation. The worst-case orientation varies across the destination image (i.e. the worst-case orientation is different at different values of $\bar{u}_2$). The orientation varies because the hole size depends on the directions of $\Delta\bar{u}_{2A}$ and $\Delta\bar{u}_{2B}$, and the directions of these vectors vary across the destination image.

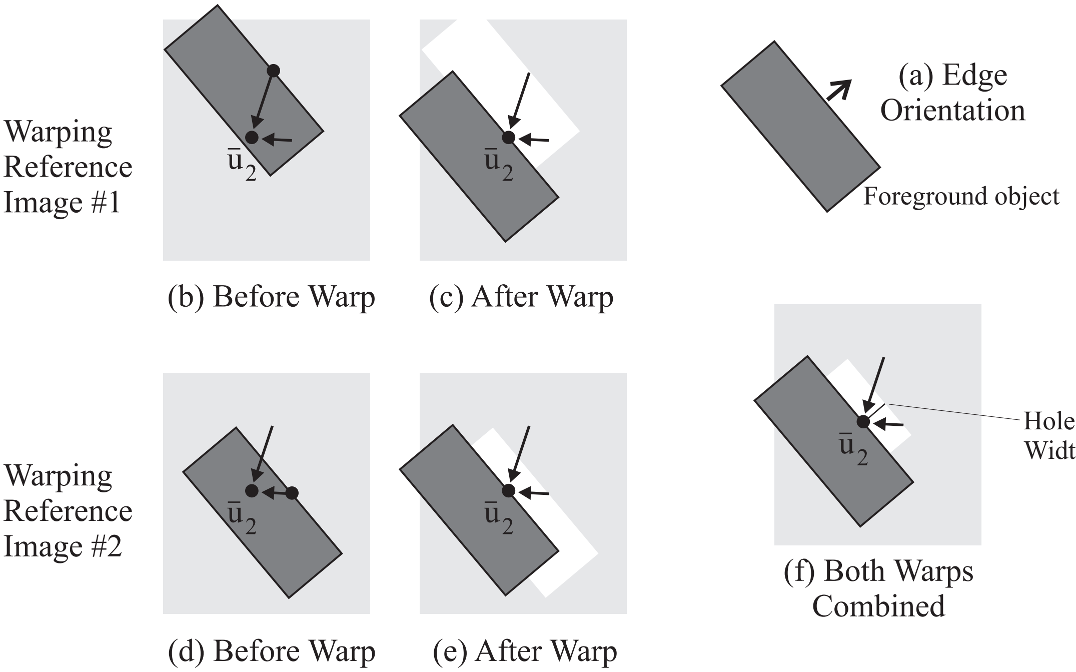


Figure 3.25: *Hole size for a particular foreground-object edge and pair of warps. (a) shows the orientation of the edge, in the destination image. The edge orientation can be represented by a normal vector. (b) shows reference image #1 before the warp, with the translation vectors for both warps shown. (c) shows the result of warping reference image #1. (d) shows reference image #2 before the warp. (e) shows the result of warping reference image #2. (f) shows the hole left behind after completing both warps. The thin line from $\bar{u}_2$ to the foreground-object edge represents the width of the hole. This line is perpendicular to the edge.*

Figure 3.26 illustrates the worst-case orientation of the occluder edge for two different examples of the vectors $\Delta\bar{u}_{2A}$ and $\Delta\bar{u}_{2B}$. In the first example (case #1), the worst-case edge orientation is indicated by the line q between the tails of the two warp translation vectors. In the second example (case #2), the worst-case edge orientation is perpendicular to the shorter of the two warp translation vectors. These two examples represent the two different mathematical cases for computing the worst-case edge orientation from $\Delta\bar{u}_{2A}$ and $\Delta\bar{u}_{2B}$.

In case #1, why is the chosen orientation the worst possible one? The answer is that the final hole size is determined, in effect, by the smaller of the holes generated by each reference image's warp. In case #1, these sizes are exactly equal. Any change from this orientation causes the hole generated by one of the two reference images to shrink. A change in orientation is equivalent to pivoting the edge about the tip of one of the two warp translation vectors.

In case #2, similar reasoning holds, but the geometric construction used for case #1 produces the wrong answer. This incorrect geometric construction produces a hole size smaller than the one we get

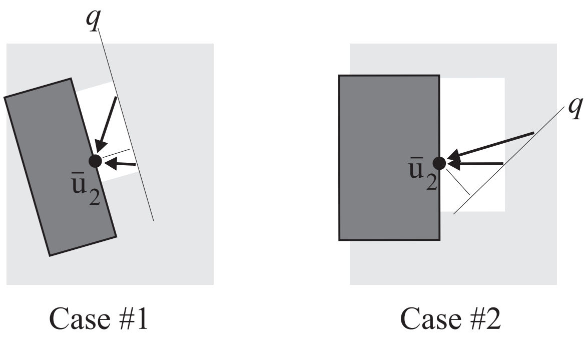


Figure 3.26: *There are two possible cases for the worst-possible edge orientation. To determine the correct case, begin by drawing a line, q, between the tails of the two translation vectors. Then, find the perpendicular from line q to point $\bar{u}_2$ (this perpendicular is the short, thin line in the figure). If the angle at which this perpendicular intersects $\bar{u}_2$ is in between the angles at which the two translation vectors leave $\bar{u}_2$, then case #1 is indicated. For case #1, the worst-possible edge orientation is parallel to line q. If the angle of the perpendicular is not in between the translation-vector angles, then case #2 is indicated. For case #2, the worst-possible edge is perpendicular to the shorter of the two translation vectors.*

by orienting the edge perpendicular to the smaller of the two warp translation vectors. So, in case #2 the worst-possible edge orientation is perpendicular to the shorter of the two warp translation vectors.

We can easily algorithmically distinguish between cases #1 and #2. Figure 3.27 illustrates the lengths and angles that I use. $\Delta\bar{u}_{2A}$ and $\Delta\bar{u}_{2B}$ represent the two image-space translational-movement vectors, as before. Then, let $\bar{q} = \Delta\bar{u}_{2B} - \Delta\bar{u}_{2A}$. If the sign of $\bar{q} \cdot \Delta\bar{u}_{2A}$ is different from the sign of $\bar{q} \cdot \Delta\bar{u}_{2B}$, then case #1 is indicated. If the signs of the dot products are the same, then case #2 is indicated.

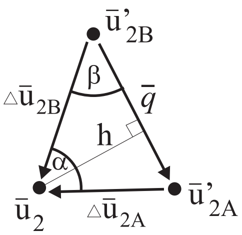


Figure 3.27: *Angles and lengths for hole-size computation.*

The hole size for case #2 is the length of the shorter of the two vectors $\Delta\bar{u}_{2A}$ and $\Delta\bar{u}_{2B}$. The hole size for case #1 can be calculated using trigonometric relationships. We can begin with one of two possible sets of information:

1. Given: $\Delta\bar{u}_{2A}$ and $\Delta\bar{u}_{2B}$. Easily compute $\|\bar{q}\|$.

2. Given: $\|\Delta\bar{u}_{2A}\|$, $\|\Delta\bar{u}_{2B}\|$, and α.

Using the law of cosines, we obtain the unknown length $\|\bar{q}\|$ or unknown angle α from the known values. Then, we can apply the law of sines to compute the angle β. Finally, from $\|\Delta\bar{u}_{2B}\|$, β, and the right angle, we can compute the hole size h:

$$h = \|\Delta\bar{u}_{2B}\| \sin(\beta). \tag{3.39}$$

3.3.5 Bound on hole size

Suppose that we are given the parameters of a warp. Then, using the equations derived in the previous few sections, we can calculate the worst-case occluder-edge orientation, and hence the maximum possible hole size at any *particular* location in the destination image.

But, can we establish a bound on hole size that is valid *anywhere* in the destination image? In this section, we derive such a bound using geometric arguments.

Equation B.20 gives the small-movement approximation for a 3D warp. These equations use the diverging epipoles produced by a perturbation away from the point-on-line condition. The equations show that image-space translational movement due to a warp is in the direction exactly towards or away from the corresponding epipole. The magnitude of the movement is the distance to the epipole, multiplied by a scale factor. The scale factor is different for the two warps — the first scale factor is $\frac{S_2}{z_2}e_{2A,z}$, and the second one is $\frac{S_2}{z_2}e_{2B,z}$.

Let us re-express Equation B.20 as follows:

$$\Delta\bar{u}_{2A} \approx s\bar{v}_A, \qquad \Delta\bar{u}_{2B} \approx s\gamma\bar{v}_B \tag{3.40}$$

where

$$s \equiv \frac{S_2 e_{2A,z}}{z_2} \qquad \text{and} \qquad \gamma \equiv -\frac{e_{2B,z}}{e_{2A,z}} \tag{3.41}$$

and

$$\bar{v}_A \equiv -(\bar{u}_2 - \bar{e}_{2A}) \tag{3.42}$$

$$\bar{v}_B \equiv \bar{u}_2 - \bar{e}_{2B}. \tag{3.43}$$

With this reformulation, $\bar{v}_A$ and $\bar{v}_B$ represent the vectors between the point being warped (P) and the epipoles (A and B). Figure 3.28 illustrates this situation, and shows the constructions used for the geometric argument that I will present. Note that we expect $\gamma > 0$ because the signs of $e_{2A,z}$ and $e_{2B,z}$ are different.

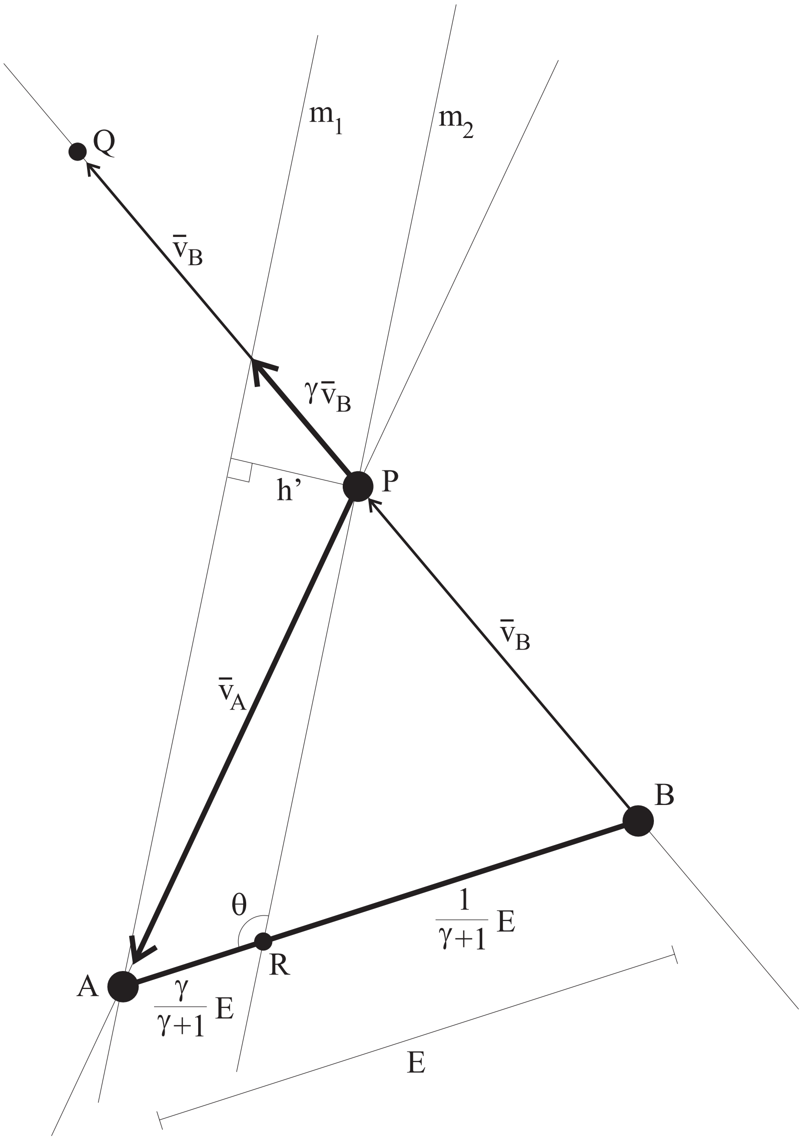


Figure 3.28: *Geometric argument for bound on hole-size* h*, under case #1.*

Vectors $\bar{v}_A$ and $\gamma\bar{v}_B$ (shown in the figure) are the scaled translational movement vectors that define the visibility hole in the figure. The figure depicts case #1 that was discussed in the previous subsection (3.3.5). The (scaled) size of the hole is thus defined by the length, h', shown in the figure. The true hole size, h, is related to h' by the scale factor: $h = sh'$. The true hole size h is measured in units of distance in the destination-image plane. This destination-image plane is at distance S_2 from the destination-image center of projection.

The figure shows a point, R, constructed in the figure. This point remains fixed for any fixed A, B, and γ, regardless of the location of P. We can show that this point is fixed because of the similar triangles formed by $\bar{v}_B$ and $\bar{v}_B + \gamma\bar{v}_B$ with respect to the vector between the epipoles. For this same reason, lines m_1 and m_2 are always parallel.

Thus, the hole size h' is determined by the distance between lines m_1 and m_2. This distance will reach a maximum when $\theta = \frac{\pi}{2}$. Thus, we have our bound:

$$h' \leq \frac{\gamma}{\gamma+1}E, \tag{3.44}$$

where

$$E = \|\bar{v}_A + \bar{v}_B\| \tag{3.45}$$

The argument we just made only strictly holds for case #1 of the hole-size computation. The transition between case #1 and case #2 is illustrated in Figure 3.29.

For points, P, inside the larger circle in Figure 3.29, the maximum hole size is $\|\gamma\bar{v}_B\|$. Since $\|\bar{v}_B\| \leq \frac{1}{\gamma+1}E$, we know that, inside this region,

$$h' \leq \frac{\gamma}{\gamma+1}E, \tag{3.46}$$

exactly as before.

The same bound applies inside the smaller circle as well, although in that case $\|\bar{v}_A\| \leq \frac{\gamma}{\gamma+1}E$.

Thus, making substitutions from earlier equations, the hole size h is bounded anywhere in the image plane:

$$h' \leq \frac{-\frac{e_{2B,z}}{e_{2A,z}}}{-\frac{e_{2B,z}}{e_{2A,z}}+1}\|\bar{e}_{2A} - \bar{e}_{2B}\| \tag{3.47}$$

Simplifying further, and using h rather than h',

$$h \leq S_2 \frac{-e_{2A,z}}{z_2\left(1 - \frac{e_{2A,z}}{e_{2B,z}}\right)}\|\bar{e}_{2A} - \bar{e}_{2B}\| \tag{3.48}$$

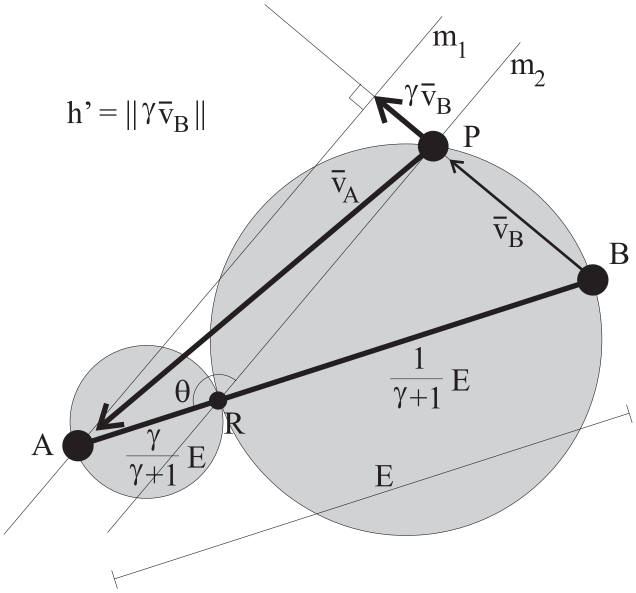


Figure 3.29: *Transition between hole-size case #1 and case #2. The transition occurs when point P lies at the edge of the grey circles. The interior of the circles represents the region in which case #2 is applicable.*

Remember that this bound is in fact only an approximation, because it relies on the approximations made earlier in this document. In particular, it relies on the linearization of the 3D warp, and on the small-perturbation approximation made for the deviation from the point-on-line condition.

Even so, this expression tells us some interesting things. First, it says that the hole size approaches zero as the “diverged” epipoles get closer. When the two epipoles coincide, indicating that the point-on-line condition is satisfied, the hole size is zero.

We can gain some additional insight into the meaning of this hole-size expression by substituting the parameters that define deviation from the point-on-line condition. From Equation B.8, we can make the following substitution:

$$\bar{e}_{2A} - \bar{e}_{2B} = \frac{d'_z}{b'_z\, t(1-t)} \left(\frac{d'_x}{d'_z} - \frac{b'_x}{b'_z} \quad , \quad \frac{d'_y}{d'_z} - \frac{b'_y}{b'_z} \right) \tag{3.49}$$

Then,

$$h \leq S_2 \frac{-e_{2A,z}}{z_2 \left(1 - \frac{e_{2A,z}}{e_{2B,z}}\right)} \cdot \frac{d'_z}{b'_z\, t(1-t)} \sqrt{\left(\frac{d'_x}{d'_z} - \frac{b'_x}{b'_z}\right)^2 + \left(\frac{d'_y}{d'_z} - \frac{b'_y}{b'_z}\right)^2} \tag{3.50}$$

This expression can be simplified further by making the following substitution (which in turn is generated from Equations 3.24, 3.16, 3.21, and 3.22):

$$e_{2A,z} = -tb'_z - d'_z \qquad \text{and} \qquad e_{2B,z} = (1-t)b'_z - d'_z \tag{3.51}$$

With this substitution, Equation 3.50 becomes:

$$h \leq S_2 \frac{t b'_z + d'_z}{z_2 \left(1 + \frac{t b'_z + d'_z}{(1-t) b'_z - d'_z}\right)} \cdot \frac{d'_z}{b'_z \, t(1-t)} \sqrt{\left(\frac{d'_x}{d'_z} - \frac{b'_x}{b'_z}\right)^2 + \left(\frac{d'_y}{d'_z} - \frac{b'_y}{b'_z}\right)^2} \tag{3.52}$$

After some simplification:

$$h \leq S_2 \frac{t(1-t)\left(b'_z\right)^2 - (2t-1) b'_z d'_z - \left(d'_z\right)^2}{z_2 b'_z} \cdot \frac{d'_z}{b'_z \, t(1-t)} \sqrt{\left(\frac{d'_x}{d'_z} - \frac{b'_x}{b'_z}\right)^2 + \left(\frac{d'_y}{d'_z} - \frac{b'_y}{b'_z}\right)^2} \tag{3.53}$$

What might this bound be under some reasonable conditions? I will examine an example set of such conditions. We know that $\vec{d} \perp \vec{b}$. Suppose that these vectors lie in the xz (image coordinates) plane, with each at a $\pm 45°$ angle from the z direction. For a $90°$ horizontal FOV, the vector $\frac{d'_x}{d'_z} - \frac{b'_x}{b'_z}$ spans the entire width of the image. Also, in this configuration b'_z is significantly greater than d'_z, since we expect that the magnitude of deviation from the point-on-line condition will be small with respect to the distance between the source images. Thus, we can make the approximation (which, in fact, usually holds) that $\left(d'_z\right)^2 \approx 0$. For a displayed frame near the midpoint between the reference frames, $t \approx \frac{1}{2}$. I only use this approximation for the $b'_z d'_z$ term. Using both of the approximations just described and choosing $\mathbf{P_2}$ such that $S_2 = 1$ allows dramatic simplification:

$$h_{\text{example}} \leq \frac{d'_z}{z_2} \sqrt{(\text{imagewidth})^2 + (0)^2} \tag{3.54}$$

I also need to specify the ratio between d'_z and z_2. This ratio is the ratio between the magnitude of the z-component of the deviation from the point-on-line condition and the z-component of the distance to the closest object in the scene. This dependence is what we would intuitively expect (except perhaps for the fact that only the destination-image z-component matters). Suppose that the distance to the nearest object in the scene is at least five times the distance between the two reference images. Furthermore, suppose that the deviation from the point-on-line condition is less than one fifth of the the distance between the reference images. Then, $\frac{\|\vec{d}\|}{z_2} = \frac{1}{25}$. For the configuration described, $d'_z = \frac{\sqrt{2}}{2}\left\|\vec{d}\right\|$. So,

$$h_{\text{example}} \leq \frac{(0.707)}{25} \cdot \text{imagewidth} = (0.028) \cdot \text{imagewidth} \tag{3.55}$$

So, for a 640 x 480 image under these conditions, the *maximum* hole size is less than or equal to approximately 18 pixels. For typical scenes, most holes will be substantially smaller than the maximum size.

3.3.6 3D calculation of hole size

In this section, I have computed an expression for a bound on hole size, using image-space calculations. The previous sub-section calculated the value of this bound for a particular example case. In this subsection, I show that we obtain approximately the same result using a 3-space calculation for a similar configuration.

The configuration is depicted in Figure 3.30. The "front object" depicted in the figure lies in the worst-case direction with respect to the two source image centers of projection. For a background object at infinity, the angle subtended on the displayed-image plane by the resulting hole is α. Using the dimensions from the figure, $\tan(\alpha) = \frac{1}{22.5}$. Thus, $\alpha = 2.54°$. For a 640 x 480 displayed image with $90°$ horizontal field of view, the hole size is thus 18.1 pixels. This result is almost the same as the result I calculated in the previous subsection using the image-space approach.

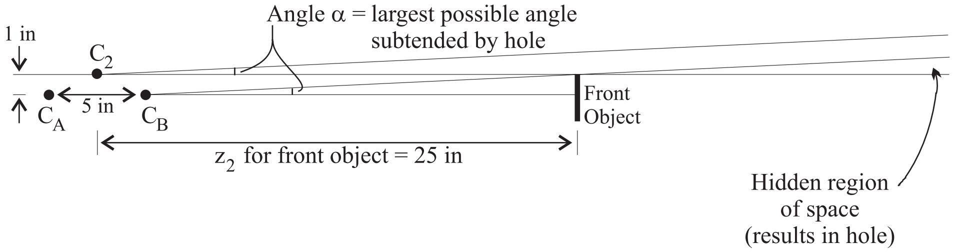

Angle α = largest possible angle subtended by hole
1 in
C2
CA
5 in
CB
Front Object
z2 for front object = 25 in
Hidden region of space (results in hole)

Figure 3.30: *The 3D geometry used for my example 3D calculation of hole size. C_A and C_B represent the two source-image centers of projection. The distance between these two centers of projection is $\|\vec{b}\| = 5$. The displayed-image center of projection is labeled C_2. The distance between this center of projection and the line between the two source images is $\|\vec{d}\| = 1$. C_2 is equidistant from C_A and C_B. The distance from the displayed-image center of projection to the nearest object in the scene is $z_2 = 25$. The hole that opens between the "front object" and a background surface at infinite distance (not shown) subtends an angle α from the displayed-image center of projection (C_2). For calculation purposes, I also show this angle relative to position C_B. Note that the angle α used in this figure is a different angle from the α used in Figure 3.27.*

3.4 Summary

This chapter has addressed the problem of visibility in a post-rendering warping system. First, I described a technique for choosing reference-frame viewpoints. The technique always warps two reference frames to produce each displayed frame. I then showed that the image quality produced by such a system is highly dependent on the accuracy of the position prediction subsystem. The efficiency of the post-rendering warping system depends on the increase in FOV required for the reference frames as compared to the displayed frames. This increase depends in turn on the viewer’s head rotation rate and on the frame rates of the reference frames and displayed frames.

Next, I presented a technique for filling any visibility holes that remain after warping both reference frames. Finally, I developed image-space expressions that bound the size of visibility holes. These expressions and the accompanying figures also provide qualitative insight into the properties of visibility holes.

CHAPTER 4

RECONSTRUCTION AND RESAMPLING

The previous chapter described how to choose viewpoints for reference-frame rendering. Given a pair (or more) of such reference frames, and a displayed-image viewpoint, how should a post-rendering-warping system compute the colors of the displayed-image pixels? This problem is the classical computer graphics problem of reconstruction[1] and resampling, as it applies to post-rendering warping. In this chapter, I discuss this problem.

I begin by developing a framework for thinking about the reconstruction and resampling problem for 3D image warping. In particular, I emphasize the importance of thinking about the problem as a 3D problem, not a 2D problem. The 2D reconstruction and resampling techniques used in classical image processing are not adequate for the problem of 3D warp reconstruction and resampling.

Next, I argue that a PRW system does not have sufficient information available for perfect reconstruction. Thus, it is necessary to develop heuristic approaches that work well in practice. I describe two variants of the approach that I have developed.

Finally, I describe a variety of alternative approaches, and discuss their advantages and disadvantages with respect to my preferred approach. In this last part of the chapter, I also discuss some of the work done by other researchers that is related to 3D warping reconstruction.

4.1 The problem

In 3D warping, we are given one or more reference images, and a viewpoint for which we want to compute a destination image. The simplest approach to this problem is to do the following for each reference-image pixel: Using the 3D warp equations, transform the pixel to compute its location in the

[1] Reconstruction is the process of calculating a continuous-domain representation of signal from a set of discrete samples of that signal.

destination-image space (Figure 4.1). This location will fall within some pixel in the destination image (assuming that the location is not off-screen). Perform a Z-buffer test at this pixel to see if the warped pixel is visible. If the warped pixel is visible, store the color obtained from the reference image in the destination-image pixel.

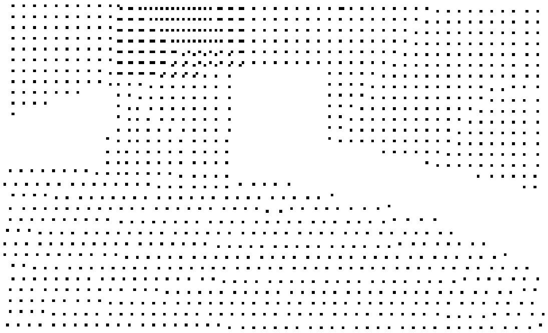


Figure 4.1: *When pixels are transformed from the reference image to the destination image, the transformed locations do not form a regular grid in the destination image. Instead, they are irregularly located. This figure shows the destination-image locations of transformed pixel centers for a portion of a frame from the kitchen walkthrough. Only one (of two) reference frame's transformed pixel centers are shown.*

This approach leads to noticeable artifacts. Figure 4.2 illustrates the type of artifacts that result. To avoid these artifacts, a more sophisticated approach is required that considers the structure of the three-dimensional scene represented by the reference image(s).

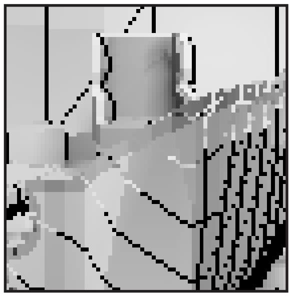


Figure 4.2: *The simplest form of resampling uses a one pixel reconstruction footprint for each reference pixel and causes holes to appear in surfaces that are slightly under-sampled. The effect is exaggerated in this figure for illustration purposes, by using a larger source-to-destination viewpoint distance than is typical for PRW. A later figure (Figure 4.16a) will provide a non-exaggerated example.*

I will now describe more carefully the information that is provided by the pixels in a reference image. If the reference image is generated by a conventional polygon renderer that does not perform anti-aliasing, then each pixel represents a point sample of a 2D surface that lies in 3-space. If the reference image is acquired from cameras or other sensing devices, then the pixels will typically represent area samples rather than point samples. The distinction between point samples and area samples is an important one. In particular, it is not always possible to associate a single depth value with an area sample. Since this dissertation concentrates on rendered imagery, I will assume from here on that the samples are point samples.

One can also think of each pixel in the reference image as representing the information provided by casting a ray through space until it hits a surface. This conceptual model makes it clear that each pixel indicates that a particular region of space is empty, as well as providing the location and color of a point on a 2D surface. The empty-space information provided by a pixel can be thought of as an infinitely dense set of 100% transparent samples. These samples lie along the pixel's ray, between the center of projection and the first 2D surface.

Conventional polygon renderers consider polygonal surfaces to be pure 2D surfaces represented at infinite precision. Thus, the surface sample represented by a pixel always represents a completely solid surface—there is no sense of partial presence of a surface, as one might have from volume rendering of a band-limited volumetric model. This property would not necessarily hold if pixels represented area samples rather than point samples.

Even though a sample is taken from a completely solid surface, it is possible for the solid surface to be partially transparent. Consider, for example, a polygonal representation of a colored glass surface. In my approach, I exclude this possibility by prohibiting the presence of partially transparent surfaces in the reference frames. This restriction is not strictly necessary for the purposes of reconstruction and resampling. Instead, the restriction is imposed because a single-layer reference-frame pixel can only represent one surface. If the first surface is partially transparent, then the surface behind it should visible but can not be represented. The simplest solution to this problem is to exclude partially transparent surfaces.

4.2 Ideal reconstruction

I have just described the information provided by the reference images. Given this information, how do we generate a destination image? In particular, how do we use the information provided by two or more reference images to generate a destination image?

One approach is to construct a best-estimate 3D model, based on the information provided by the reference images. The destination image is then generated by rendering this best-estimate 3D model. For post-rendering warping, it is clear that explicitly building such a 3D model would be undesirable—

the system already has the original polygonal 3D model, so why build a new 3D model? However, I will briefly discuss this approach, for two reasons. First, it is conceptually useful, and it helped to guide the development of the approach that I do use. Second, it is not necessarily an unreasonable approach for acquired imagery, if the acquired imagery can be pre-processed off-line (although a system that explicitly constructs a new 3D model stretches the definition of "image-based rendering").

There is an important property that the best-estimate 3D model should satisfy: When it is used to generate a destination image at one of the reference-image viewpoints, the destination image should be identical to the corresponding reference image.

Because we know that the 3D volume that we are trying to construct originated as a polygonal model, the reconstruction problem is simplified somewhat. Rather than trying to reconstruct a general 3D volumetric function, we are instead trying to reconstruct a set of 2D surfaces at various locations within the 3D volume.

Given this restriction, the reconstruction algorithm must answer two questions:

1. Where are the surfaces in 3-space?

2. What is the color at each point on these surfaces?

The most difficult part of the first problem is locating breaks between different surfaces (*discontinuities*). Figure 4.3 shows a set of surface samples to be checked for discontinuities. In this figure, it is easy to distinguish between the discontinuities and non-discontinuities, because we can see the true (continuous-domain) representation of the surface as well as the samples. However, in Figure 4.4a, which does not show the true surfaces, it is more difficult to distinguish the discontinuities from the non-discontinuities. The samples in Figure 4.4a could belong to any of the surface configurations shown in Figures 4.4b, 4.4c, and 4.4d. In one case there is a discontinuity between the middle pair of samples, and in the other two cases there is not.

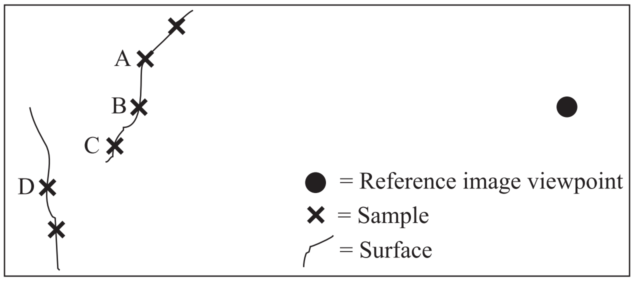
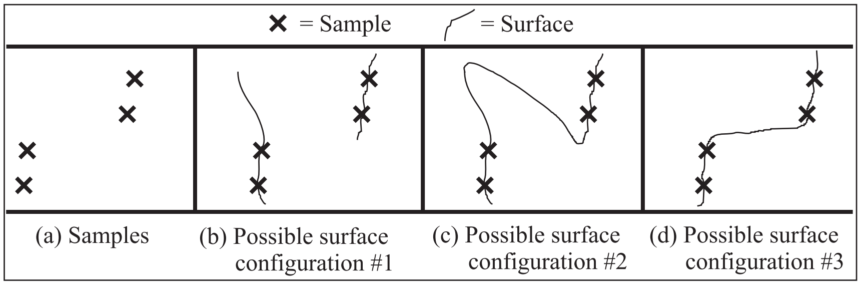


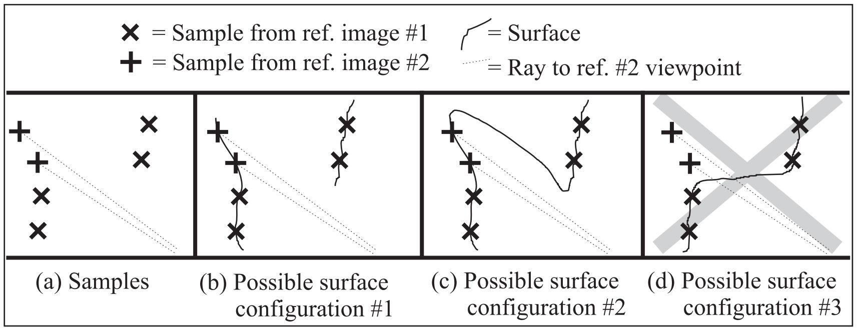
Figure 4.3: *A surface discontinuity. Samples C and D are clearly from different surfaces, while samples A and B are clearly from the same surface.*



✕ = Sample
= Surface
(a) Samples
(b) Possible surface configuration #1
(c) Possible surface configuration #2
(d) Possible surface configuration #3

Figure 4.4: *Different surface configurations are possible with the same set of samples. In every part of this figure, the reference-image viewpoint is far to the right.*

Often, the information provided by a second source image can resolve or partially resolve ambiguities of the sort depicted in Figure 4.4. Figure 4.5 shows the additional information added to Figure 4.4 by a second reference image. The empty-space information provided by this second reference image eliminates the surface configuration depicted in Figure 4.5d as a possibility.



✕ = Sample from ref. image #1
+ = Sample from ref. image #2
= Surface
= Ray to ref. #2 viewpoint
(a) Samples
(b) Possible surface configuration #1
(c) Possible surface configuration #2
(d) Possible surface configuration #3

Figure 4.5: *Information provided by a second reference image can resolve or partially resolve ambiguities in surface reconstruction. This figure shows that the two additional samples from a second reference image eliminate surface configuration #3 as a possibility.*

The reconstruction algorithm is not always able to resolve ambiguities like that shown in Figure 4.4 by using the information from an additional reference image. When only a small number of additional reference images are available, it is possible that none of them will have a view of the ambiguous region of space. The reconstruction algorithm must use a heuristic to resolve the ambiguity.

The visibility holes discussed in Chapter 3 are an example of this type of ambiguity. The holes result when an ambiguous region like the one shown in Figure 4.4a is visible in the destination image. The reconstruction algorithm heuristically decides that the situation depicted in 4.4b is the correct one. The result is a gap in the destination image, which is filled by the hole-filling algorithm. Although I discussed the hole-filling algorithm in the context of the visibility problem, from an algorithmic point of view it is actually part of the reconstruction algorithm.

With the limited information available from a few reference images, there will be reconstruction ambiguities. I can make the same argument more formally. There are two reasonable models of the 3D space that we could use for reconstruction of the 3D geometry. The first, and least appropriate, is a general volumetric density function, $f(x, y, z)$, such as one assumes in volume rendering. The second is a set of 2D surfaces $S_1 \ldots S_n$ located within 3-space, where $S_i : \vec{x} = S_i(s, t)$. I will discuss these two models in turn.

We can perfectly reconstruct a volumetric density function $f(x, y, z)$ if the following conditions are met:

1. $f(x, y, z)$ is band-limited to a frequency F.

2. $f(x, y, z)$ is uniformly sampled (in three dimensions) at or above the Nyquist rate, $2F$, in each dimension.

For the PRW reconstruction problem, neither of these conditions is met. The first condition is not met because the volumetric function contains infinite frequencies. These infinite frequencies are caused by the step function change from unoccupied space to occupied space at the polygonal surfaces. The second condition is not met because space is non-uniformly sampled. Even from a qualitative point of view, this non-uniform sampling causes some regions of space to be inadequately sampled.

The surface representation, $S_1 \ldots S_n$ is a more natural fit for the PRW reconstruction problem, since this representation corresponds more closely to a polygonal model. Generally speaking, we can reconstruct this surface model in a topologically correct manner with small, bounded error if three conditions are met [Amenta98]. These conditions must be met for all surfaces, or portions of surfaces, that are visible in the destination image:

1. The rate of surface curvature in (x, y, z) space is limited.

2. There is a minimum distance between different surfaces.

3. The surface location is sufficiently densely sampled in (x, y, z) space. The required sampling density varies locally, depending on the local maximum surface curvature and the local minimum inter-surface distance.

Once again, for the post-rendering warping problem, these conditions are not generally met. The first condition is not met because surface curvature is infinite at the edges between polygons that form polyhedral objects. The second condition is not met because there are generally no restrictions on the location of surfaces in a polygonal model. The third condition is not met at visibility holes, and possibly at other locations as well.

With either of the theoretical approaches to reconstruction it is clear that a PRW system has inadequate information available to perfectly reconstruct the needed portions of the scene. However, by using appropriate heuristics, it is possible to obtain good results in practice. In particular, the heuristics make use of an important property of PRW: the destination-image viewpoint is generally quite close to the reference-image viewpoints. As a result, the displayed image should be similar to the reference images. In the next section, I will begin to discuss the reconstruction algorithm that I have adopted, and the heuristics that it uses.

4.3 A more practical approach

Building a new, best-estimate 3D model from the samples provided by the reference images is not a practical approach to PRW reconstruction. A practical approach must work with the existing reference images, rather than building a completely new representation of the scene. My system performs the 3D reconstruction incrementally, as each source image is warped. Surfaces are independently reconstructed in each source image, and resampled on destination-image pixel centers. The resampled data originating from the different source images is then composited to form the displayed image.

Many reconstruction and resampling algorithms do not truly perform a full reconstruction prior to resampling, and mine follows this pattern—my algorithm never stores a representation of the reconstructed surfaces. The algorithm consists of the following steps:

1. **Transform** reference-image samples into destination-image space, retaining depth (3D) information.

2. Conceptually **reconstruct** 2D manifolds (surfaces) in 3-space.

   (a) Segment transformed samples into different surfaces.

   (b) Reconstruct each surface. (Note: The actual computation is in destination-image space.)

3. **Resample** the reconstructed surfaces at destination-image pixel centers, producing *candidate pixels*.

4. **Merge/Composite**: At each destination-image pixel, composite the candidate pixels coming from different surfaces to determine the final pixel content.

These steps are performed for each reference image in turn (more precisely, for local neighborhoods of pixels within each reference image in turn). Thus, the compositing step (#4) is performed incrementally. In addition to compositing candidate pixels originating from different surfaces, this step must also arbitrate between candidate pixels originating from the same surface represented in different reference images. I will discuss steps 2a, 2b, 3, and 4 in detail in the next few subsections.

4.3.1 Surface segmentation

For a single reference image, the surface segmentation problem can be reduced to a simpler problem: Given a pair of adjacent reference-image samples, decide whether or not they belong to the same surface. More precisely, the problem is to decide whether or not the adjacent samples belong to *adjacent* portions of the same surface (see Figure 4.6).

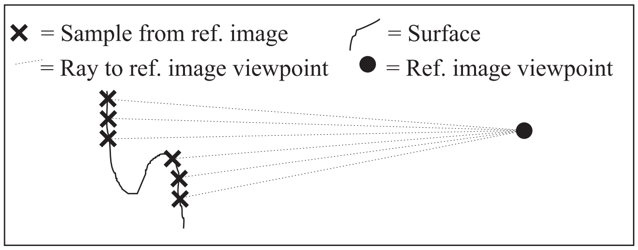


Figure 4.6: *When surface segmentation is performed using only a single reference frame, it is common for part of a surface to be occluded (possibly by itself, as shown here). In such cases, the surface segmentation algorithm considers the disjoint portions of the surface to be separate surfaces for the purposes of reconstruction.*

From the discussion earlier in this chapter, we know that even with information from all reference images, the surface segmentation problem is not necessarily correctly solvable. It is even more difficult when only the information from a single reference image is used, as I am requiring for my algorithm. For example, with the data provided in Figure 4.4a, the algorithm will not be able to correctly distinguish between the cases depicted in Figures 4.4b, 4.4c, and 4.4d.

For the problem of surface segmentation using a single source image, standard 2D signal processing techniques can provide some insight. The system is trying to detect discontinuities that indicate the transition from one surface to another. If the depth function (of image-space u and v) is locally band-limited in most regions, then we can consider a discontinuity to be a region of the signal with significant energy above the band-limit. Thus, if we sample substantially above the Nyquist rate for this band-limit, then we will be able to detect these discontinuities. However, if we sample at or below the Nyquist rate, then the discontinuities will be indistinguishable from the rest of the signal.

In PRW, the 2D depth function is not band-limited at all, even within a single surface. The reason is that there are no restrictions on the geometric behavior of surfaces. Thus, we can not distinguish the discontinuities from the rest of the signal.

It has been suggested that additional reference-image information in the form of object-ID numbers could allow better surface segmentation. There are two difficulties with the object-ID approach. First, it requires extensive integration with the scene graph layer of the rendering system, in order to propagate ID's from objects to pixels. I have consistently tried to avoid imposing this type of burden on the rendering system. Second, the object-ID approach does not work for non-convex objects, or for intersecting objects. Both are too common to ignore. A non-convex object can generate a case such as that shown in Figure 4.6, which should be considered to be a discontinuity. The object-ID approach will incorrectly consider this case to be a single surface. Conversely, intersecting surfaces from two different objects (or two differently ID'd portions of the same object) should be considered to be a single surface for the purpose of reconstruction. But, the object-ID approach will incorrectly consider them to be different surfaces, possibly causing pinholes to appear at the intersection between the two surfaces.

So, given this theoretically impossible situation, what can be done? A system can do quite well in practice by using a surface-segmentation algorithm that satisfies two criteria. First, the algorithm should work well for the common cases. In particular, when there is an ambiguity between an extremely under-sampled surface and a discontinuity (as in Figure 4.4), the algorithm should categorize the configuration as a discontinuity. Second, in borderline cases the algorithm should pick the choice that, if wrong, will produce the smallest error in the *perceptual* sense.

The algorithm that I have developed is very simple. First, it projects two adjacent reference-image samples into the destination-image plane, using the 3D warp transform. If the image-plane distance between the two projected samples is greater than a fixed threshold, then the two samples are considered to represent different surfaces. Otherwise, they are considered to represent the same surface. Figure 4.7 illustrates the algorithm.

This algorithm depends on the destination-image viewpoint. This dependence may seem inappropriate—why should the scene structure depend on the point from which it is viewed? But, this dependence is crucial to satisfying the goal of minimizing perceived error. Because the

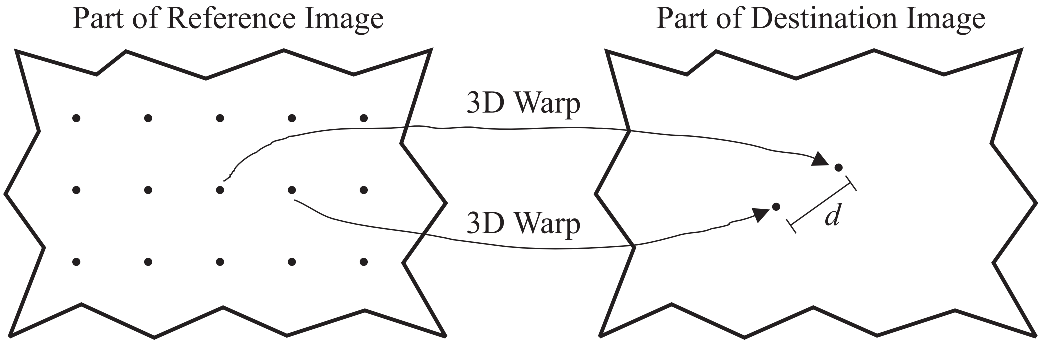
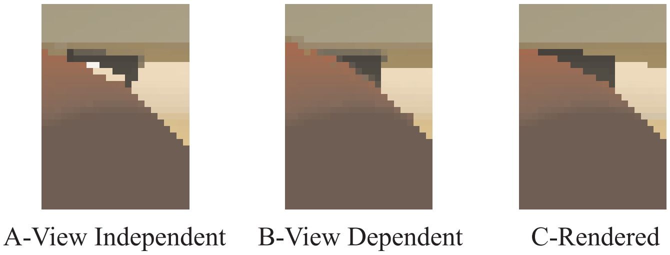


Figure 4.7: *The surface segmentation algorithm uses an image-space distance threshold to determine whether or not adjacent source-image samples represent the same surface. If the distance, d, between the transformed locations of the samples exceeds a fixed threshold, then the samples are considered to represent different surfaces.*

destination-image viewpoint is typically very close to the source-image viewpoint, most parts of the reference image should usually remain unchanged. The view-dependent surface-segmentation algorithm achieves this goal by only introducing discontinuities when absolutely necessary.

Figure 4.8 illustrates the advantage of the view-dependent algorithm using magnified images. One of my previous approaches to surface segmentation ([Mark97b] provides details) was destination-view independent. It relied solely on reference-image pixel depths and pixel normals to find discontinuities. This previous approach determined that a discontinuity existed between the red foreground object and the small, dark middle-depth object in Figure 4.8a. As a result, the light-colored background shows through the discontinuity gap. The new, view-dependent, algorithm (Figure 4.8b) recognizes that the size of the (possible) gap is so small that it should be ignored. In this case, the second choice happened to be right (Figure 4.8c). But more importantly, the perceptual consequences of incorrectly allowing a gap are much greater than the perceptual consequences of incorrectly disallowing a gap. In animation, an incorrect gap manifests itself as a disturbing flicker. An incorrect non-gap usually just delays the opening of the gap by a few frames, until a new reference image is available or the image-space gap threshold is exceeded. These errors are usually imperceptible.

If a surface is sufficiently under-sampled, then the view-dependent segmentation algorithm will not recognize it as a single surface. Such an under-sampled surface is one that is at a nearly grazing angle to the view rays in the reference image, but is at a non-grazing angle to the view rays in the destination image. In making this decision, the segmentation algorithm is getting the common case correct, since such samples usually belong to two distinct samples rather than to a single under-sampled surface. Furthermore, if the samples *do* represent an under-sampled surface, the surface will typically



A-View Independent B-View Dependent C-Rendered

Figure 4.8: *View-independent vs. view-dependent surface segmentation. (a) Shows the results from my previous, view-independent, technique. A gap incorrectly opens between the red foreground object and the brown mid-ground object, allowing the yellow/tan background to show through. (b) Shows the results from my view-dependent technique. (c) Shows the results from conventional rendering.*

be represented at a higher sampling density in another reference image, so that no artifacts will be visible in the displayed image.

The view-dependent segmentation algorithm is not particularly sensitive to the value used for the destination-image-plane distance threshold. The threshold value is expressed as a percentage of the "expected" distance, in pixels, between adjacent transformed samples. The expected distance distance is defined as the distance between transformed samples for a planar surface facing the reference image center of projection. I have successfully used thresholds of 140% and 230% of the expected distance. Higher values can cause slight blurring at some foreground/background edges, where there are discontinuities that are not recognized as such. Lower values can occasionally allow pinholes of the type shown in Figure 4.8a. For PRW with five reference frames/sec and thirty displayed frames/sec, I have settled on a threshold value of 140% of the expected distance. For lower reference-frame rates, a larger threshold value should be used to avoid pinholes.

4.3.2 Reconstructing and resampling each surface

Once the reference image has been segmented into different surfaces, each surface must be reconstructed and resampled. Ideally, each surface reconstruction would be done in 3-space. Each reconstructed surface would then be projected into 2D destination-image space, and resampled at destination-image pixel centers. As is common in computer graphics, I approximate the 3-space reconstruction by reconstructing in the 2D destination-image space.

Within this framework, I have developed two different approaches to reconstruction and resampling of surfaces. The first approach is generally applicable, but somewhat expensive. It relies on explicit interpolation between transformed samples. The second approach is designed to be used in conjunction with super-sampled anti-aliasing. It avoids explicit interpolation by relying on the implicit interpolation performed by the averaging of the super-samples prior to display. Both approaches require that extra data be stored with each reference-frame and displayed-frame pixel. Appendix C summarizes the contents of reference-frame and displayed-frame pixels.

General algorithm

The more general of the two surface reconstruction and resampling algorithms is illustrated in Figure 4.9.

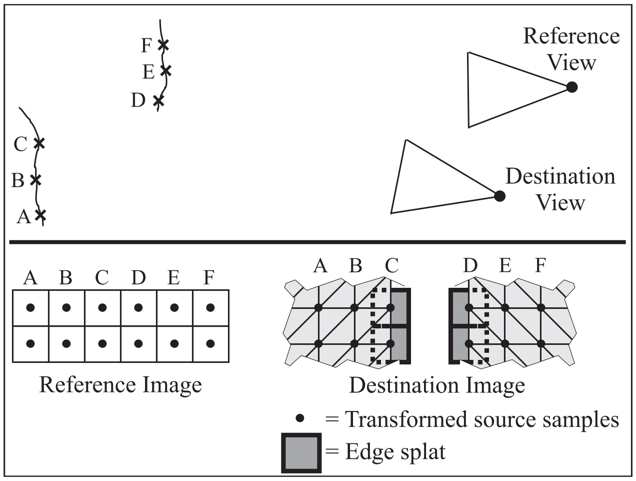
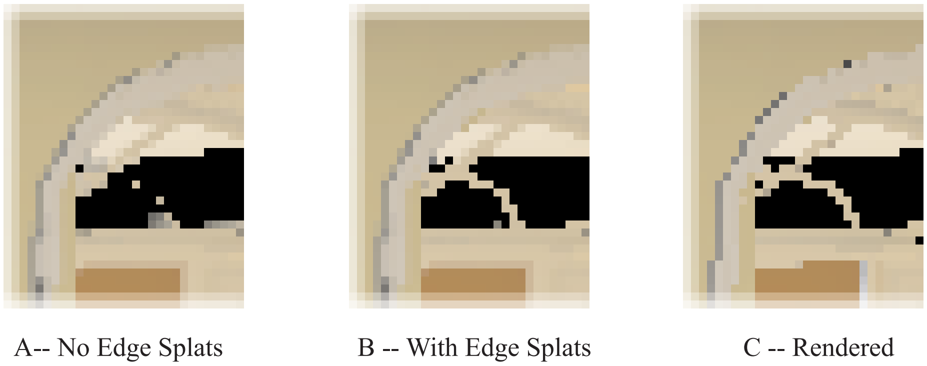


Figure 4.9: *My general reconstruction and resampling technique. The top half of the figure is a top view of a scene. The bottom half shows the corresponding reference and destination images. The destination image depicts the transformed samples, the triangle mesh used for reconstructing the interiors of surfaces, and the edge splats used for reconstructing the edges of surfaces.*

In the interior regions of surfaces, the algorithm reconstructs by treating the surface as a triangle mesh. The reference-image samples form the vertices of the mesh. These vertices are warped to destination-image coordinates, and the mesh is rasterized in the destination image space. The candidate pixels produced by this rasterization are then composited into the destination image. Colors and depths (depth is represented as $\frac{1}{Z}$) are linearly interpolated in destination image space by the rasterization. Linear interpolation of colors in image space is not equivalent to linear interpolation of colors in 3-space, but the error is generally negligible for small polygons. It is for this reason that the 2D reconstruction of each surface is an acceptable substitute for 3D reconstruction of each surface.

Using just this technique effectively shaves one-half of a pixel off the edges of all surfaces. The most extreme example is a one-pixel-wide line, which disappears completely. Figure 4.10a illustrates this problem in a magnified image, and Figure 4.10b illustrates our solution, which performs extra reconstruction at surface edges.



A-- No Edge Splats

B -- With Edge Splats

C -- Rendered

Figure 4.10: *Using edge splats improves image quality. Anti-aliasing was disabled for all of the following three images: (a) An image produced without using edge splats. Parts of the window trim disappear completely. (b) An image produced using edge splats. (c) A conventionally rendered image, for comparison purposes.*

This extra reconstruction treats pixels at the edge of a surface specially. I define an edge pixel as one which has a discontinuity between it and at least one of its source-image neighbors. For edge pixels, the reconstruction algorithm performs a splat-like [Westover90] quadrilateral reconstruction. The algorithm calculates the corners of this *edge splat* using the sample's surface orientation, by assuming that the sample comes from a planar surface. The details of this calculation will be discussed later in this chapter.

Because the splat color is not interpolated, the algorithm should avoid overwriting portions of the adjacent mesh color with the splat color. A bit at each pixel in the destination image ("*SplatFlag*" in Appendix C) indicates whether the pixel originated from mesh or splat reconstruction. A splat pixel is never allowed to overwrite a mesh pixel that is at the same depth (within a tolerance). Thus, the edge splat only contributes to the portion of the surface outside the meshed interior. Figure 4.9 shows the edge splat contributions in dark grey.

The introduction of edge splats can produce a new type of undesirable artifact. If the same surface is represented in two or more source images, but is extremely under-sampled in one of the images, then we would like to reconstruct it using the source image with the best samples. Normally, the compositing algorithm (discussed later) solves this problem. The compositing algorithm arbitrates between competing samples that represent the same 3D surface but come from different source images by comparing their sampling densities. However, an edge splat from an under-sampled reference image can produce a rough edge that will stick out further in the destination image than the same edge from a better-sampled image. The protruding edge pixels will never have the opportunity to lose to better samples, and will thus remain in the final image.

The reconstruction algorithm avoids this problem by suppressing an edge splat when both of the following two conditions are met:

1. The surface is poorly sampled. The algorithm designates the surface as poorly sampled if the density of transformed samples in the destination image is less than 0.5 samples per destination-image pixel. The sample density is calculated using the surface orientation in the reference-image, the sample depth, and the source- and destination-image viewpoints. This sample-density computation assumes a planar surface.

2. Another source image is expected to sample the surface better. The assumption is made that the surface will be visible in the other source image.

This enhancement to the algorithm achieves the desired result, except when the “better” source image does not in fact sample the surface at all, due to occlusion. In this instance, the result is the same as that obtained without the edge-splat enhancement: one-half of a pixel is shaved off surface edges.

I often refer to this reconstruction and resampling algorithm as my *hybrid mesh/splat* algorithm, because it treats the interior regions of surfaces like a triangle mesh, and treats the edges like a splat. I will refer to the algorithm by this name later in the chapter.

Algorithm for use with anti-aliasing

I have developed a second reconstruction algorithm which is specialized for use with anti-aliased PRW. To produce good-quality results, this algorithm also requires an approximately 1-to-1 ratio between the angle subtended by a reference-image sub-pixel and the angle subtended by a displayed-image sub-pixel. Figure 4.11 shows a displayed frame produced using this algorithm. Before discussing the algorithm, I will briefly describe how anti-aliasing works in conjunction with PRW.

When performing super-sampled anti-aliasing for PRW, both the reference images and the destination image should be represented at super-sampled resolution. The 3D warp works at super-sampled resolution as well—the final averaging of the super-samples occurs after the warp. It would be incorrect to average the super-samples before the warp, because such averaging would blend foreground and background surfaces which ought to move differently during the warp [Chen93]. In this subsection, when I refer to “pixels”, I am actually discussing the super-samples.

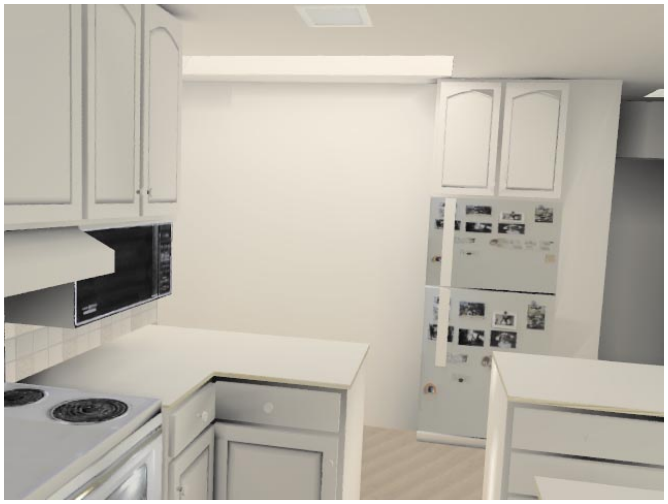


Figure 4.11: *An anti-aliased displayed frame, produced with my reconstruction algorithm designed for use with anti-aliasing.*

When both the source and destination images are super-sampled, the reconstruction algorithm can be greatly simplified. My approach to anti-aliased reconstruction was inspired by the REYES system's flat-shaded micro-polygons [Cook87]. Because the reconstruction algorithm can rely on the averaging of super-samples to implicitly perform color interpolation, there is no longer any need to explicitly interpolate colors. My anti-aliased reconstruction algorithm also simplifies the geometric portion of the reconstruction—the algorithm uses axis-aligned rectangles rather than triangles. This approach is designed to be easily and cheaply implemented in hardware.

Although the anti-aliased reconstruction algorithm no longer explicitly interpolates colors or depths between adjacent samples, it does not completely ignore the connectivity of a surface. If the algorithm did ignore this relationship, by independently splatting each source pixel into the destination image, it would generate pinholes and other artifacts.

The algorithm begins the reconstruction process by transforming each source-image sample to determine its location in the destination image (Figures 4.12a and 4.12b). Next, the algorithm computes the extent of the reconstruction footprint for each sample. Initially, this reconstruction footprint is a general quadrilateral (Figure 4.12c). For the interior regions of a surface, each corner of the quadrilateral is computed by averaging, in 2D destination-image space, the coordinates of the four transformed samples surrounding that corner. For example, in Figure 4.12c, corner #1's location is computed by averaging the coordinates of points A, B, C, and D. Figure 4.12c only shows the reconstruction footprint for sample D, but corner #1 is shared with the reconstruction footprints for samples A, B, and C (Figure 4.12d). This corner sharing guarantees that there are no cracks in the mesh that could cause pinholes. Finally, the algorithm converts the reconstruction footprint's extent from an arbitrary quadrilateral to an axis-aligned rectangle, to increase rasterization efficiency. This conversion is performed by taking the arbitrary quadrilateral's axis-aligned bounding box as the reconstruction footprint (Figure 4.12e). All pixel centers inside the axis-aligned footprint are filled with the sample's color and the sample's transformed $1/Z$ value.

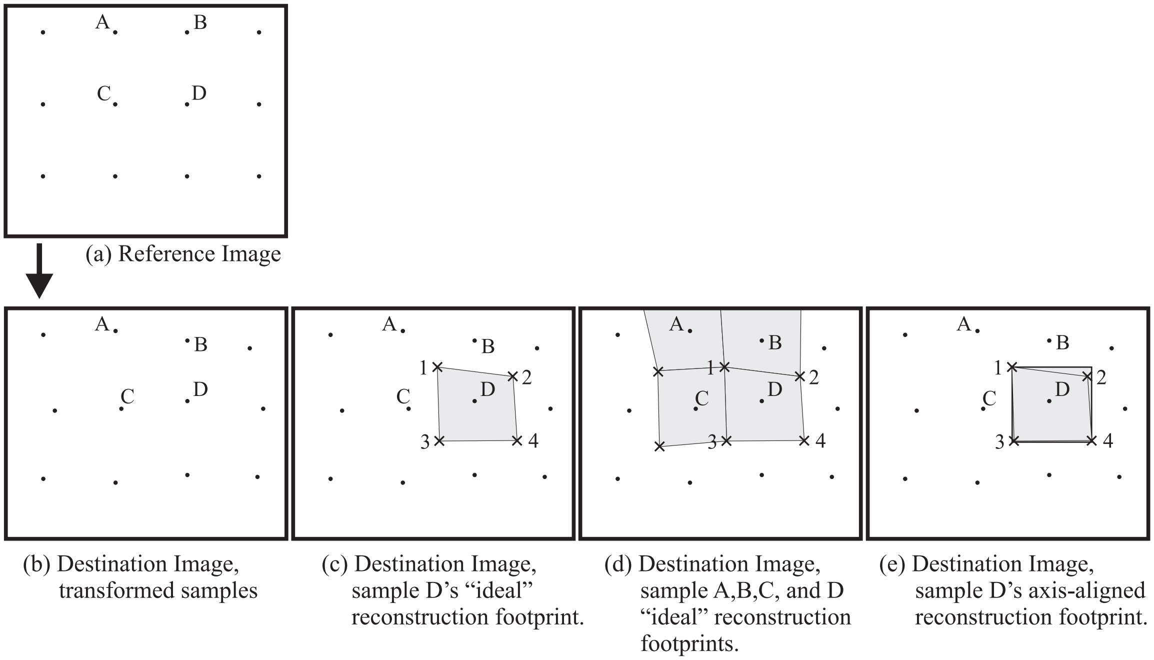

A
B
C
D
(a) Reference Image
A
B
C
D
1
2
3
4
(b) Destination Image, transformed samples
(c) Destination Image, sample D's "ideal" reconstruction footprint.
(d) Destination Image, sample A,B,C, and D "ideal" reconstruction footprints.
(e) Destination Image, sample D's axis-aligned reconstruction footprint.

Figure 4.12: *Reconstruction technique used in conjunction with super-sampled anti-aliasing. Part (a) shows the reference image, and parts (b) through (e) show how the reconstruction footprint is formed using the transformed reference-image samples.*

If the corner of a reconstruction footprint lies on the edge of a surface, the above algorithm is modified. A footprint corner is considered to lie on an edge when the four reference-image samples surrounding the corner belong to more than one surface. The determination as to whether or not two samples belong to the same surface is made using the discontinuity-detection algorithm discussed earlier. When a footprint corner is on an edge, the algorithm computes the corner's location using *only* information from the sample to which the footprint belongs. More specifically, the corner's location is computed using the edge-splat computation discussed in the previous subsection.

The algorithm uses the splat computation instead of the averaging computation to compute a footprint corner's location in one other instance. This other instance is when all four samples surrounding a corner have passed the discontinuity test (i.e. are considered to belong to the same surface), but fall in a *fold-over* configuration with respect to each other. Figure 4.13 illustrates this case, which usually occurs at an occlusion boundary where the front object has moved over the rear object, but not far enough to trigger a failure of the discontinuity test. In a fold-over configuration, calculating the footprint corner by averaging the transformed locations of the four samples that were adjacent in the source image would result in a misshapen footprint. Such a misshapen footprint might not even cover the transformed position of its own sample, which is clearly an undesirable result.

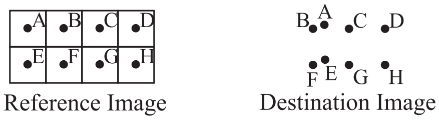


Figure 4.13: *A fold-over configuration. Samples A and E are on the "wrong" side of samples B and F.*

The reconstruction algorithm detects a fold-over configuration by examining the quadrilateral formed by the transformed positions of the four samples surrounding the corner in the source image. The vertex connectivity of this quadrilateral is defined by the relationship between the samples in the *source* image. In the normal case, this quadrilateral is front-facing and convex. In the fold-over case it will be either non-convex or back-facing. The convexity/back-facing test that I use is adapted from a Graphics Gem [Hill94].

The conversion of the footprint from an arbitrary quadrilateral to an axis-aligned bounding box (Figure 4.12e) can result in a slight overlap of adjacent reconstruction footprints. Because rotations about the view direction in a fraction of a second are generally very small, this overlap is minimal in practice—it is rare for the center of a destination pixel to fall in the overlap region. I have not observed any artifacts caused by this occasional overlap. For PRW, the tradeoff is worthwhile, since it is much cheaper to rasterize an axis-aligned rectangle than it is to rasterize an arbitrary quadrilateral. In other 3D warping applications with more rotation about the view direction, it might be necessary to rasterize the general quadrilaterals.

4.3.3 Compositing

In my ideal approach to reconstruction and resampling, each 3D surface is independently reconstructed. Then the 3D surfaces are projected into 2D destination-image space, and resampled on destination-image pixel centers to produce candidate pixels. Different 3D surfaces can generate samples at the same destination-image pixel, so a compositing step arbitrates between the contending candidate pixels to determine the displayed pixel. In this ideal approach to reconstruction, the compositing step consists of Z-buffering, and possibly alpha blending at the edges of surfaces.

In my more practical approach to reconstruction and resampling, the surface reconstruction is performed independently in each source image. If the same surface is represented in both reference images, then the compositing step must arbitrate between candidate pixels that come from different reference images but represent the same 3D surface. As in the ideal approach to reconstruction, the compositing step must also arbitrate between candidate pixels that represent different 3D surfaces.

So, the compositing algorithm must achieve two goals. First, it must arbitrate between candidate pixels that are at different depths (i.e., from different surfaces). This goal is achieved by Z-buffering. Second, the algorithm must arbitrate between candidate pixels that are at the *same* (or almost same) depth. Such a pair of candidate pixels represents an instance in which the same surface is visible in both reference images. The compositing algorithm must determine which candidate pixel better represents the surface.

My PRW system resolves visibility by Z-buffering, rather than by using McMillan's occlusion-compatible traversal [McMillan95a], because McMillan's traversal can only resolve occlusion relationships within a single reference image. My system must resolve occlusion relationships between two or more reference images.

My PRW system performs its compositing incrementally, as part of the warping and reconstruction process. In this sense, the compositing is very much like an enhanced Z-buffer algorithm. As candidate pixels are produced by the reconstruction and resampling algorithm, they are composited with candidate pixels already stored in the destination image.

When a reference-image pixel is transformed, the warper computes a bit-mask specifying which other reference images ought to sample the same surface better. This determination is made based on the surface orientation information carried with each reference-image pixel (as I will describe in the next subsection). If the surface would be less oblique in another reference image, then that image's bit is set in the mask. The algorithm for setting the bit-mask could also consider other criteria, although the current algorithm does not. One possible additional criterion is the ratio between different reference-to-destination viewpoint distances.

During compositing, the bit-mask is used to arbitrate between candidate pixels with similar destination-image 1/Z values. The source-image number associated with a candidate pixel is stored with it in the destination image. To arbitrate between a new candidate pixel and the candidate pixel already stored in the destination image, the compositor checks the bit in the new candidate pixel's bit-mask that corresponds to the already-stored pixel's source-image number. [2]

For most surfaces in a PRW system, the sampling density changes very little from one reference image to another. The reason is that the two source-image viewpoints and the destination-image viewpoint are generally all close to each other. Thus, for most surfaces it is unnecessary to explicitly arbitrate between candidate pixels that represent the same surface, as I do—it would be sufficient to randomly choose one of the candidate pixels. However, there is one exception to this statement. If the surface in question is shaded using highly view-dependent lighting (e.g. highly specular phong lighting), then the two contending samples may have different colors. Randomly choosing one or the other leads to a disturbing time-varying speckling of the surface. It is better to consistently choose one reference image or the other to determine the displayed color of the surface.

In systems that display acquired imagery, the sampling question becomes much more important. In such a system, the reference-image viewpoints are more widely dispersed in space. Some members of our research group (David McAllister *et al.*) encountered problems with surface sampling in their system for warping acquired imagery, until they implemented a technique to choose the better-sampled candidate pixel. In order to implement such a selection on PixelFlow [Molnar92], they use the low-order bits of the "Z" value to represent the sampling density of a surface [McAllister99]. Thus, in their system the Z-buffer test simultaneously compares depth and sampling density, at the cost of some loss in the depth precision of the Z-buffer.

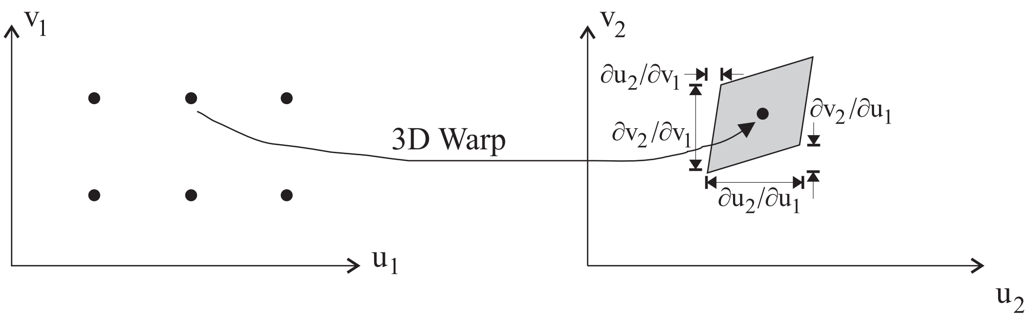
If a surface has some view-dependence to its shading, then the boundary at which there is a change in the best-sampling reference image may be visible. I have not found this to be a problem in my PRW systems (although I believe that such cases could be constructed), but it is a problem in systems that render acquired imagery. The visual impact of the transition can be reduced by changing from a winner-take-all algorithm to a blending algorithm. A blending algorithm weights the contributions from different reference images based on their sampling density. The image-based rendering systems described in [Pulli97] and [Darsa97] both use such a blending algorithm.

[2] My current implementation stores the *bit-mask* with the candidate pixel in the destination image. In Appendix C this bit-mask is labeled "*BetterMask*." This implementation requires slightly more per-pixel storage than storing the source-image number.

4.3.4 Splat-size and sample-area computations

In this subsection, I describe in detail two of the computations used in my reconstruction algorithms. The first is the edge splat computation, which was introduced in section 4.3.2. The second is the sample-area computation, which was introduced in section 4.3.3.

Figure 4.14 illustrates how a square pixel in the source image maps to a parallelogram in the destination image. This mapping makes a first-order approximation, which is equivalent to assuming that the surface being warped is locally planar and that there is negligible perspective foreshortening over the extent of the splat. Thus, if a planar surface is warped using this form of reconstruction (and if the source-image pixels are sufficiently small that perspective foreshortening can be ignored), there will be no gaps or overlaps in the destination image—the destination-image quadrilaterals will perfectly tile the destination image.



v_1
u_1
3D Warp
v_2
$\partial u_2/\partial v_1$
$\partial v_2/\partial v_1$
$\partial v_2/\partial u_1$
$\partial u_2/\partial u_1$
u_2

Figure 4.14: *3D warping splat geometry. If we assume that the surface is planar, then we can compute the destination-image surface element that corresponds to a pixel in the source image. The computation requires information about the surface orientation in the source image. This information is in the form of partial derivatives of source-image depth with respect to u_1 and v_1.*

The 3D warp maps the source-image pixel's center from (u_1, v_1) to (u_2, v_2), so (u_2, v_2) is the center of the destination-image parallelogram. The four corners of the parallelogram (the splat) are:

$$\text{corners} = \left(u_2 \pm \frac{1}{2}\left(\frac{\partial u_2}{\partial u_1} + \frac{\partial u_2}{\partial v_1}\right) \quad , \quad v_2 \pm \frac{1}{2}\left(\frac{\partial v_2}{\partial u_1} + \frac{\partial v_2}{\partial v_1}\right)\right) \tag{4.1}$$

The expressions for the necessary partial derivatives can be computed from the 3D warp equations (Equations 1.12). For convenience, I restate these equations here:

$$\begin{aligned} u_2 &= \frac{w_{11}u_1 + w_{12}v_1 + w_{13} + w_{14}\delta(\bar{u}_1)}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta(\bar{u}_1)} \\ v_2 &= \frac{w_{21}u_1 + w_{22}v_1 + w_{23} + w_{24}\delta(\bar{u}_1)}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta(\bar{u}_1)} \end{aligned} \tag{1.12}$$

Then, the partial derivatives are:

$$\frac{\partial u_2}{\partial u_1} = \frac{(\partial\delta/\partial u_1)(w_{14} - w_{34}u_2) + w_{11} - w_{31}u_2}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta} \tag{4.2}$$

$$\frac{\partial u_2}{\partial v_1} = \frac{(\partial\delta/\partial v_1)(w_{14} - w_{34}u_2) + w_{12} - w_{32}u_2}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta} \tag{4.3}$$

$$\frac{\partial v_2}{\partial u_1} = \frac{(\partial\delta/\partial u_1)(w_{24} - w_{34}v_2) + w_{21} - w_{31}v_2}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta} \tag{4.4}$$

$$\frac{\partial v_2}{\partial v_1} = \frac{(\partial\delta/\partial v_1)(w_{24} - w_{34}v_2) + w_{22} - w_{32}v_2}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta}. \tag{4.5}$$

As with the 3D warp's transform equations, these computations can be performed incrementally to save much of the computation cost. Typically, the size of the splat is capped to prevent unreasonably-large-sized splats from being generated by source-image surfaces that are extremely oblique.

Since $\delta = \frac{S}{z}$, if $S = 1$, the partial derivatives of δ with respect to u_1 and v_1 are equal to the partial derivatives of $1/Z$ with respect to source-image u_1 and v_1. The partial derivatives with respect to $1/Z$ represent surface orientation, in much the same way that a normal vector does. However, the partial-derivative representation is superior to a normal-vector representation for two reasons. First, I have found that using the partial derivatives of $1/Z$ results in simpler computations than a normal-vector representation. Second, the partial derivatives of $1/Z$ are readily available, since these values are already maintained internally in almost all conventional polygon renderers to interpolate $1/Z$. The only modification necessary is to store them in the framebuffer with each pixel.

The area of the splat described by Equation 4.1 can be computed using a cross product:

$$\mathit{Area} = \left\| \left(\frac{\partial u_2}{\partial u_1}, \frac{\partial v_2}{\partial u_1}, 0 \right) \times \left(\frac{\partial u_2}{\partial v_1}, \frac{\partial v_2}{\partial v_1}, 0 \right) \right\| \tag{4.6}$$

Simplifying,

$$\mathit{Area}' = \frac{\partial u_2}{\partial u_1} \cdot \frac{\partial v_2}{\partial v_1} - \frac{\partial u_2}{\partial v_1} \cdot \frac{\partial v_2}{\partial u_1}, \tag{4.7}$$

where the sign of *Area*′ indicates whether the splat is front-facing or back-facing. When two candidate pixels (from different source images) represent the same 3D surface, this area computation is used to decide which candidate pixel samples the surface best, as described earlier in this chapter.

4.3.5 Over-sampling

In my discussions about sampling density, I have been primarily concerned with the problem of under-sampling. It is also possible for over-sampling to occur, when surfaces are at a more oblique angle in the destination image than in the reference image. In this case, some samples will be effectively discarded in the reconstruction process, as more than one sample maps to the same destination-image pixel (more precisely, some interpolating triangles never overlap a destination-image pixel center). In most circumstances, this is roughly equivalent to having sampled at a lower rate to begin with—it is what you would get by conventionally rendering from the destination-image viewpoint. However, when the reference-image samples are pre-filtered (i.e. they come from rendering MIP-mapped textures), discarding samples could introduce artifacts that would otherwise be preventable by the pre-filtering. I have not observed any artifacts of this type, and believe that they would be less serious than those caused using isotropic rather than anisotropic texture filtering. Possible solutions to this problem include slightly increasing the super-sampling in the destination image; modifying the conventional renderer's algorithm for choosing MIP-map levels; and modifying the warper's compositing algorithm to discriminate against excessive over-sampling as well as excessive under-sampling.

4.3.6 Moving objects and highly view-dependent lighting

The reconstruction strategy that I have pursued implicitly assumes that objects in the scene are in the same 3-space location in all reference images in which they appear. If an object changes position from one reference image to another, then it will be interpreted as two distinct objects. As a result, it will appear twice in the displayed image.

Because my PRW system generates each reference frame at a different time, any objects that are moving or deforming with time will be incorrectly reconstructed. Thus, PRW is restricted to static scenes. Any moving objects must be directly rendered into the displayed frame in a separate per-displayed-frame rendering pass. I believe that alternative techniques, such as associating motion information with reference-frame pixels [Costella93], are impractical. These techniques would require extensive integration with the scene-graph level of the application.

My reconstruction strategy also implicitly assumes that surfaces in the scene are lit in a view-independent manner. Mild specularity is acceptable in practice, but highly view-dependent lighting will not be correctly reconstructed (especially for flat or low-curvature surfaces, where highlights move more quickly). In a PRW system, any view-dependent lighting on a surface will change at the reference-frame rate, not the displayed-frame rate. Thus, effects such as reflection maps will appear to jump around at the reference-frame rate. This problem could conceivably be overcome by using deferred shading [Whitted82, Deering88]. Max has already implemented deferred shading in a 3D warper [Max95]. However, I believe that the current trend towards increasingly complex shading (with corresponding increases in the number of shading parameters) makes the practicality of deferred shading questionable.

4.4 Alternative approaches

In the previous few sections, I described my preferred approach to PRW reconstruction. In this section, I describe several alternative approaches, and compare them to my preferred approach. I have implemented some, but not all, of these alternative approaches in one or more of my PRW systems. Some of these alternative approaches have been implemented by other researchers as well. This section is reasonably detailed, because I feel that it contributes an important understanding of the tradeoffs involved in choosing a reconstruction technique for 3D warping.

4.4.1 Fixed-sized splats

One alternative approach to reconstruction and resampling is the use of what I refer to as *fixed-sized splats*. In this approach, surface segmentation is never explicitly performed. Instead, each reference-image pixel is independently transformed to the destination image, and its color is written to one or more destination-image pixels. The approach is similar to Westover's splatting algorithm for volume rendering [Westover90]. Figure 4.15 illustrates the fixed-size splat approach. Variations of this approach have been used by several researchers, including [Chen93, Rafferty98, Shade98].

The simplest variant of the fixed-size splat approach is to write a single destination-image pixel for each source-image pixel that is warped. As I stated at the beginning of the chapter, this approach usually leaves pinholes in the destination image. When warping only one reference image, the results are completely unacceptable (Figure 4.16a). Surfaces that are under-sampled contain pinholes.[3] However, the results improve somewhat when two reference images are warped (Figure 4.16b), and

[3] Such under-sampling can occur for two reasons: First, the surface can be more oblique or distant in the source image than in the destination image due to viewpoint translation. Second, the sampling density can change due to view rotation, because a planar image does not sample uniformly in angular coordinates ($\tan(\theta) \neq \theta$).

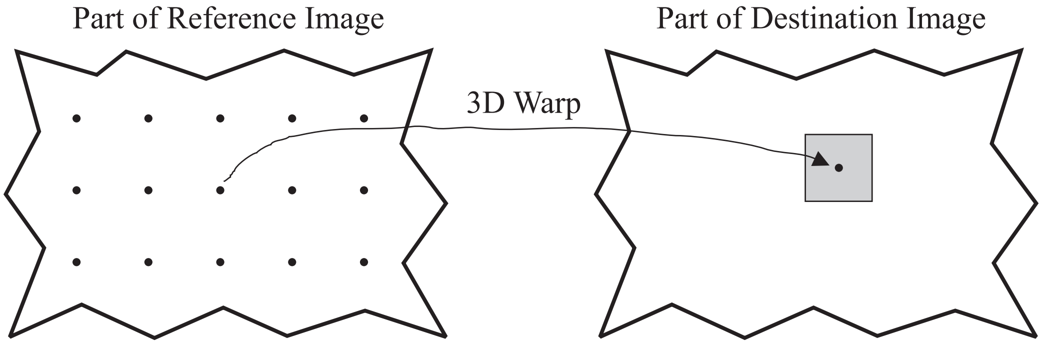
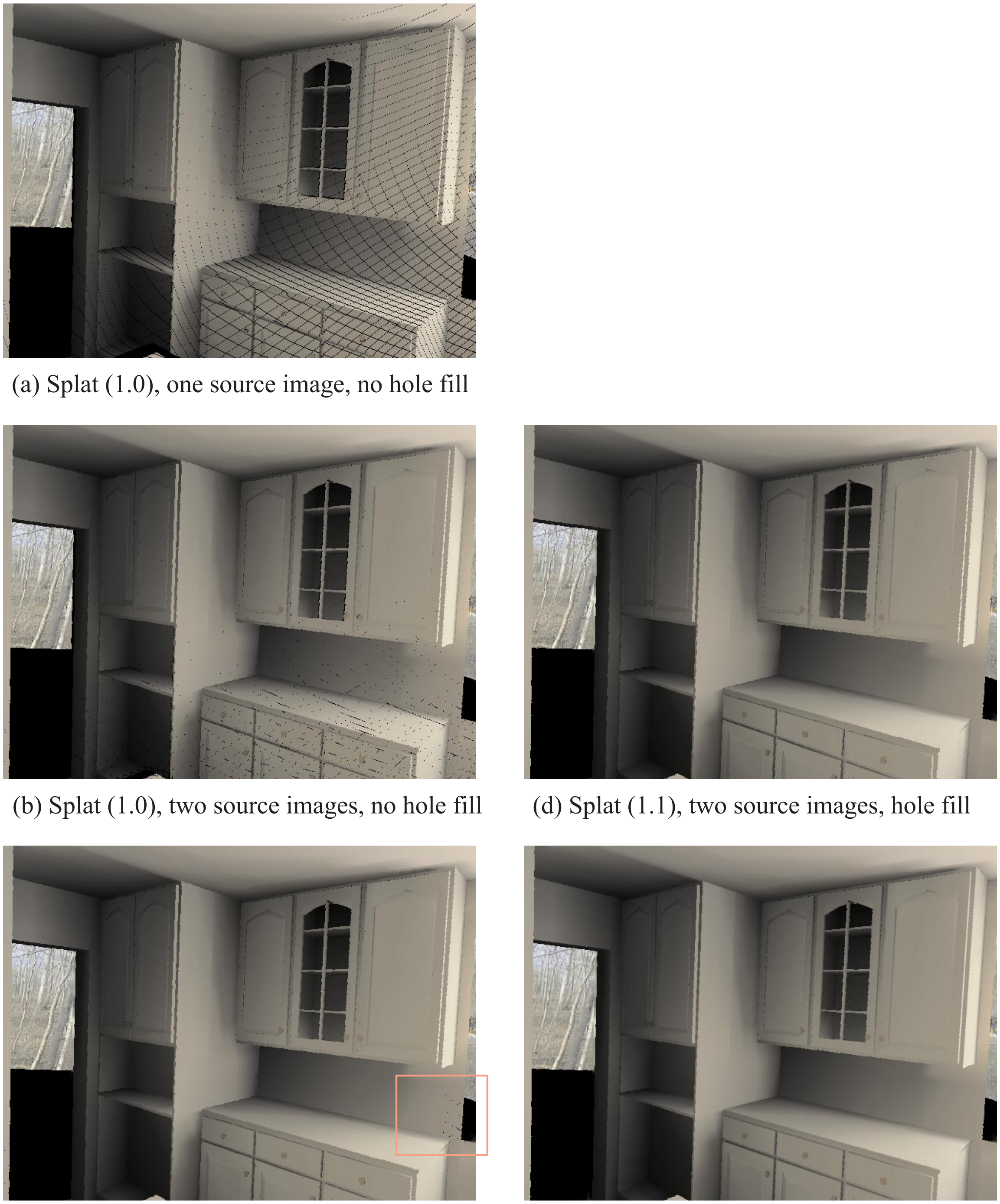


Figure 4.15: *Fixed-size splat approach to reconstruction. Each reference-image sample is transformed to the destination image. Then, the warper fills a fixed-size axis-aligned region surrounding the transformed location with the sample's color.*

improve further when my hole-filling algorithm is used (Figure 4.16c). The reason is that pinholes from one of the source images tend to get filled in by the other source image, or by the hole filling algorithm. Despite this improvement, some pinholes remain. These pinholes are typically caused by a rear surface that "shows through" the pinholes in a front surface. Because the rear surface has already filled the pinholes, the hole-filling algorithm does not get a chance to fill them.

These remaining pinholes can be eliminated by using slightly oversized splats. It is possible to do so by writing 2 pixel by 2 pixel or 3 pixel by 3 pixel splats, as [Rafferty98] does. But, the larger splats cause problems, including expansion of foreground edges and effective shifting of the contents of the entire destination image by one or more pixels from the correct location. I have experimented with a non-integer splat size of 1.1 pixels by 1.1 pixels, which minimizes these effects. For the small viewpoint changes of PRW, a 1.1 x 1.1 splat is large enough to eliminate most pinholes. The non-integer splat is implemented by filling all destination-image pixels whose centers lie within a 1.1 pixel by 1.1 pixel axis-aligned rectangle. The location of the transformed reference-image pixel defines the center of the rectangle. Figure 4.16d illustrates the results obtained from this approach.

Although the 1.1 x 1.1 pixel splat eliminates pinholes, it is still somewhat inferior to the hybrid mesh/splat warp, which I described in section 4.3. Note, for example, the difference between the quality of the cabinet edges between Figures 4.16d and 4.16e. The interpolation performed by the hybrid warp reduces the severity of the resampling artifacts generated by the 3D warp. Figure 4.17 illustrates this difference using a zoomed-in view of images generated by the two algorithms. The 1.1 x 1.1 pixel splat is especially prone to artifacts at edges, because the over-sized splats can confuse the compositing algorithm. The compositing algorithm is designed to cope with multiple candidate pixels describing the same surface that originate from *different* source images. However, it is not designed to cope



(a) Splat (1.0), one source image, no hole fill

(b) Splat (1.0), two source images, no hole fill

(d) Splat (1.1), two source images, hole fill

(c) Splat (1.0), two source images, hole fill

(e) Hybrid warp, two source images, hole fill

Figure 4.16: *Evaluation of warping using fixed-size splats. All of these images are from frame #309 of the kitchen sequence, at 640x480 resolution with no anti-aliasing. (a) shows an image produced using 1.0 x 1.0 splats, with only a single source image, and no hole filling. (b) shows that using a second source image fills in many of the pinholes. (c) shows that most of the remaining pinholes are eliminated by the hole filling algorithm, but a few remain in the region outlined by the light-red box. (d) shows that slightly larger 1.1 x 1.1 splats eliminate these pinholes. For comparison purposes, (e) shows the results of my preferred hybrid mesh/splat warp, which was discussed in section 4.3.*

with multiple candidate pixels describing the same surface that originate from the *same* source image. Such candidate pixels are produced (at approximately every 5th pixel) by the 1.1 x 1.1 pixel splats. The compositing algorithm makes what is essentially an arbitrary choice between two such competing pixels.

There are two other important drawbacks to the fixed-size splat algorithm. First, it is only appropriate when there is an approximately 1-to-1 mapping from source-image pixels to destination-image pixels. If there are many destination-image pixels per source-image pixel, then the splat size must be increased accordingly, and the resulting image looks blocky. In contrast, the hybrid mesh/splat algorithm will properly interpolate between samples, so that the image looks blurred rather than blocky.

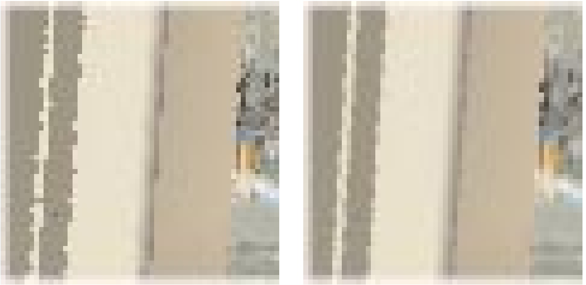
The second drawback to the fixed-size splat algorithm is that it becomes less appropriate as the distance between source and destination viewpoints increases. To avoid pinholes, the splat size must be increased, which increases the severity of the technique's artifacts.

Despite its drawbacks, the 1.1 x 1.1 fixed-size splat algorithm can provide acceptable quality images for PRW. For a software-only PRW system, its simplicity probably makes it the algorithm of choice, as long as it is used in conjunction with a hole-filling algorithm.

4.4.2 Splats computed from normal vector

There is a variant of the splat approach in which the splat size varies for each reference-image pixel. The splat size is computed using the normal vector of the reference-image pixel. More precisely, the partial derivatives of $\frac{1}{z}$, which provide information equivalent to that provided by a normal vector, are used to compute the splat size. Equation 4.1 described the computation, which is identical to that used for the edge splats in my hybrid mesh/splat approach. My remote display system, which is described in more detail in Chapter 6, was the first system to use this normal-vector splat for 3D warping. More recently, Shade *et al.*'s LDI warper [Shade98] has used a similar approach to determining splat size.

For planar surfaces, the normal-vector approach works better than the fixed-size–splat approach. For under-sampled planar surfaces, the splats grow enough to eliminate holes. For adequately-sampled planar surfaces the splats remain small, thus avoiding the introduction of unnecessary artifacts. However, on sharply curving surfaces and at corners the approach does not work as well. In particular, gaps are likely to appear at corners. By slightly over-sizing the splats, these gaps can be reduced or eliminated, but the usual artifacts associated with over-sized splats are also produced.



(a) Splat (1.1) (b) Hybrid warp

Figure 4.17: *Zoomed-in comparison of mesh/splat hybrid warp to fixed-size-splat warp. Supersampled anti-aliasing was not performed when producing these images.*

From a theoretical point of view, normals do not eliminate the basic problem that the PRW system has inadequate information available to perfectly reconstruct the scene. The normals do provide some additional information—the information is equivalent to providing the first derivatives of depth. In 1D Fourier reconstruction, the Nyquist frequency is doubled if first-derivative values are available at all sample points [Jerri77]. In 2D Fourier reconstruction, the Nyquist frequency is increased (but not quite doubled) by providing first-derivative values. However, doubling the Nyquist frequency does not solve the basic problem that the scene itself is insufficiently constrained for *any* sampling rate to be adequate for perfect reconstruction.

4.4.3 Conditional mesh

In another approach that I have explored [Mark97b], the warper uses each reference image to define a triangle mesh, but some mesh triangles are treated specially. I refer to this approach as the *conditional mesh* approach to 3D warp reconstruction. Triangles which cross a discontinuity in the reference image are considered to represent false surfaces and are identified as *low connectedness* triangles. The discontinuities in the reference image are identified by a surface segmentation algorithm, such as the one discussed in section 4.3 (although [Mark97b] actually uses a somewhat different surface-segmentation algorithm). Triangles for which all three vertices belong to the same 3D surface are considered to be *high connectedness* triangles.

In the hybrid mesh/splat approach that I discussed earlier, low connectedness triangles were discarded in the reconstruction process. In the conditional mesh approach, these triangles are rasterized into the destination image. However, the compositing process is modified so that a candidate pixel from a low-connectedness triangle always loses to a candidate pixel from a high-connectedness triangle.

Thus, a false “surface” will not occlude a true surface. The Z values of candidate pixels are only used to arbitrate between two candidate pixels that both originate from high-connectedness triangles.

This algorithm does not use a separate hole-filling algorithm, because the low-confidence triangle rasterization substitutes for the hole-filling algorithm. The coloring of the low-confidence triangles is modified to fulfill this purpose. Instead of coloring the triangle by interpolating all three vertex colors (colors of the transformed source-image pixels), the triangle is colored with the color of the furthest-away vertex. Figure 4.18 illustrates this technique. This approach to hole filling produces stripe artifacts similar to those generated by the non-blurring version of the hole-filling algorithm described in Chapter 3.

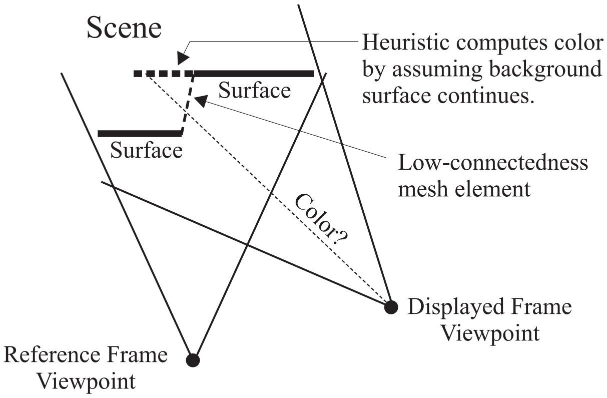

Scene
Heuristic computes color by assuming background surface continues.
Surface
Surface
Low-connectedness mesh element
Color?
Displayed Frame Viewpoint
Reference Frame Viewpoint

Figure 4.18: *In the conditional mesh approach, low-connectedness mesh triangles are flat shaded with the color of the vertex that is furthest away.*

The major advantage of the conditional-mesh reconstruction algorithm as compared to most other algorithms is that it does not require separate hole-filling pass(es) over the destination image. Because all of the writes to the destination image are defined in terms of triangle rasterizations, the algorithm can implemented using existing triangle rasterizers, if the extra complexity in the compositing step can be accommodated.

The conditional-mesh algorithm also has a number of substantial disadvantages as compared to other algorithms. First, the computation required by the conditional-mesh algorithm is not bounded by the image resolution, as it is for most other algorithms. The reason is that the low-connectedness triangles can be arbitrarily large, as can their depth complexity. This problem occurs when there are rapidly alternating depths in the reference image. The arbitrarily-large triangles also require a very general triangle rasterizer. In contrast, techniques that use bounded-size triangles can used a fixed-footprint hardware triangle rasterizer.

The quality of images produced by the conditional-mesh algorithm is also inferior. Because the algorithm does not use any form of edge splats (as defined in section 4.3, one-half of a pixel is shaved off of the edge of all foreground surfaces, and one-pixel-wide features disappear completely. The hole-filling results are poorer as well. They are inferior to those obtained using a separate pass, because the conditional-mesh algorithm does not perform any blurring as it fills holes.

4.4.4 Partially transparent splats

The hybrid mesh/splat warp that I described in section 4.3 could be enhanced by performing anti-aliasing at the edges between foreground and background surfaces. As described, the algorithm uses completely opaque edge splats. Instead, the algorithm could use edge splats with a gradual falloff from 100% opacity to 0% opacity as the distance from the transformed sample point increases. This use of partial transparency would properly indicate the uncertainty in the location of the surface edge.

Rendering of partially transparent pixels always complicates reconstruction, and this situation is no different. A correct implementation would require one of the usual solutions, such as in-order rendering or the use of an A-buffer [Carpenter84]. I decided that the additional complexity required by this approach did not justify the slight quality improvement. However, in a system for warping acquired imagery, the source-image sampling density can be lower, and thus the region of uncertainty at an edge can be larger. In such a system, the benefit from edge anti-aliasing might be large enough to justify the expense.

The layered-depth-image warper described in [Shade98] performs all of its reconstruction using partially transparent splats. Because LDI's can be warped using an occlusion-compatible traversal, this LDI warper easily satisfies the in-order rendering requirement for partial transparency. Thus, Shade *et al.*'s warper correctly performs foreground/background edge anti-aliasing.

However, the use of partially transparent splats is not as attractive as it seems. When these splats are used for the *interior* regions of surfaces, they do not always produce the correct result. In particular, their use can cause background color to incorrectly bleed through a fully opaque foreground surface. Consider, for example, a destination-image pixel covered by two foreground splats, where each foreground splat is 50% transparent at this pixel. When the first splat is rendered, the destination-pixel color becomes a blend of 50% (previously-stored) background color and 50% foreground color. After the second splat is rendered, the pixel color becomes 25% background color and 75% foreground color. It should be 100% foreground color. The problem could be cured by using an A-buffer for reconstruction, so that the reconstruction for each surface is performed independently.

There is an additional problem with using partially transparent splats for interior regions of surfaces: It may be difficult to insure that the foreground-surface opacity always sums to at least 100%, unless the splats are considerably over-sized. In an ideal system, the splat's shape and size are perfectly adapted to the local sample density and geometry. When this ideal is reached, the partially-transparent-splat approach produces results identical to those produced by interpolating between transformed samples.

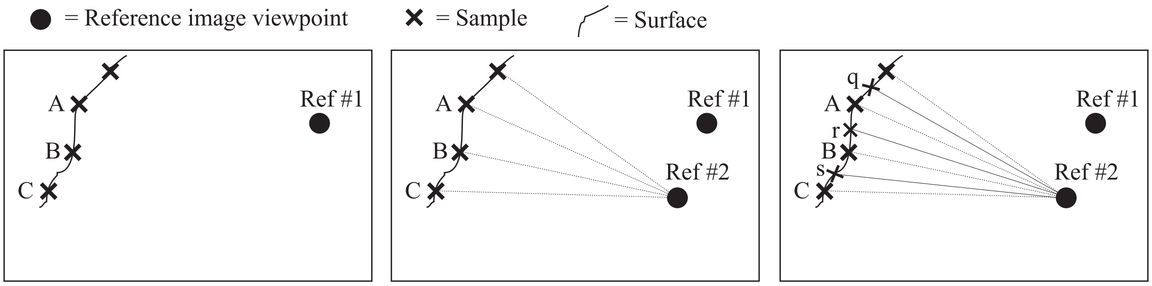
4.4.5 Back-projection to other images

My preferred approach to reconstruction and resampling works with one image at a time. It is only in the compositing step that information from different images is compared and combined. When that step is reached, the implicit 3D reconstruction and resampling has already been completed.

By using information from other reference images as each reference image is warped, the reconstruction and resampling can potentially be improved. There are two disadvantages to this approach: It is more complicated, and it requires simultaneous access to all reference images, which is undesirable in some types of hardware warpers. For these reasons, I avoided this approach, but I will briefly describe it now.

I refer to this approach as the *back-projection* approach. Reference images are still warped one at a time, but with a modification. When a reference-image pixel is warped, its 3-space location is projected back to the other reference-image viewpoint(s) (see Figure 4.19). For each of the other reference image(s), this back-projection determines the correct (u, v) location for the pixel's surface element in the other reference image. The warper then examines the depth at this (u, v) location (in practice, it examines the depth at the grid points surrounding the location). From this depth, the warper determines whether or not the surface is occluded in the other reference image. If it is not occluded, the additional information provided by the other reference image can be used in the reconstruction process. For example, this additional information typically increases the effective sampling density of the surface.

Back-projection can also be used to perform surface segmentation. If the "surface" connecting two samples in the reference image being warped is pierced by a ray from another reference image, then the surface does not in fact exist (Figure 4.20). If all potentially visible surfaces are properly



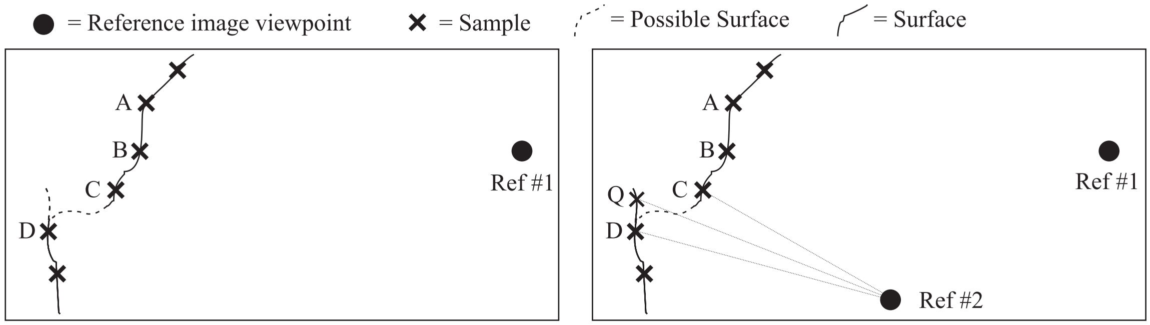
(a) Surface as seen from reference image #1

(b) Sample points are back-projected into reference image #2.

(c) Surface samples from reference image #2 are obtained at the image locations identified by back-projection.

Figure 4.19: *Back-projection can be used to locate a particular surface's samples in a second reference image.*

represented in one of the reference images, then this technique can be used as the sole means of surface segmentation, replacing the technique described earlier in this chapter. However, in a PRW system, there will almost always be some visibility holes. For example, such a situation would occur in Figure 4.20, if reference image #2's view of the region of interest was blocked by an occluder—the gap between samples C and D would be a visibility hole with no rays from image #2 reaching it. Thus, the back-projection surface-segmentation technique would not detect this hole, and the hole-filling algorithm described in Chapter 3 would not be invoked. In fact, with the back-projection surface-segmentation algorithm, the hole-filling algorithm would *never* be invoked. For this reason, in a PRW system the back-projection approach to surface segmentation must be augmented with an algorithm such as the one described earlier in this chapter.



(a) First reference image only

(b) First reference image, with back-projection to second reference image.

Figure 4.20: *Using back-projection for surface segmentation (in flatland). In (a), it is unclear whether or not samples C and D lie on the same surface. In (b), back-projection to a second reference image is used to resolve the ambiguity. Samples C and D are projected into reference image #2, and a new sample Q is detected that lies in between them in the image plane. Because this sample is farther away, there can not be a valid surface connecting samples C and D.*

4.4.6 Inverse warping

Laveau and Faugeras [Laveau94] and McMillan [McMillan97] have developed inverse-mapped approaches to 3D warping. These approaches are analogous to the ray-tracing approach to polygonal rendering: For each destination-image pixel, a search is made in the reference image(s) to find the appropriate data. Because of the constrained geometry of the 3D warp, the search is confined to a line segment in each reference image. The inverse-mapped approach has the advantage that it readily allows the information from all source images to be considered simultaneously. However, the search that is required in the source images is expensive, especially since PRW does not provide the opportunity to create auxiliary data structures (e.g. pyramidal image representations). In a recent system for warping a single source image, Shade *et al.* [Shade98] avoid this search by first performing a forward warp of depth values, then using an inverse warp to determine color values. I chose not to pursue the inverse-mapped approach in detail, because I believe that it is too computationally expensive for PRW.

4.5 Previous work

Although I have cited some previous work throughout this chapter, I have not yet discussed in detail some of the most important work that came before mine and helped to guide it. I discuss this previous work in this section. By deferring the discussion until now, I can explain how previous approaches to the reconstruction problem fit into the conceptual framework that I have used in this chapter.

Chen and Williams [Chen93] recognized that writing a single pixel for each transformed point produces pinholes in the warped image. They proposed two solutions. Their first proposed solution is to treat the reference image as a quadrilateral mesh—the system rasterizes the destination-image quadrilaterals formed by the transformed sample points. Thus, colors are interpolated in destination-image space. Since the mesh is unconditional (in contrast to the conditional mesh that I discussed earlier), it can not be used with multiple reference images. The unconditional mesh is one of two techniques that McMillan implemented in his warping system [McMillan97].

Greg Ward's *pinterp* program [Ward90] uses a modification of this mesh strategy, in which a quadrilateral is only rasterized if its samples' Z-values are similar to each other. In terms of my conceptual framework, Ward's technique can be thought of as implicitly segmenting the image into different surfaces based on a Z-difference criterion.

Chen and Williams' second proposed solution to the pinhole problem is to detect and fill the uncolored pixels in a post-processing step. These uncolored pixels are filled by interpolating the colors of nearby colored pixels. Chen and Williams observed that some pinholes will be erroneously filled by background surfaces before the post-processing step can do its work. This problem is a manifestation of the fact that each 2D surface residing in 3-space should be independently reconstructed.

McMillan's second reconstruction technique assumes that all reference-image samples represent circular discs in space. The normal vector of the discs is oriented in the $\pm z$ direction in the reference-image coordinate system. These discs project to ellipses in the destination-image coordinate system. This approach is similar to the splat-based approach that I described earlier, except that McMillan's approach assumes a fixed surface orientation and uses discs instead of quadrilaterals.

McMillan recognized that both the disc-based reconstruction technique and the unconditional-mesh reconstruction technique can produce undesirable artifacts. He describes several types of these artifacts in [McMillan97].

4.5.1 Points as primitives

Levoy and Whitted [Levoy85] developed a system to use points as the rendering system's fundamental display primitive. In their system, the points are generated by routines that convert geometric primitives to arbitrarily-located 3D points. Each portion of a surface is represented exactly once in the point set, an important difference from my PRW system. In my PRW system, a surface can be represented more than once (visible in multiple reference images), or not at all (occluded in all of the reference images). Levoy and Whitted's system is also different from my system in that it supports partial transparency of surfaces.

Levoy and Whitted's system faces a reconstruction and resampling problem similar to the one that I encountered in PRW, because their system must reconstruct surfaces from points in the final point-rendering step. Their point-display system handles this problem by assuming that all surfaces are continuous and differentiable. Furthermore, the system implicitly assumes that surfaces are locally planar, by using a discrete approximation to the normal vector in order to compute a weight for each point. This weight corresponds to the expected coverage area for the point. For surfaces which are sharply curved with respect to the point density, the computed weight is incorrect. This error manifests itself as an error in the opacity of the surface. For example, an opaque surface can be erroneously considered to be partially transparent. My PRW system avoids this type of error by requiring that all surfaces be fully opaque, and by making a binary decision as to whether or not a surface covers any particular destination-image pixel.

Levoy and Whitted's system uses a windowed-Gaussian filter to compute the contribution of each point to nearby destination-image pixels. For each point, the contribution to nearby pixels is computed. Multiple contributions falling at the same pixel are considered to belong to the same surface if their depths fall within a tolerance value of each other. The tolerance value depends on the orientation of the surface. At each pixel, the color and coverage value for each surface is computed from the one or more contributions originating from that surface. Then, the results from different surfaces are combined using alpha blending. Unlike my PRW system, this technique correctly handles over-sampling.

In work concurrent with mine, Grossman and Dally [Grossman98] have implemented a more complex point-sample-rendering system. Their system uses a pyramidal representation of the destination image to perform reconstruction. The points representing a particular surface are rendered into a level of the pyramid that is sufficiently coarse to guarantee no holes in the surface. A variant of Gortler's push-pull algorithm [Gortler96] is then used to fill holes in the most detailed level of the pyramid with information from the coarse levels. This most-detailed level of the pyramid then becomes the displayed image.

This system uses a preprocessing step to generate its point samples. This preprocessing step attempts to insure that all surfaces in the scene are adequately sampled at some constant resolution. Thus, this approach is not usable for PRW reconstruction, where there is no opportunity for such a preprocessing step.

4.6 Summary

This chapter has discussed the 3D warping reconstruction and resampling problem for multiple source images. I have concentrated on the problem of warping images that consist of point samples, rather than area samples. I argued that reconstruction for the 3D warp should be performed independently for each 2D surface residing in 3-space. In this sense, reconstruction for 3D warping is quite different from the strictly 2D reconstruction typically performed in image processing, and from the strictly 3D reconstruction performed in volume rendering.

To perform independent reconstruction for each 2D surface, the point samples provided by the source images must first be segmented into different surfaces. Since this problem is theoretically impossible for the source images used in post-rendering warping, a heuristic technique is required that typically produces good results (as judged by human perception). I presented such a technique that has reasonable computational cost. The technique independently performs surface segmentation in each reference image using a view-dependent algorithm. I described two reconstruction/resampling algorithms based on this surface-segmentation technique. The first algorithm is designed for general use, and the second is designed to be used in conjunction with super-sampled anti-aliasing. The compositing step used in these algorithms resolves visibility using a modified Z-buffer approach.

Finally, I described several alternative reconstruction and resampling techniques and discussed their strengths and weaknesses using my conceptual framework. One alternative technique, the fixed-size splat, is of particular interest for post-rendering warping. It produces images which are only slightly inferior to those produced by my preferred technique, yet is significantly simpler. It is an appropriate choice for software-based warping systems which would otherwise be unable to achieve their performance objectives.

CHAPTER 5

HARDWARE AND PERFORMANCE ISSUES

The cost of a 3D warp is approximately determined by the number of pixels in the output image, and is therefore almost independent of scene complexity. This fixed cost is one of the major advantages of the 3D warp. However, this fixed cost is high. It is my belief that current CPU's and graphics architectures will not be able to achieve the necessary combination of performance and cost for 3D warping in the next few years. More specifically, I do not believe that current system designs or their derivatives will be able to simultaneously achieve the following three desirable goals for the 3D warp:

1. **High Quality**: 640x480 or higher resolution, anti-aliased, minimal artifacts.

2. **Fast**: 30-70 Hz frame rate.

3. **Cheap**: $1000 or less.

Reaching these goals will require hardware that is designed to support 3D image warping. In this chapter, I discuss how several properties of the 3D warp can be used to design hardware that is optimized for it. However, instead of attempting to present a detailed hardware design, I confine my discussion to characterization and explanation of the relevant design issues. Although the discussion is oriented towards hardware design, much of it is also relevant to the design of efficient software 3D warpers.

The chapter begins with a discussion of the differences between 3D warping and polygon rendering. A hardware design that takes advantage of these differences should be able to provide a better price/performance ratio for 3D warping than polygon rendering hardware can provide. The rest of the chapter consists of several sections that discuss how some of these special properties of the 3D warp can be used to design efficient hardware. The two most lengthly sections discuss fixed-point computation of the 3D warp and memory access patterns of the 3D warp.

5.1 Special properties of the 3D warp

If we expect 3D warping hardware to provide a better price/performance ratio for its task than general polygon rendering hardware can provide, the 3D warp hardware must take advantage of special properties of the 3D warp. The following list summarizes the differences between 3D warping and general polygon rendering:

1. The triangles (or axis-aligned rectanges) rasterized by the warper have a fixed maximum size, and most of them are only one pixel in size. In contrast, there is no maximum size for triangles rasterized by a general polygon renderer. The warper's maximum-size triangles allow the use of a single, small, SIMD-style rasterizer to rasterize all of the triangle's pixels simultaneously. Furthermore, rasterizing small triangles eliminates the need for triangle clipping during setup, since scissoring becomes an efficient solution for partially-on-screen triangles. One disadvantage of small triangles is that there is very little opportunity to use intra-triangle coherence to increase rasterization efficiency. This disadvantage can be reduced by choosing the division of labor between triangle setup and rasterization so that it is optimal for small triangles.

2. A 3D warper does not require texture mapping capability. A general polygon renderer must typically support texture mapping.

3. The 3D warper's triangles come from a reference image. A polygon renderer's triangles are generated by an application. There are a number of important implications of this difference:

   - If the 3D warp's reference image is traversed in a regular manner by the warper, the reference image's epipolar geometry with respect to the displayed image imposes a partial coherence on the memory accesses to the displayed image. In contrast, the displayed-image memory accesses of a polygon renderer are potentially random from polygon to polygon, unless sorting is used. The partial coherence of the 3D warp's memory accesses allows the warper to use a cheap, cache-based memory system. I will discuss this issue in greater detail later in this chapter. The organized memory accesses also allow relatively simple parallelization of the 3D warp (in effect, getting the advantages of a sort-first architecture [Molnar94] without the need to sort).
   - In a 3D warp, the regular organization of the reference image allows incremental evaluation of the transformation equations, reducing the computational cost of the transformations [McMillan95a]. No such incremental evaluation is possible for a polygon renderer.

- The constraints imposed on the 3D warp's transformation by the reference-image organization allow the transformation equations to be implemented using fixed-point arithmetic. A polygon renderer must use more expensive floating-point arithmetic for its transformation equations. I will discuss the fixed-point formulation of the 3D warp transformation later in this chapter.

- The epipolar geometry of the 3D warp allows the warper to use a painter's algorithm to resolve occlusion within a single reference image [McMillan95a]. A general polygon renderer must use a Z-buffer to resolve occlusion, unless sorting is performed. Note that for multiple reference images, this property of the 3D warp is much less useful, since Z-buffering is required to composite the multiple images. Therefore I do not anticipate that PRW systems will take advantage of this property, but I include it here for completeness.

If the 3D warp uses the reconstruction/resampling technique designed for anti-aliasing which I described in Chapter 4, there are two additional differences between general polygon rendering and 3D warping:

4. The warper rasterizes axis-aligned rectangles. The polygon renderer rasterizes arbitrary triangles.

5. The warper uses flat shading. The polygon rendering uses interpolated shading.

Some (but not all) of the advantages of the 3D warp over general polygon rendering also apply to a micro-polygon renderer, such as that used in the REYES system [Cook87]. The advantages numbered 1, 2, and 5 above apply to micro-polygon renderers.

5.2 Fixed-point computation of 3D warp

In this section and its associated appendix, I show that the transformation equations of McMillan's 3D warp can be implemented with fixed-point arithmetic. The values computed from this fixed-point formulation of the 3D warp retain the precision required for post-rendering warping.

The fixed-point formulation of the 3D warp is potentially more efficient to implement than the floating-point formulation. The reason is that fixed-point arithmetic units are simpler and cheaper to implement in hardware than comparable floating-point arithmetic units. So, a hardware implementation of the 3D warp, especially a piplined hardware implementation, can be more cheaply built using fixed-point arithmetic units.

Some software or firmware implementations of the 3D warp can benefit from a fixed-point formulation as well. High-performance SIMD multimedia instructions such as Intel's MMX instructions often accelerate only fixed-point arithmetic operations. On highly-parallel machines with small ALU's, such as UNC's PixelFlow, there is usually not any hardware acceleration of floating point operations. On such machines, fixed-point operations are more efficient.

To begin the analysis leading to a fixed-point formulation of the 3D warp, I restate the 3D warp's transform equations (originally appearing as Equation 1.12).

$$\begin{aligned} u_2 &= \frac{w_{11}u_1 + w_{12}v_1 + w_{13} + w_{14}\delta(u_1, v_1)}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta(u_1, v_1)} \\ v_2 &= \frac{w_{21}u_1 + w_{22}v_1 + w_{23} + w_{24}\delta(u_1, v_1)}{w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta(u_1, v_1)} \end{aligned} \tag{5.1}$$

In this chapter, I use resolution-dependent image coordinates. So, for a 800x500 reference image, u_1 would range between 0 and 799, rather than between 0.0 and 1.0. The w_{ij} coefficients must therefore be defined (via Equation 1.13) using a resolution-dependent $\mathbf{P}_1$ and $\mathbf{P}_2$. Chapter 1 explained the difference between the resolution-independent and resolution-dependent definitions of $\mathbf{P}$ matrices.

Equations 5.1 are of the same form as a general graphics modeling transformation followed by a homogeneous division. But, for a 3D warp, the variables in the equation are restricted to a more limited range of possible values than they are for a general transformation. A general modeling transformation can include arbitrary scaling, translation, and skew, as well as rotation. The 3D warp transformation used in post-rendering warping represents arbitrary rotation, but limited scaling, limited translation, and no skew. These constraints allow us to bound the ranges of intermediate values in the 3D warp equations, so that a fixed-point calculation is possible.

I will break equations 5.1 into three separate sets of equations, to simplify their analysis. The first set of equations maps 2D source-image locations (u_1, v_1) to 3D locations relative to the source image's center of projection. I assume that the source-image plane is rectangular and centered about the view direction.

$$\begin{aligned} x &= s_{u1} \cdot u_1 + o_{u1} \\ y &= s_{v1} \cdot v_1 + o_{v1} \end{aligned} \tag{5.2}$$

The second set of equations transforms points from the source-image's 3D space to the destination image's 3D space. The second set of equations also performs the homogeneous division that maps the 3D points to the destination-image plane. The second set of equations is:

$$\begin{aligned} x' &= w'_{11}x + w'_{12}y + w'_{13} + w'_{14}\delta(u_1,v_1) && [A] \\ y' &= w'_{21}x + w'_{22}y + w'_{23} + w'_{24}\delta(u_1,v_1) && [B] \\ z' &= w'_{31}x + w'_{32}y + w'_{33} + w'_{34}\delta(u_1,v_1) && [C] \\ u'_2 &= \frac{x'}{z'} && [D] \\ v'_2 &= \frac{y'}{z'} && [E] \end{aligned} \tag{5.3}$$

The following matrix, formed from nine of the coefficients in the above equations, is a rotation matrix:

$$\begin{bmatrix} w'_{11} & w'_{12} & w'_{13} \\ w'_{21} & w'_{22} & w'_{23} \\ w'_{31} & w'_{32} & w'_{33} \end{bmatrix} \tag{5.4}$$

The requirement that this matrix be a rotation matrix forces a choice for the otherwise arbitrary common scale factor for the coefficients w'_{ij}. The scale factor is chosen so that the determinant of the matrix above is 1.

The coordinates u'_2 and v'_2 in Equations 5.3 are still in the destination image's 3D space, even though they represent points on the image plane. Thus, a final shift and scaling is required to map points from the destination-image plane to the destination-image coordinates (u_2, v_2). This shift and scaling forms the third set of equations:

$$\begin{aligned} u_2 &= s'_{u2} \cdot u'_2 + o'_{u2} \\ v_2 &= s'_{v2} \cdot v'_2 + o'_{v2} \end{aligned} \tag{5.5}$$

The s' and o' variables are primed in this equation to indicate that the shift and scale is of the form $imagecoord = f(3dcoord)$. In contrast, Equation 5.2, which used unprimed s and o variables, expressed a shift and scale that was of the opposite form $3dcoord = f(imagecoord)$.

Initially, I will analyze the second set of equations (Equations 5.3). Later, I will extend the analysis to the first and third sets of equations (Equations 5.2 and 5.5 respectively), and partially recombine them with the second set of equations. The second set of equations uses only 3D Cartesian coordinates, unlike the monolithic 3D warp equations (Equations 5.1). Thus, it is easy to analyze this second set of equations geometrically.

5.2.1 Important parameters

Before beginning the in-depth analysis, I will define several parameters that will be used repeatedly:

$$\begin{aligned}
srcxres &\equiv \text{Horizontal pixel resolution of source image} \\
srcyres &\equiv \text{Vertical pixel resolution of source image} \\
\beta_{src,x} &\equiv \text{One-half of horizontal field-of-view of source image} \\
\beta_{src,y} &\equiv \text{One-half of vertical field-of-view of source image} \\
destxres &\equiv \text{Horizontal pixel resolution of destination image} \\
destyres &\equiv \text{Vertical pixel resolution of destination image} \\
\beta_{dest,x} &\equiv \text{One-half of horizontal field-of-view of destination image} \\
\beta_{dest,y} &\equiv \text{One-half of vertical field-of-view of destination image} \\
destxticks &\equiv \text{Desired horizontal subpixel precision (per pixel) in dest. image} \\
destyticks &\equiv \text{Desired vertical subpixel precision (per pixel) in dest. image} \\
T_{max} &\equiv \text{Maximum distance between source and destination viewpoints} \\
z_{min} &\equiv \text{Minimum depth of an object in the source image}
\end{aligned} \tag{5.6}$$

An example will make the definition of *destxticks* and *destyticks* more clear. If $destxticks = 3$, then we desire image-space precision to $1/3$ of a pixel in the horizontal direction.

The parameters above are used to define the shift and scale constants used in Equations 5.2 and 5.2:

$$s_{u1} = \frac{2\tan(\beta_{src,x})}{srcxres} \qquad s_{v1} = \frac{2\tan(\beta_{src,y})}{srcyres} \tag{5.7}$$

$$o_{u1} = -\tan(\beta_{src,x}) \qquad o_{v1} = -\tan(\beta_{src,y}) \tag{5.8}$$

$$s'_{u2} = \frac{destxres}{2\tan(\beta_{dest,x})} \qquad s'_{v2} = \frac{destyres}{2\tan(\beta_{dest,y})} \tag{5.9}$$

$$o'_{u2} = \frac{1}{2}destxres \qquad o'_{v2} = \frac{1}{2}destyres \tag{5.10}$$

These definitions for the scale and offset constants assume that the source-image and destination-image planes are at unit distance from the origin. Using the notation described in Chapter 1, this condition is equivalent to requiring that $S_1 = 1.0$ and $S_2 = 1.0$. I make this assumption throughout this section. It is a reasonable assumption, since these scale factors can be set to any desired value when defining the image planes.

5.2.2 Bounds on x', y', and z' for perspective division

Equations 5.3 form the second of the three sets of equations into which I decomposed the 3D warp. In this subsection, I will bound the values of x', y' and z' that are computed in this second set of equations. Because these values are used by the perspective division that computes u'_2 and v'_2 (again, see Equations 5.3), the bounds on x', y' and z' are also bounds on the inputs to the perspective division operator.

The analysis in this section is based on geometric arguments. First I geometrically describe the simplified situation of a 3D warp that is restricted to rotation only. Then I generalize the geometric analysis to a 3D warp with both rotational and translational components, in which the magnitude of the translation is limited in a manner consistent with a PRW system. From this more general geometric analysis, I determine bounds on x', y' and z'.

To begin, consider a restricted 3D warp that assumes an unchanging viewpoint and allows only changes in view direction. That is, the warp is a perspective warp rather than a full 3D warp. Mathematically, this restriction is represented in Equations 5.3 by $w'_{14} = w'_{24} = w'_{34} = 0$. Alternatively, we can guarantee a perspective warp by requiring that all points represented in the source image are at infinite distance (i.e. $\delta(u_1, v_1) = 0$).

This restricted warp is depicted geometrically in Figure 5.1. The figure illustrates 1D images in a 2D space, rather than 2D images in a 3D space, but the concepts extend easily from 2D space to 3D space. Several of the figures in this chapter will follow this convention.

In Figure 5.1, the distance from the image plane to the center of projection is 1.0. As stated earlier, I impose this restriction on both the source and destination images throughout this section. This restriction is equivalent to choosing the value 1.0 for the arbitrary scale factor S defined in Equation 1.5.

With this restriction, I guarantee that all points on the source-image plane are at a distance between 1.0 and $1/cos(\beta_{src})$ from the source-image center of projection. Because the source and destination images share a center of projection for a perspective warp, the distance between a point on the source image plane (i.e. a point to be re-projected) and the destination-image center of projection also falls within this same range. This restriction on range would be useful for formulating the perspective re-projection as a fixed-point computation.

We can calculate a similar type of restricted range for the full 3D warp (i.e. removing the restriction to rotation only). Figure 5.2 represents this situation. As before, the figure depicts the 2D situation rather than the 3D situation.

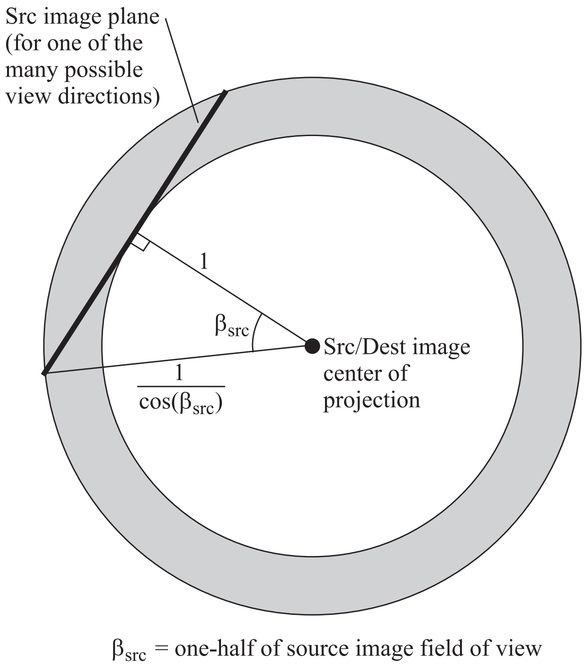


Figure 5.1: *A geometric depiction of the behavior of Equations 5.3 for the case of a rotation-only warp (perspective warp). The depiction is in 2D rather than 3D. The grey region indicates the region of world space which can be touched by points on the source-image plane, if we allow an arbitrary view direction for the source image. The destination image (not shown) shares the center of projection with the source image, and also has an arbitrary view direction.*

Figure 5.2 uses McMillan's disparity-based geometric depiction of the 3D warp [McMillan97]. In this type of depiction, viewpoint change is expressed as movement of the source-image points rather than a separation between the source-image and destination-image viewpoints. The distance that source-image points move is proportional to both their disparity and to the distance between the source and destination viewpoints. This disparity-based geometric depiction accurately reflects the properties of equations 5.3.

If we assume bounds on the disparity and viewpoint translation, then we can bound the distance that source image points can move away from the source-image plane. I assume a maximum disparity value, δ_{max}, or, equivalently, a minimum source-image z value, $z_{min} = \frac{1}{\delta_{max}}$. This restriction is equivalent to a near clip plane in the source image.

I also assume a maximum distance between source-image and destination-image viewpoints, $T_{max} = \left\| \dot{C}_1 - \dot{C}_2 \right\|_{max}$. Then, the maximum the disparity-based movement away from the source image plane, t_{max}, is defined as:

$$t_{max} = \frac{T_{max}}{z_{min}} \tag{5.11}$$

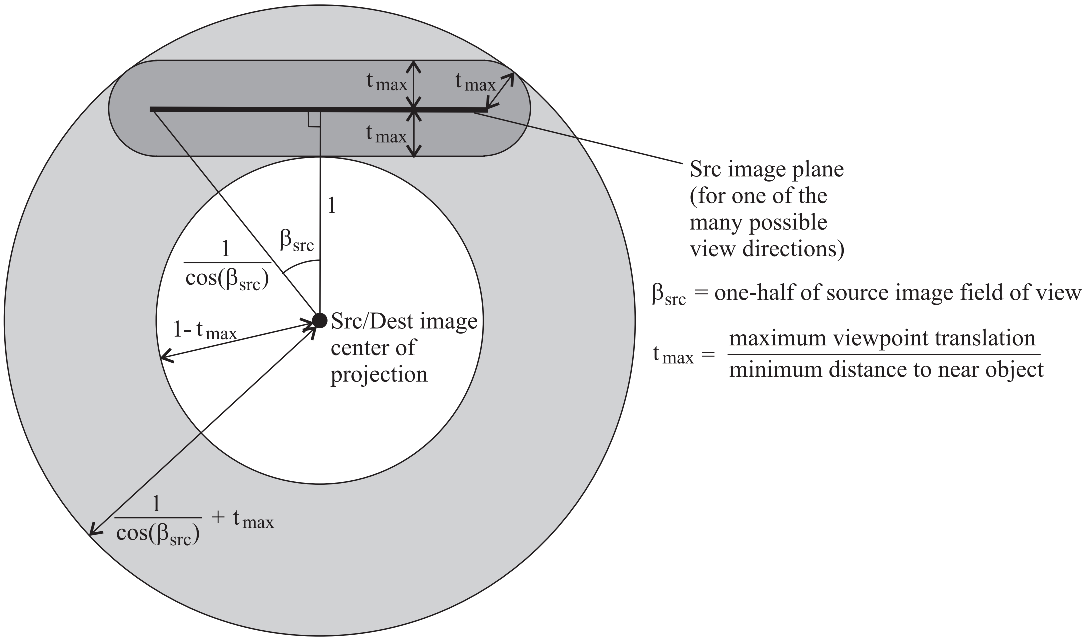


Figure 5.2: *A geometric depiction of the behavior of Equations 5.3 for the 3D warp. As in the previous figure, the depiction is in world space, but it is for a 2D world rather than a 3D world. The figure uses McMillan's disparity-based depiction of the 3D warp. Instead of representing the effects of viewpoint translation by showing separate source and destination viewpoints, the disparity-based depiction uses co-located source and destination viewpoints, and expresses translation by moving the source-image points away from the source-image plane. The magnitude of this movement is proportional to the disparity (δ) of the point. The movement is bounded if we impose restrictions on the distance to the nearest object in the scene and the distance between the (true) source and destination image viewpoints. The grey region of the figure indicates the region of world space which can be touched by points on the source-image plane, after they have been translated. In this disparity-based depiction, the destination image shares its center of projection with the source image, but the destination-image view direction (not shown) is arbitrary.*

Using this maximum movement distance, t_{max}, I can again calculate the region of space in which the (translated) source-image points will fall prior to re-projection. This region is shown in light grey in Figure 5.2. The points in this region are at a distance from the source/destination center between r_{min} and r_{max}, with:

$$\begin{aligned} r_{min,2D} &= 1 - t_{max} \\ r_{max,2D} &= \tfrac{1}{\cos(\beta_{src})} + t_{max} \end{aligned} \qquad (5.12)$$

For the 3D world, the distance to the farthest point on the image plane is different from what it is in the 2D world. I define $\beta_{src,x}$ and $\beta_{src,y}$ as one-half of the horizontal and vertical FOV's of the source image respectively. From these values, we can define the beta value for the source-image diagonal, $\beta_{src,xy}$:

$$\beta_{src,xy} = \arctan\left(\sqrt{\tan^2(\beta_{src,x}) + \tan^2(\beta_{src,y})}\right) \tag{5.13}$$

Then, for the 3D world,

$$\begin{aligned} r_{min} &= 1 - t_{max} \\ r_{max} &= \tfrac{1}{\cos(\beta_{src,xy})} + t_{max}. \end{aligned} \tag{5.14}$$

Now that I have described the behavior of points as they are transformed by the 3D warp, I can begin to place bounds on some of the intermediate values in the warp computation. Figure 5.3 shows the relationship of the destination-image view frustum to the region in which translated points can fall.

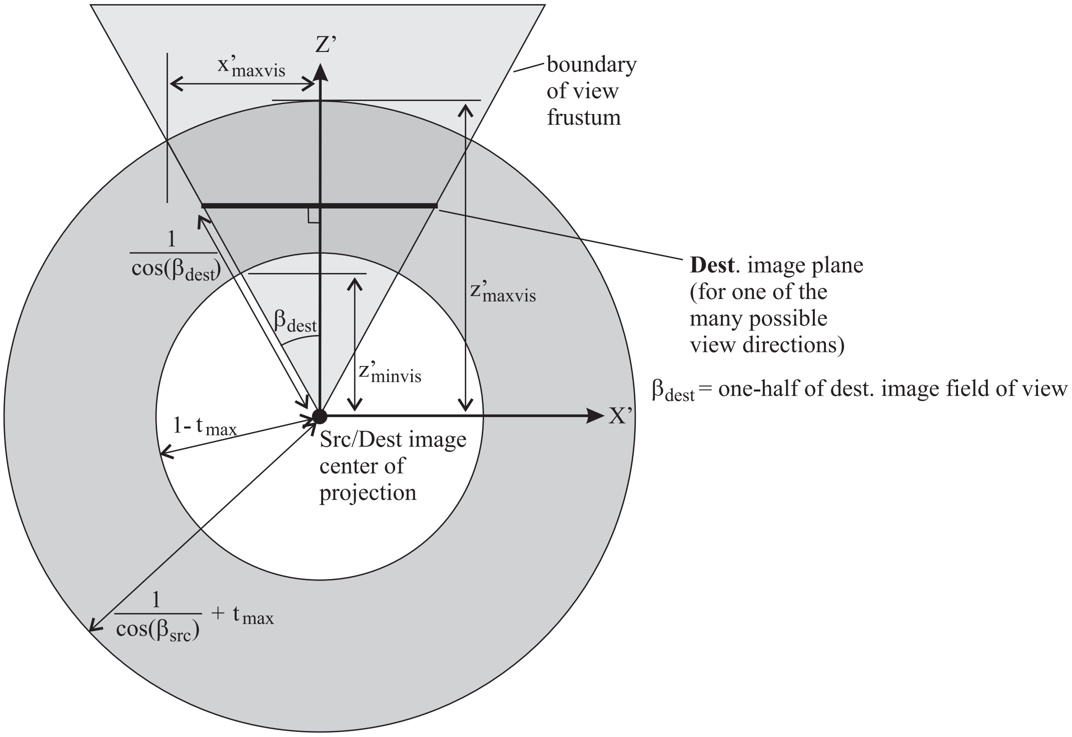

Z'
x'maxvis
boundary of view frustum
1
cos(βdest)
βdest
z'maxvis
z'minvis
Dest. image plane (for one of the many possible view directions)
βdest = one-half of dest. image field of view
X'
1- tmax
Src/Dest image center of projection
1
cos(βsrc)
+ tmax

Figure 5.3: *Valid locations of transformed points within the destination-image view frustum. The destination-image view frustum is shown in light grey. The destination-image's 3D coordinate system (actually 2D in this reduced-dimension figure) is indicated by the X' and Z' axes. The points to be projected onto the destination image plane all fall within the dark grey region, as calculated earlier. Thus, these points have a minimum and maximum possible z' value: z'_{min} and z'_{max}.*

The destination image's 3D coordinate system (sometimes called the *eye coordinate system*), is specified by the axes X', Y' and Z'. The Z' axis is perpendicular to the destination image plane. In this coordinate system, the points to be projected onto the destination image plane will have z' coordinates between z'_{minvis} and z'_{maxvis}.

The values of z'_{minvis} and z'_{maxvis} can be calculated from the values r_{min} and r_{max} which were determined earlier. For the 2D world case:

$$\begin{aligned} z'_{minvis,2D} &= r_{min,2D}\cos(\beta_{dest}) \\ z'_{maxvis,2D} &= r_{max,2D}. \end{aligned} \tag{5.15}$$

For the 3D world case, z'_{min} and z'_{max} become:

$$\begin{aligned} z'_{minvis} &= r_{min}\cos(\beta_{dest,xy}) \quad \text{where} \quad \beta_{dest,xy} = \arctan\left(\sqrt{\tan^2(\beta_{dest,x}) + \tan^2(\beta_{dest,y})}\right) \\ z'_{maxvis} &= r_{max}. \end{aligned} \tag{5.16}$$

Similarly, we can guarantee bounds for the x' and y' coordinates of points that fall within the view frustum. These points will have x coordinates between $-x_{maxvis}$ and $+x_{maxvis}$, and y coordinates between $-y_{maxvis}$ and $+y_{maxvis}$. For the 2D world case,

$$x'_{maxvis,2D} = r_{max,2D}\sin(\beta_{dest}) \tag{5.17}$$

For the 3D world case,

$$\begin{aligned} x'_{maxvis} &= r_{max}\sin(\beta_{dest,x}) \\ y'_{maxvis} &= r_{max}\sin(\beta_{dest,y}) \end{aligned} \tag{5.18}$$

I will now summarize the results of this subsection. Consider a 3D warp that represents an arbitrary rotation and a translation by less than a pre-determined maximum distance. Then, we can bound the values of x', y', and z' in Equations 5.3 for points that fall within the destination-image view frustum. For these points,

$$\begin{aligned} |x'| &\leq x'_{maxvis} \\ |y'| &\leq y'_{maxvis} \\ z'_{minvis} \leq z' &\leq z'_{maxvis} \end{aligned} \tag{5.19}$$

If a point is outside the destination-image view frustum, it is possible for the x', y', or z' coordinates to fall outside the bounds just given. However, since such points are not visible, there is no need to perform the perspective divisions needed to calculate u'_2 and v'_2 from x', y' and z' (Equations 5.3[D] and 5.3[E]). If the warping system detects such out-of-range coordinates prior to the perspective division, then the bounds provided above represent the maximum and minimum magnitudes of values fed to the perspective division unit(s).

5.2.3 Precision in perspective division

In the previous subsection, I calculated the range of magnitudes for the values x', y', and z' which are fed to the perspective division unit. In this subsection, I will calculate the precision required for these values in order to obtain the desired precision in u' and v'.

The first step in this process is to determine the required precision in u' and v'. Several constants are needed:

$$\begin{aligned} u'_{2,prec} &= \frac{2\tan(\beta_{dest,x})}{destxres \cdot destxticks} \\ v'_{2,prec} &= \frac{2\tan(\beta_{dest,y})}{destyres \cdot destyticks} \end{aligned} \tag{5.20}$$

Standard error propagation analysis techniques [Pizer83] tell us that for a function $c = f(a)$, the absolute error in the input value is related to the absolute error in the output value by $c_{abs} \approx a_{abs}\left|f'(a)\right|$. Similarly, for a two-variable function $c = f(a,b)$, the relationship is $c_{abs} \approx a_{abs}\left|\frac{\partial f}{\partial a}\right| + b_{abs}\left|\frac{\partial f}{\partial b}\right|$. For $c = \frac{a}{b}$, we get $c_{abs} \approx a_{abs}\left|\frac{1}{b}\right| + b_{abs}\left|\frac{a}{b^2}\right|$. Applying this formula to Equations 5.3[D] and 5.3[E], we get:

$$\begin{aligned} u'_{2,abs} &\approx x'_{abs}\left|\frac{1}{z'}\right| + z'_{abs}\left|\frac{x'}{(z')^2}\right| \\ v'_{2,abs} &\approx y'_{abs}\left|\frac{1}{z'}\right| + z'_{abs}\left|\frac{y'}{(z')^2}\right| \end{aligned} \tag{5.21}$$

The worst-case error described by these expressions occurs when $x' = x'_{maxvis}$, $y' = y'_{maxvis}$, and $z' = z'_{minvis}$. I split the allowed error equally between the two inputs (and substitute $=$ for $\approx$), so that:

$$\begin{aligned} x'_{prec} &= \tfrac{1}{2}u'_{2,prec} \cdot z'_{minvis} \\ y'_{prec} &= \tfrac{1}{2}v'_{2,prec} \cdot z'_{minvis} \\ z'_{prec} &= \min\left(\tfrac{1}{2}u'_{2,prec}\frac{\left(z'_{minvis}\right)^2}{x'_{maxvis}} \quad , \quad \tfrac{1}{2}v'_{2,prec}\frac{\left(z'_{minvis}\right)^2}{y'_{maxvis}}\right) \end{aligned} \tag{5.22}$$

Any actual implementation of the perspective divisions will probably perform one reciprocal operation and two multiplies, rather than two full divisions. The reason is that divisions (and reciprocal operations) are expensive, so it is usually more efficient to perform the reciprocal just once. I will now analyze the reciprocal and multiply operations. Let:

$$\begin{aligned} R &= \frac{1}{z'} \\ u'_2 &= Rx' \\ v'_2 &= Ry' \end{aligned} \tag{5.23}$$

Then, with an analysis similar to the earlier one for the division-based computation:

$$\begin{aligned}
R_{max} &= \frac{1}{z'_{minvis}} \\
R_{prec} &= \min\left(\frac{1}{2}\frac{u'_{2,prec}}{x'_{maxvis}}, \frac{1}{2}\frac{v'_{2,prec}}{y'_{maxvis}}\right) \\
z'_{prec} &= R_{prec}\left(z'_{minvis}\right)^2 \\
&= \min\left(\frac{1}{2}u'_{2,prec}\frac{\left(z'_{minvis}\right)^2}{x'_{maxvis}}, \frac{1}{2}v'_{2,prec}\frac{\left(z'_{minvis}\right)^2}{y'_{maxvis}}\right)
\end{aligned} \tag{5.24}$$

Not surprisingly, the equation for z'_{prec} is the same as the one for the division-based computation.

5.2.4 Bounds on x', y', and z' for sum accumulators

I have already calculated the maximum values of x', y', and z' for points that are visible. However, these maximum values may be exceeded for points that are not visible. Even for points that are visible, these maximum values may be temporarily exceeded during the intermediate computations required by Equations 5.3[A,B,C].

A software or hardware implementation that computes the sum expressed by each of Equations 5.3[A,B,C] requires one or more accumulators for the partial sum. In this subsection, I compute the size required by these accumulators.

All three accumulators are used to compute sums of the following form:

$$Ax + By + C + D\delta \tag{5.25}$$

We will temporarily neglect the last term of this expression. Because of the rotation-matrix constraint stated for Equation 5.4, the coefficients A, B, and C above satisfy the property that $A^2 + B^2 + C^2 = 1$. We will also assume that x and y are bounded, so that $|x| \leq x_{max}$ and $|y| \leq y_{max}$. Then, using the derivation in Appendix D, we know that:

$$|Ax| + |By| + |C| \leq \frac{x_{max}^2 + y_{max}^2 + 1}{\sqrt{x_{max}^2 + y_{max}^2 + 1}} \tag{5.26}$$

I will now consider the effect of the disparity-based translation represented by the term $D\delta$. The magnitude of this term is bounded by t_{max}. So,

$$|Ax| + |By| + |C| + |D| \leq \frac{x_{max}^2 + y_{max}^2 + 1}{\sqrt{x_{max}^2 + y_{max}^2 + 1}} + t_{max} \tag{5.27}$$

Applying this bound to the accumulators used for computing Equations Equations 5.3[A,B,C] gives us:

$$x'_{maxacc} = y'_{maxacc} = z'_{maxacc} = \frac{x_{max}^2 + y_{max}^2 + 1}{\sqrt{x_{max}^2 + y_{max}^2 + 1}} + t_{max} \tag{5.28}$$

The values x_{max} and y_{max} are determined by the FOV of the source image:

$$\begin{aligned} x_{max} &= \tan(\beta_{src,x}) \\ y_{max} &= \tan(\beta_{src,y}) \end{aligned} \tag{5.29}$$

5.2.5 Precision of x', y', and z' sum accumulators

The accumulators which hold the partial sums needed to compute x', y' and z' must have greater precision than the final result, because each addition contributes error. Let $x'_{accprec}$ represent the precision of the x' accumulator, and of the products used to compute it. Each product carries an error with it of up to $\frac{1}{2}x'_{accprec}$. Thus, the total error in the sum is $\frac{4}{2}x'_{accprec}$. We want the error to be less than or equal to $\frac{1}{2}x'_{prec}$. The same analysis applies to $y'_{accprec}$ and $z'_{accprec}$. So,

$$\begin{aligned} x'_{accprec} &= \tfrac{1}{4}x'_{prec} \\ y'_{accprec} &= \tfrac{1}{4}y'_{prec} \qquad \text{[Non-Incremental]} \\ z'_{accprec} &= \tfrac{1}{4}z'_{prec} \end{aligned} \tag{5.30}$$

McMillan and Bishop showed that the 3D warp equations can be computed incrementally, by traversing the reference image in a raster scan pattern so that the $w'_{i1}x$ and $w'_{i2}y$ products are replaced by additions [McMillan95a]. Since additions are generally less expensive than multiplications, this formulation of the 3D warp can improve performance and/or lower cost. I will consider the implications of an incremental evaluation across a single scan line of the reference image. In this situation, the $w'_{i2}y$ product is a constant, and the $w'_{i1}x$ product is replaced by an addition.

In such a design, the addition of the $w'_{i4}\delta$ product is performed last, in a separate step, and can be done with the precision described in Equation 5.30 (in fact, with one bit less). The addition of w'_{i1} is done first, and must be done with better precision, because the resulting partial sum is preserved from pixel to pixel. I will refer to the precision (value of least-significant bit) of the w'_{11} partial-sum accumulator as $x'_{accprec'}$.

In the incremental case, the total error in the sum due to the incremental computation is $\frac{srcxres}{2}x'_{accprec'}$. So,

$$\begin{aligned} x'_{accprec'} &= \tfrac{1}{srcxres}x'_{accprec} \\ y'_{accprec'} &= \tfrac{1}{srcxres}y'_{accprec} \qquad \text{[Incremental]} \\ z'_{accprec'} &= \tfrac{1}{srcxres}z'_{accprec} \end{aligned} \tag{5.31}$$

Thus, for an incrementally-computed warp, the w_{i1} values, and the partial sum, must be computed at the $accprec'$ precisions. The other w_{ij} values can be computed at the same $accprec$ precision that is used for a non-incrementally-computed warp.

5.2.6 *Multiplying by* w_{ij}

In this subsection, I analyze the multipliers that use the coefficients w_{ij} as one of their inputs. For a multiplier $c = a \cdot b$, the absolute error in the output is related to the absolute error in the inputs as follows:

$$c_{abs} \approx a_{abs} \cdot b + b_{abs} \cdot a \tag{5.32}$$

The worst-case error occurs for the maximum values of a and b. So, for the w_{1j} multipliers (and similarly for the w_{2j} and w_{3j} multipliers):

$$\begin{aligned} w'_{11,prec} \cdot x_{max} + x_{prec(1)} \cdot 1.0 &= x'_{accprec} \\ w'_{12,prec} \cdot y_{max} + y_{prec(1)} \cdot 1.0 &= x'_{accprec} \\ w'_{14,prec} \cdot \delta_{max} + \delta_{prec(1)} \cdot w'_{14,max} &= x'_{accprec} \end{aligned} \tag{5.33}$$

The subscripted (1) in $x_{prec(1)}$ indicates that the variable represents the precision required for x to compute Equation 5.3[A]. A subscript of (2) would indicate the precision required to compute Equation 5.3[B]. The value $w'_{14,max}$ is equal to the maximum distance between the source-image and destination-image viewpoints, T_{max}. The value δ_{max} is equal to $\frac{1}{z_{min}}$. So,

$$\frac{w'_{14,prec}}{z_{min}} + \delta_{prec(1)} \cdot T_{max} = x'_{accprec} \tag{5.34}$$

I split the allowed error evenly between the two inputs. Then:

$$\begin{aligned}
w'_{11,prec} &= \frac{x'_{accprec}}{2x_{max}} & w'_{12,prec} &= \frac{x'_{accprec}}{2y_{max}} & w'_{14,prec} &= \frac{1}{2}x'_{accprec}\cdot z_{min} \\
w'_{21,prec} &= \frac{y'_{accprec}}{2x_{max}} & w'_{22,prec} &= \frac{y'_{accprec}}{2y_{max}} & w'_{24,prec} &= \frac{1}{2}y'_{accprec}\cdot z_{min} \\
w'_{31,prec} &= \frac{z'_{accprec}}{2x_{max}} & w'_{32,prec} &= \frac{z'_{accprec}}{2y_{max}} & w'_{34,prec} &= \frac{1}{2}z'_{accprec}\cdot z_{min} \qquad (5.35) \\
x_{prec(1)} &= \frac{1}{2}x'_{accprec} & y_{prec(1)} &= \frac{1}{2}x'_{accprec} & \delta_{prec(1)} &= \frac{x'_{accprec}}{2T_{max}} \\
x_{prec(2)} &= \frac{1}{2}y'_{accprec} & y_{prec(2)} &= \frac{1}{2}y'_{accprec} & \delta_{prec(2)} &= \frac{y'_{accprec}}{2T_{max}} \\
x_{prec(3)} &= \frac{1}{2}z'_{accprec} & y_{prec(3)} &= \frac{1}{2}z'_{accprec} & \delta_{prec(3)} &= \frac{z'_{accprec}}{2T_{max}}
\end{aligned}$$

Later, I will combine Equations 5.2 and 5.3, so that x and y will become u_1 and v_1. Since u_1 and v_1 are integer valued, they will always be represented with perfect precision if the one's digit of the integer is represented. Therefore, I will not discuss x_{prec} and y_{prec} any further.

Almost everywhere in the evaluation of the 3D warp equations, the actual values of z_{min} and T_{max} are not relevant—only the ratio between the two is relevant. The inputs to the multipliers that evaluate $w'_{i4}\cdot\delta$ are an exception to this generalization, as evidenced by the appearance of z_{min} and T_{max} in the equations above. The $w'_{i4}\cdot\delta$ multipliers must be able to accommodate the full range of possible $\frac{1}{z}$ values and the full range of possible reference-to-destination viewpoint distances. For image-warping systems that warp pre-computed images, it might be desirable to pre-scale these values before run-time to constrain them to a specific range. In a post-rendering warping system, the handling of this issue is intertwined with the specific representation used by the conventional renderer for the z-buffer.

5.2.7 Putting it all together

So far, I have analyzed the second equation (Equation 5.3) from the set of three equations into which I broke down the 3D warp. It is now time to unify the three equations, and to adjust the analysis accordingly.

The unified equations do not have quite the same form as the original 3D warp equation (Equation 5.1). Instead, they have the following form, in which an extra addition is performed after the perspective division:

$$u_2 = \frac{w^*_{11}u_1 + w^*_{12}v_1 + w^*_{13} + w^*_{14}\delta(u_1,v_1)}{w^*_{31}u_1 + w^*_{32}v_1 + w^*_{33} + w^*_{34}\delta(u_1,v_1)} + o'_{u2}$$

$$v_2 = \frac{w^*_{21}u_1 + w^*_{22}v_1 + w^*_{23} + w^*_{24}\delta(u_1,v_1)}{w^*_{31}u_1 + w^*_{32}v_1 + w^*_{33} + w^*_{34}\delta(u_1,v_1)} + o'_{v2}, \qquad \text{where} \tag{5.36}$$

The new w^* coefficients can be expressed in terms of the w' coefficients used in Equation 5.3:

$$\begin{aligned}
w^*_{11} &= s'_{u2}s_{u1}w'_{11} & w^*_{12} &= s'_{u2}s_{v1}w'_{12} \\
w^*_{13} &= s'_{u2}\left(w'_{13} + w'_{11}o_{u1} + w'_{12}o_{v1}\right) & w^*_{14} &= s'_{u2}w'_{14} \\
w^*_{21} &= s'_{v2}s_{u1}w'_{21} & w^*_{22} &= s'_{v2}s_{v1}w'_{22} \\
w^*_{23} &= s'_{v2}\left(w'_{23} + w'_{21}o_{u1} + w'_{22}o_{v1}\right) & w^*_{24} &= s'_{v2}w'_{24} \\
w^*_{31} &= s_{u1}w'_{31} & w^*_{32} &= s_{v1}w'_{32} \\
w^*_{33} &= w'_{33} + w'_{31}o_{u1} + w'_{32}o_{v1} & w^*_{34} &= w'_{34}
\end{aligned} \tag{5.37}$$

Alternatively, the new w^* coefficients can be expressed in terms of the w coefficients which are used in Equation 5.1 and defined in Equation 1.13:

$$\begin{aligned}
w^*_{11} &= w_{11} - o'_{u2}w_{31} & w^*_{12} &= w_{12} - o'_{u2}w_{32} & w^*_{13} &= w_{13} - o'_{u2}w_{33} & w^*_{14} &= w_{14} - o'_{u2}w_{34} \\
w^*_{21} &= w_{21} - o'_{v2}w_{31} & w^*_{22} &= w_{22} - o'_{v2}w_{32} & w^*_{23} &= w_{23} - o'_{v2}w_{33} & w^*_{24} &= w_{24} - o'_{v2}w_{34} \\
w^*_{31} &= w_{31} & w^*_{32} &= w_{32} & w^*_{33} &= w_{33} & w^*_{34} &= w_{34}
\end{aligned} \tag{5.38}$$

These definitions of the w^* coefficients in terms of the w coefficients are only correct if the scale factors S_1 and S_2 corresponding to $\mathbf{P}_1$ and $\mathbf{P}_2$ are equal to one, as I required at the beginning of this section. I will now explain why.

A $\mathbf{P}$ matrix for a non-skewed image centered about the view direction can be decomposed into the following form:

$$\mathbf{P} = \mathbf{RQ}, \quad \text{with} \quad \mathbf{Q} = \begin{bmatrix} s_u & 0 & o_u \\ 0 & s_v & o_v \\ 0 & 0 & 1 \end{bmatrix} \tag{5.39}$$

The matrix $\mathbf{R}$ is a scaled rotation matrix, with $\det \mathbf{R} = \pm S^3$. If $S = 1$, then $\mathbf{R}$ is a true rotation matrix, with $\det \mathbf{R} = \pm 1$. Earlier, I stated that the matrix in Equation 5.4 must be a true rotation matrix.

This matrix is the product of $\mathbf{R}_2^{-1}$ and $\mathbf{R}_1$. Thus, if $\mathbf{R}_1$ and $\mathbf{R}_2$ are both true rotation matrices, then the matrix in Equation 5.4 will be also.

The revised warp equation described in Equation 5.36 requires corresponding revisions in the precisions and maximum magnitudes calculated earlier in this section. I will now calculate these revised precisions and magnitudes.

The destination-image scale factors s'_{u2} and s'_{v2} in Equation 5.36 cause a scaling of the values fed to the perspective division unit. Thus, I must restate the size and precision required for several variables. The new maxima and precisions are denoted by a * below, and are expressed in terms of the original ones:

$$\begin{aligned}
x'^*_{maxvis} &= s'_{u2}x'_{maxvis} \\
y'^*_{maxvis} &= s'_{v2}y'_{maxvis} \\
z'^*_{maxvis} &= z'_{maxvis} \\
z'^*_{minvis} &= z'_{minvis} \\
x'^*_{prec} &= s'_{u2}x'_{prec} \\
y'^*_{prec} &= s'_{v2}y'_{prec} \\
z'^*_{prec} &= z'_{prec}
\end{aligned} \tag{5.40}$$

The scale and shift of the reference-image coordinates (using s_{u1}, s_{v1}, o_{u1}, and o_{v1}) does not change the values fed to the perspective division unit. The reason is that my earlier analysis assumed that this scale and shift had already been performed. Therefore, the precisions required to compute the numerator and denominator of the perspective division are not affected by the reference-image scale and shift values. However, the reference-image scale and shift *do* affect the maximum size of the intermediate expressions used in computing the numerator and denominator.

In Equation 5.36 the offset is separated into the constant term, rather than being performed on u_1 and v_1 prior to multiplication. The centering of the image plane about the view direction is performed by this offset, so separating the offset into a different term is equivalent to working with an un-centered image in the initial part of the computation. For the un-centered image, x_{max} and y_{max} are doubled, which will in turn increase the maximum value to be stored in the x', y', and z' accumulators:

$$\begin{aligned} x^*_{max} &= 2x_{max} \\ y^*_{max} &= 2y_{max} \\ maxacc^* &= \frac{(x^*_{max})^2 + (y^*_{max})^2 + 1}{\sqrt{(x^*_{max})^2 + (y^*_{max})^2 + 1}} + t_{max} \end{aligned} \tag{5.41}$$

Incorporating the destination-image scale factors:

$$\begin{aligned} x'^*_{maxacc} &= s'_{u2} maxacc^* \\ y'^*_{maxacc} &= s'_{v2} maxacc^* \\ z'^*_{maxacc} &= maxacc^* \end{aligned} \tag{5.42}$$

The scaling by s_{u1} and s_{v1} affects the magnitude and precision required for the w'_{i1} and w'_{i2} multipliers (now the w^*_{i1} and w^*_{i2} multipliers), as does the scaling by s'_{u2} and s'_{v2}.

$$\begin{array}{lll} w^*_{11,max} = s'_{u2}s_{u1} & w^*_{12,max} = s'_{u2}s_{v1} & w^*_{14,max} = s'_{u2}t_{max} \\ w^*_{21,max} = s'_{v2}s_{u1} & w^*_{22,max} = s'_{v2}s_{v1} & w^*_{24,max} = s'_{v2}t_{max} \\ w^*_{31,max} = s_{u1} & w^*_{32,max} = s_{v1} & w^*_{34,max} = t_{max} \\ \\ w^*_{11,prec} = s'_{u2}s_{u1}w'_{11,prec} & w^*_{12,prec} = s'_{u2}s_{v1}w'_{12,prec} & w^*_{14,prec} = s'_{u2}w'_{14,prec} \\ w^*_{21,prec} = s'_{v2}s_{u1}w'_{21,prec} & w^*_{22,prec} = s'_{v2}s_{v1}w'_{22,prec} & w^*_{24,prec} = s'_{v2}w'_{24,prec} \\ w^*_{31,prec} = s_{u1}w'_{31,prec} & w^*_{32,prec} = s_{v1}w'_{32,prec} & w^*_{34,prec} = w'_{34,prec} \end{array} \tag{5.43}$$

Trivially,

$$\begin{aligned} u_{1,max} &= srcxres \\ v_{1,max} &= srcyres \end{aligned} \tag{5.44}$$

Figure 5.4 illustrates the computation tree represented by Equation 5.36. The maximum values and the precisions of the various intermediate values are shown symbolically. The figure also shows the number of bits required for the intermediate values, given a particular set of base assumptions.

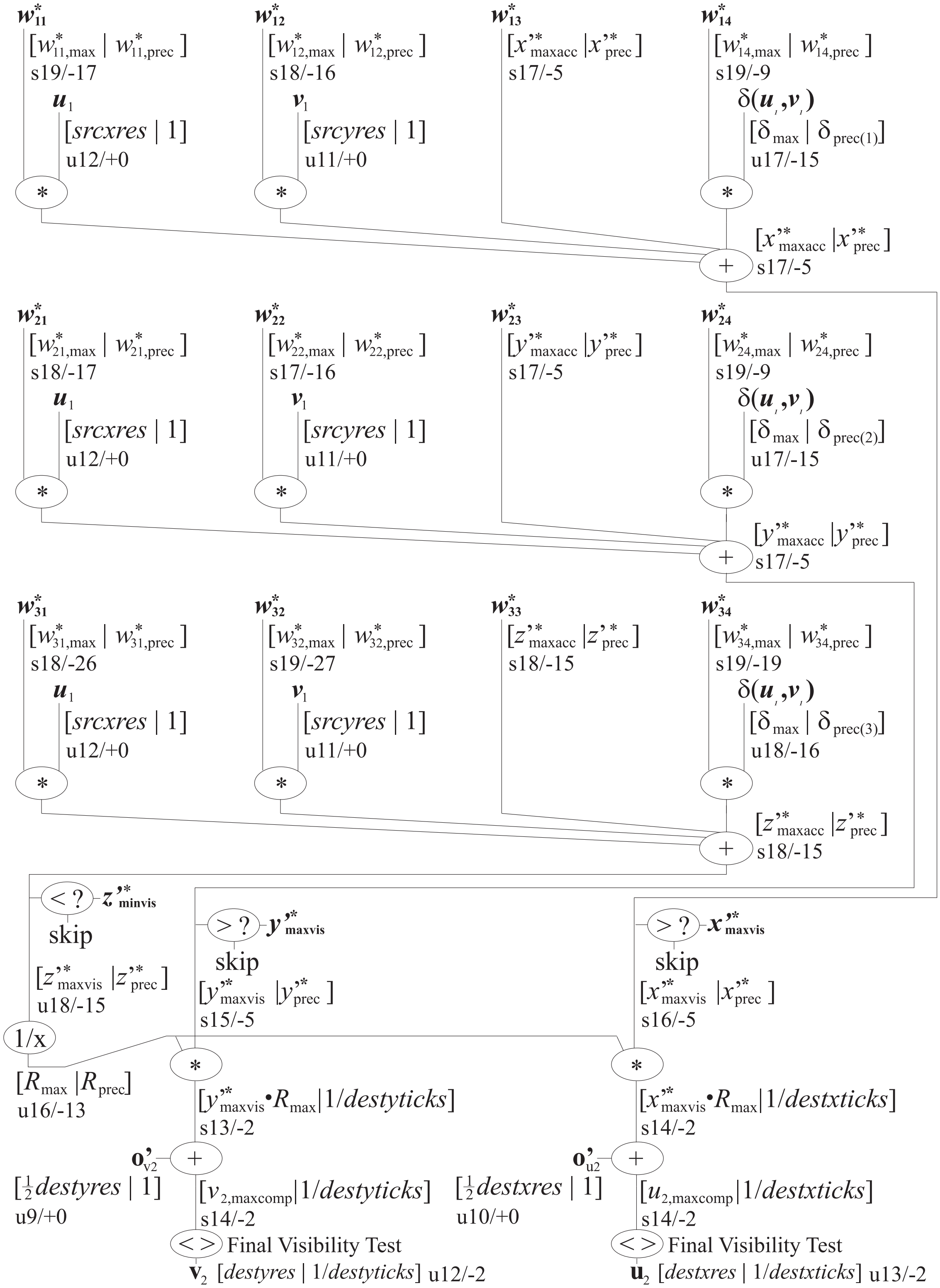

w^*_{11}
$[w^*_{11,\max} \mid w^*_{11,\text{prec}}]$
s19/-17
u_1
$[srcxres \mid 1]$
u12/+0
*
w^*_{12}
$[w^*_{12,\max} \mid w^*_{12,\text{prec}}]$
s18/-16
v_1
$[srcyres \mid 1]$
u11/+0
*
w^*_{13}
$[x'^*_{\text{maxacc}} \mid x'^*_{\text{prec}}]$
s17/-5
w^*_{14}
$[w^*_{14,\max} \mid w^*_{14,\text{prec}}]$
s19/-9
$\delta(u_1, v_1)$
$[\delta_{\max} \mid \delta_{\text{prec}(1)}]$
u17/-15
*
+
$[x'^*_{\text{maxacc}} \mid x'^*_{\text{prec}}]$
s17/-5
w^*_{21}
$[w^*_{21,\max} \mid w^*_{21,\text{prec}}]$
s18/-17
u_1
$[srcxres \mid 1]$
u12/+0
*
w^*_{22}
$[w^*_{22,\max} \mid w^*_{22,\text{prec}}]$
s17/-16
v_1
$[srcyres \mid 1]$
u11/+0
*
w^*_{23}
$[y'^*_{\text{maxacc}} \mid y'^*_{\text{prec}}]$
s17/-5
w^*_{24}
$[w^*_{24,\max} \mid w^*_{24,\text{prec}}]$
s19/-9
$\delta(u_1, v_1)$
$[\delta_{\max} \mid \delta_{\text{prec}(2)}]$
u17/-15
*
+
$[y'^*_{\text{maxacc}} \mid y'^*_{\text{prec}}]$
s17/-5
w^*_{31}
$[w^*_{31,\max} \mid w^*_{31,\text{prec}}]$
s18/-26
u_1
$[srcxres \mid 1]$
u12/+0
*
w^*_{32}
$[w^*_{32,\max} \mid w^*_{32,\text{prec}}]$
s19/-27
v_1
$[srcyres \mid 1]$
u11/+0
*
w^*_{33}
$[z'^*_{\text{maxacc}} \mid z'^*_{\text{prec}}]$
s18/-15
w^*_{34}
$[w^*_{34,\max} \mid w^*_{34,\text{prec}}]$
s19/-19
$\delta(u_1, v_1)$
$[\delta_{\max} \mid \delta_{\text{prec}(3)}]$
u18/-16
*
+
$[z'^*_{\text{maxacc}} \mid z'^*_{\text{prec}}]$
s18/-15
< ?
z'^*_{minvis}
skip
$[z'^*_{\text{maxvis}} \mid z'^*_{\text{prec}}]$
u18/-15
1/x
$[R_{\max} \mid R_{\text{prec}}]$
u16/-13
> ?
y'^*_{maxvis}
skip
$[y'^*_{\text{maxvis}} \mid y'^*_{\text{prec}}]$
s15/-5
*
$[y'^*_{\text{maxvis}} \bullet R_{\max} \mid 1/destyticks]$
s13/-2
o'_{v2}
+
$[\frac{1}{2} destyres \mid 1]$
u9/+0
$[v_{2,\text{maxcomp}} \mid 1/destyticks]$
s14/-2
< >
Final Visibility Test
v_2 $[destyres \mid 1/destyticks]$ u12/-2
> ?
x'^*_{maxvis}
skip
$[x'^*_{\text{maxvis}} \mid x'^*_{\text{prec}}]$
s16/-5
*
$[x'^*_{\text{maxvis}} \bullet R_{\max} \mid 1/destxticks]$
s14/-2
o'_{u2}
+
$[\frac{1}{2} destxres \mid 1]$
u10/+0
$[u_{2,\text{maxcomp}} \mid 1/destxticks]$
s14/-2
< >
Final Visibility Test
u_2 $[destxres \mid 1/destxticks]$ u13/-2

Figure 5.4: *Computation tree for fixed-point 3D warp transformation. Sizes and precisions are given symbolically as [maxval | precision], and numerically (for the parameters in the text) as* $s|uN/\pm M$. *s=signed, u=unsigned. N=total bits, not including sign. M=power-of-two for least significant bit.*

The power-of-two for the most and least significant bits of a variable q is computed as:

$$\begin{aligned} MSB &= \lceil \log_2 (q_{max}) \rceil \\ LSB &= \lfloor \log_2 (q_{prec}) \rfloor \end{aligned} \tag{5.45}$$

Thus, the total number of bits required is:

$$numbits = MSB - LSB + 1 \tag{5.46}$$

Two additional maxima appear in Figure 5.4. These values are the maximum possible values of u_2 and v_2 for the computation described in the figure prior to the final visibility test.

$$u_{2,maxcomp} = x'^{*}_{maxvis} \cdot R_{max} + \frac{1}{2} destxres \tag{5.47}$$

$$v_{2,maxcomp} = y'^{*}_{maxvis} \cdot R_{max} + \frac{1}{2} destyres \tag{5.48}$$

The values of parameters used to compute the numerical bit counts in Figure 5.4 are listed below. The values for source-image and destination-image parameters are the same as those used for the Brooks' house kitchen sequence discussed in Chapter 3. These values are marked with a dagger (†).

$$\begin{array}{cccc} srcxres = 1070\dagger & srcyres = 745\dagger & \beta_{src,x} = \frac{1}{2} \cdot 88^\circ\dagger & \beta_{src,y} = \frac{1}{2} \cdot 68^\circ\dagger \\ destxres = 640\dagger & destyres = 480\dagger & \beta_{dest,x} = \frac{1}{2} \cdot 60^\circ\dagger & \beta_{dest,y} = \frac{1}{2} \cdot 47^\circ\dagger \\ destxticks = 4 & destyticks = 4 & T_{max} = 0.25m & z_{min} = 0.5m \end{array} \tag{5.49}$$

One somewhat surprising result shown in Figure 5.4 is that the disparity values must be represented by eighteen bits to insure an accurate warp (with the example parameters). To researchers who have worked with 3D warping systems, this precision intuitively seems too high. I will now explain this apparent contradiction.

At each step in the computation, the error propagation equations used in this chapter (for example, Equation 5.22) split the allowed error evenly between each possible source. In practice, the values of w_{ij} may be known with greater precision than the value of $\delta(\bar{u}_1)$. We would expect this asymmetry to exist for systems that warp acquired imagery. If we assume that all inputs *except* for δ are represented with infinite precision, then the required precision of $\delta(\bar{u}_1)$ is reduced by a factor of sixteen. Then, $\delta(\bar{u}_1)$ would need to be represented by only fourteen bits. Of course, it is unrealistic to represent the w_{ij} values with infinite precision, but by merely adding a few bits to them we can come close to the ideal.

The analysis in this chapter is a worst-case analysis. For objects that are far away, or for smaller movements, $\delta(\bar{u}_1)$ can be represented less precisely. By making worst-case assumptions at each stage of the computation, the analysis may also be overly conservative in its final result. I have not done the work to determine whether or not this step-by-step approach significantly affects the results.

5.2.8 Discussion

The analysis in this chapter assumes perfect arithmetic units. In other words, the arithmetic units propagate error, but do not generate any. In any real implementation, it is likely that the reciprocal unit (and possibly others) would generate some error. The analysis would have to be adjusted slightly to account for this generated error.

In this analysis, I allowed an arbitrary rotation between the source and destination images. If the rotation were restricted to substantially less than 90°, then the precisions required in the warp computation could probably be reduced somewhat. I have not explored this possibility in detail.

The analysis in this section requires that the maximum translation distance T_{max} be less than the distance to the closest object, z_{min}. This requirement is reasonable for a post-rendering warping system, but is not necessarily reasonable for a system that warps acquired imagery. However, I believe that by using a near clipping plane in the destination image, the analysis could be easily adapted to allow for $T_{max} > z_{min}$. The number of bits required for the computation would still grow as T_{max} grows in comparison to z_{min}.

The reciprocal unit is likely to be the most expensive arithmetic unit required for the fixed-point 3D warp transform. But, for a warp that steps through adjacent source-image pixels in order, an interesting optimization is possible. I believe that a reciprocal unit based on an iterative computation would be very efficient if it was seeded with the result from the previous pixel. Since the reciprocals for adjacent pixels would usually be similar, the computation would typically complete with just one iteration. It is conceivable that, given the t_{max} bound, it might be possible to *guarantee* a result of the necessary precision in just one iteration. If not, a pipelined implementation could stall for one or more additional cycles to allow repeated iterations for the few pixels that require it.

5.3 Memory-access properties

This section discusses the memory access patterns of the 3D image warp. It also describes some warping algorithms that take advantage of these patterns to allow the use of a small cache. A cache can reduce the cost of the warper's memory system, because the cache allows the use of a high-latency main memory.

5.3.1 Reference-to-destination mapping

Once the pose of the reference image and the pose of destination image are specified, the behavior of the 3D warp is constrained. A particular reference-image pixel can no longer map to an arbitrary pixel in the destination image. Instead, each reference-image pixel maps to somewhere on a corresponding epipolar line in the destination image. The exact location on the line depends on the depth value associated with the reference-image pixel. All of the destination-image epipolar lines pass through the destination-image epipole. Figure 5.5 illustrates this mapping of reference-image pixels to epipolar lines in the destination image.

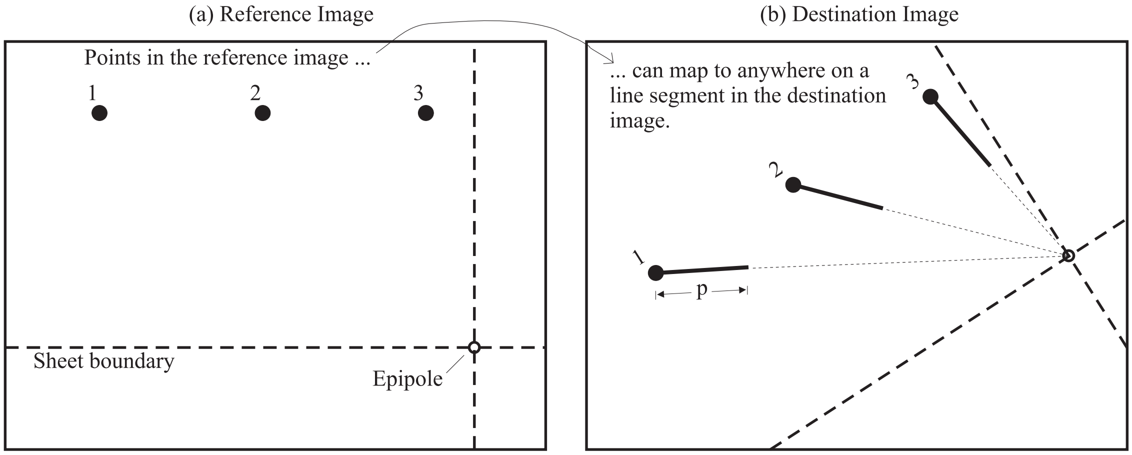

(a) Reference Image
(b) Destination Image
Points in the reference image ...
1
2
3
Sheet boundary
Epipole
... can map to anywhere on a line segment in the destination image.
3
2
1
p

Figure 5.5: *3D warp of points. A point in the reference image can map to anywhere on a line segment in the destination image. The actual location on the line depends on the point's depth. The dot at one end of these destination-image line segments shows the mapping that would result from a purely projective image warp. A point at infinite distance will map to this dot. The line segment extending from the dot thus represents the perturbation from a projective warp. Points with smaller z values will be perturbed further along this line. The distance p represents the maximum possible perturbation (established by bounds on the 3D warp's parameters, as discussed in the text).*

Unlike perspective and affine warps, the 3D warp's mapping is not inherently continuous. In a perspective image warp, the mapping from the reference image to the destination image varies continuously as a function of reference-image location. For a 3D warp, this mapping is not continuous, because it depends on the per-pixel depth values as well as the reference-image location. If the per-pixel depth values do not change smoothly, then the mapping does not change smoothly either.

However, if we place bounds on some of the 3D warp parameters, then we can make the 3D warp behave more like a perspective warp. We must bound two parameters: The per-pixel depth values, and the distance between the reference image viewpoint and destination image viewpoint. These are the same parameters that had to be bounded in order to guarantee the accuracy of the fixed-point formulation of the 3D warp's transform equations.

With bounds on these parameters, the discontinuities in the 3D warp's mapping are bounded in magnitude. A given reference-image pixel can potentially map to only a small set of destination-image pixels. This set of pixels forms a line segment in the reference image, rather than a complete line. Figure 5.5 shows the extent of these line segments.

This behavior of the 3D warp can be expressed mathematically. I begin this mathematical derivation with a formulation of the 3D warp based on a reorganization of Equation 1.4:

$$\left(\frac{z_2}{z_1}\right)\bar{u}_2 = \mathbf{P}_2^{-1}\mathbf{P}_1\bar{u}_1 + \frac{1}{z_1}\mathbf{P}_2^{-1}\left(\dot{C}_1 - \dot{C}_2\right) \tag{5.50}$$

Without losing generality, I have assumed that the scale factors S_1 and S_2 are 1.0, since they can be made so by appropriate scaling of $\mathbf{P}_1$ and $\mathbf{P}_2$. The first term on the right hand side of Equation 5.50 is a pure projective warp. Points near infinity are thus warped purely projectively, since the second term in the equation becomes insignificant when $z_1 \approx \infty$. For these points at infinity, a coherent traversal of the reference image during warping will guarantee coherent accesses to the destination image.

The second term on the right hand side of Equation 5.50 expresses the 3D warp's perturbation from a projective warp—in other words, the component of the 3D warp due to viewpoint translation. It is this term that results in partially incoherent accesses to the destination image during the warp, even if the reference image is being traversed coherently. To examine this term alone, we can re-express Equation 5.50 as follows:

$$\left(\frac{z_2}{z_1}\right)\bar{u}_2 = \tilde{u}_2 + \frac{1}{z_1}\mathbf{P}_2^{-1}\left(\dot{C}_1 - \dot{C}_2\right), \tag{5.51}$$

where $\tilde{u}_2$ is the location of $\bar{u}_1$ in the destination image due to a pure projective warp ($\tilde{u}_2 = \mathbf{P}_2^{-1}\mathbf{P}_1\bar{u}_1$).

The perturbation to the projective warp is proportional to both the reciprocal of depth, $\frac{1}{z_1}$, and the magnitude of the inter-viewpoint distance, $\left\|\dot{C}_1 - \dot{C}_2\right\|$. Thus, bounds on these factors will limit the deviation from a projective warp. We would ultimately like to determine the maximum deviation as a value p, expressed in units of destination-image pixels. I define $p = \max\left(\|\bar{u}_2 - \tilde{u}_2\|\right)$, where the magnitude is taken in image space (after the homogeneous division).

Before calculating the image-space quantity p, I will calculate a corresponding bound in 3-space. Let ϕ represent the angle between the world-space vectors $\mathbf{P}_2\tilde{u}_2$ and $\mathbf{P}_2\bar{u}_2$. Note that ϕ is independent of the projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ (the multiplication by $\mathbf{P}_2$ cancels out), but p is not.

If T is the viewpoint translation distance ($T = \left\|\dot{C}_1 - \dot{C}_2\right\|$), and d is the distance from $\dot{C}_1$ to an object in the scene, then we can see from Figure 5.6 that

$$\phi = \sin^{-1}\left(\frac{T\sin(\alpha)}{\sqrt{d^2 + T^2 - 2dT\cos(\alpha)}}\right). \tag{5.52}$$

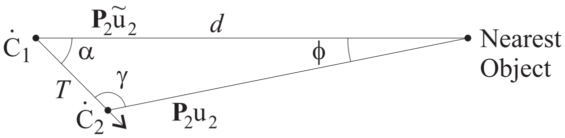


Figure 5.6: *Angular object movement across the field-of-view is a function of initial object distance and the distance between reference-image and destination-image viewpoints. The distance to the object is specified by d, and the distance between image viewpoints is specified by T.*

For a given d and T, ϕ is maximized when $\gamma = 90°$. This worst case gives the simpler expression

$$\phi = \sin^{-1}\left(\frac{T}{d}\right), \tag{5.53}$$

and therefore,

$$\phi_{max} = \sin^{-1}\left(\frac{T_{max}}{z_{min}}\right). \tag{5.54}$$

For a planar destination image $\mathbf{P}_2$, the maximum translational distance in pixels, p, can be computed from ϕ_{max} if the worst-case (largest) number of pixels per radian, a_{max} is known:

$$p = a_{max} \cdot \phi_{max}. \tag{5.55}$$

For on-center projections $\mathbf{P}_2$, this worst case occurs at the corners of the display. Using the display parameters defined in section 5.2.1 (but assuming square pixels and $destxres > destyres$),

$$a_{max} = \frac{destxres}{2}\cot\left(\beta_{dest,x}\right)\left[\left(\left(\frac{destxres}{destyres}\right)^2 + 1\right)\left(\frac{1-\cos\left(2\beta_{dest,x}\right)}{1+\cos\left(2\beta_{dest,x}\right)}\right) + 1\right], \tag{5.56}$$

which as $\beta_{dest,x} \to 0$, approaches $\frac{destxres}{2\beta_{dest,x}}$ as expected.

Display	vFOV ($2\beta_{dest,x}$)	T_{max}	z_{min}	p
640 x 480	60°	0.5m	2.0m	202
640 x 480	60°	0.1m	1.0m	80
1280 x 1024	60°	0.1m	1.0m	165

Table 5.1: *Worst-case screen-space movement of objects due to viewpoint translation.*

Table 5.1 shows values of p calculated for several different sets of conditions using the equations above.

The maximum destination-image movement, p, can only occur when several specific conditions are satisfied. One condition is that $\alpha \approx 90^\circ$, i.e. the pixel is far away (in destination-image space) from both destination-image epipoles. For the maximum movement to occur, the pixel must also be in a corner of the destination image. For illustration purposes, most of the figures in this section show the maximum movement distance p even when these conditions do not hold. I always use p in my cache-size calculations, because I am interested in the *worst-case* cache size.

Figure 5.7 illustrates what happens when a 3D warp is applied to a *line* in the reference image. The points on the reference-image line can map to anywhere in an *area* in the destination image. In my cache size calculations, I use a worst-case rectangular estimate of this area.

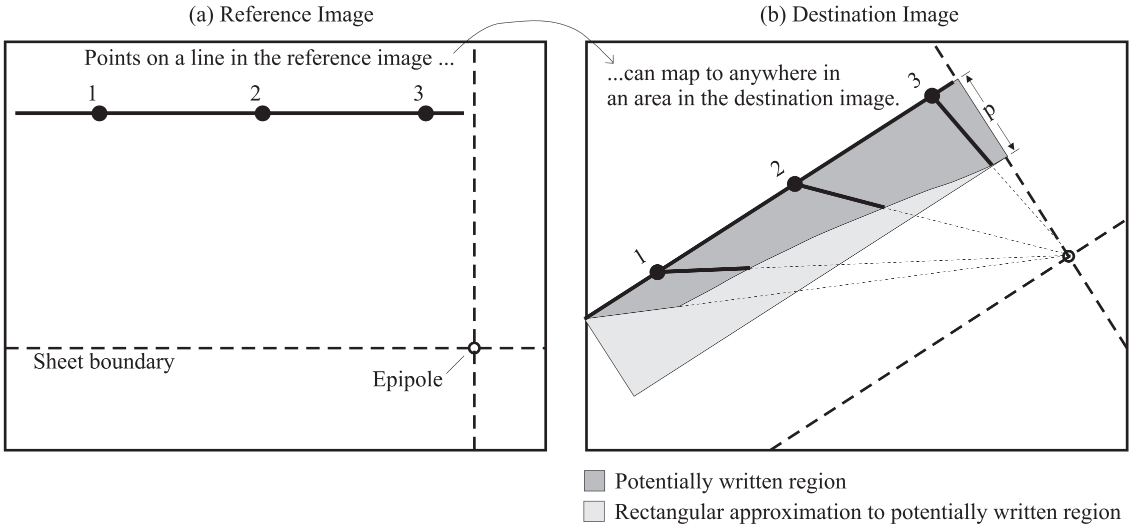

(a) Reference Image
(b) Destination Image
Points on a line in the reference image ...
...can map to anywhere in an area in the destination image.
1
2
3
Sheet boundary
Epipole
p
▩ Potentially written region
□ Rectangular approximation to potentially written region

Figure 5.7: *3D warp of a line. The points on a reference-image line can map to anywhere in an area in the destination image. My analysis approximates this area with a worst-case rectangular region.*

Note that Figures 5.5 and 5.7 show an approximately 1-to-1 scaling between reference-image pixels and destination-image pixels. Because the fields of view and resolutions of the reference and destination images can be different, other scalings are possible. My equations make no assumptions about the scaling.

5.3.2 Choosing a traversal pattern

To implement the 3D warp's memory accesses inexpensively, we should achieve three goals:

- A small destination-image working-set size, and thus a small required cache size.
- A slowly changing destination-image working set, thus minimizing cache-to-main-memory traffic. Equivalently, most (or all) destination-image pixels should enter and exit the destination-image working set only once per warp.
- A large size for each cache-to-main-memory transfer (especially important for block-transfer memories like RAMBUS [Rambus97]).

The success or failure in achieving these goals is primarily determined by the choice of a traversal pattern for the reference image. Each reference-image pixel must be warped once, but we have considerable freedom in choosing the order in which to warp these pixels. For any traversal order that we consider, we need to perform a worst-case memory-access analysis based on the maximum pixel-translation value p.

Cache blocks

One of the goals stated above is to keep the size of cache-to-main-memory transfers large. If we are to meet this goal while still transferring only pixels that are needed, we may not want use a raster organization for storing the destination image in main memory. For any particular warp, a raster organization is inefficient for those portions of the destination image where the epipolar lines are nearly perpendicular to the raster lines. In these portions of the destination image, a single reference-image pixel could map to any one of many scattered memory locations. An alternative is to use a tiled memory organization—the destination image is organized into square tiles, with all pixels from each tile stored in adjacent locations in main memory (Figure 5.8). These tiles are most naturally chosen to be the same size as a cache block. Thus, two pixels which are near each other in the destination image are likely to reside in the same cache block. The drawback to this tiled memory organization is that it requires a larger cache for video scan-out.

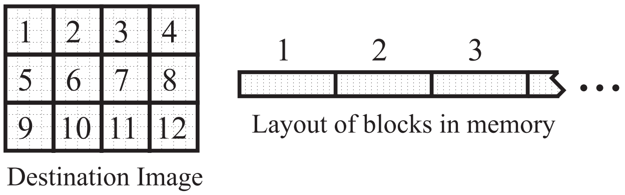


Figure 5.8: *Alternative memory layout for the destination image. Pixels belonging to 2D tiles of the destination image are stored contiguously in memory.*

To improve clarity, much of my discussion will not explicitly deal with the fact that cache blocks are greater than one pixel in size. I do however implicitly assume this fact by arguing that the entire area of the destination image that is *potentially* touched by pixels warped from a given portion of the reference image must be considered to be *actually* touched in the worst case. For cache blocks of one pixel, this assumption would not hold, since in general not all potentially-touched pixels can be actually touched. The reason is that, for a given portion of the reference image, there are more potentially-touched destination-image pixels than warped pixels. However, with block sizes larger than one pixel, only one of the pixels in the block needs to be actually touched in order to consider the entire block to be touched. Rapidly varying disparity values in the reference image can thus cause all potentially-touched blocks to contain at least one pixel that is actually touched.

Ideal traversal for warp

Given the memory access properties of the 3D warp that I have discussed, it is possible to describe an ideal reference-image traversal pattern. This ideal pattern traverses the reference image in a regular manner that optimizes the use of the destination-image cache.

This ideal traversal warps one reference-image epipolar line at a time. From Chapter 1 we know that the 3D warp maps points on a reference-image epipolar line to points on a corresponding destination-image epipolar line. Without knowing the depth of the reference-image points, we do not know to *where* on the destination-image epipolar line the points will map, but we do know that they will map to *some* point on the epipolar line. Thus, at any one time, the destination-image working set consists of the pixels on the destination-image epipolar line that corresponds to the reference-image epipolar line being warped. If we know the maximum image-space translation distance p, then the working set can be further restricted to a segment of the destination-image epipolar line of length p. Thus, the cache size is $O(p)$. The epipolar-line traversal pattern is illustrated in Figure 5.9.

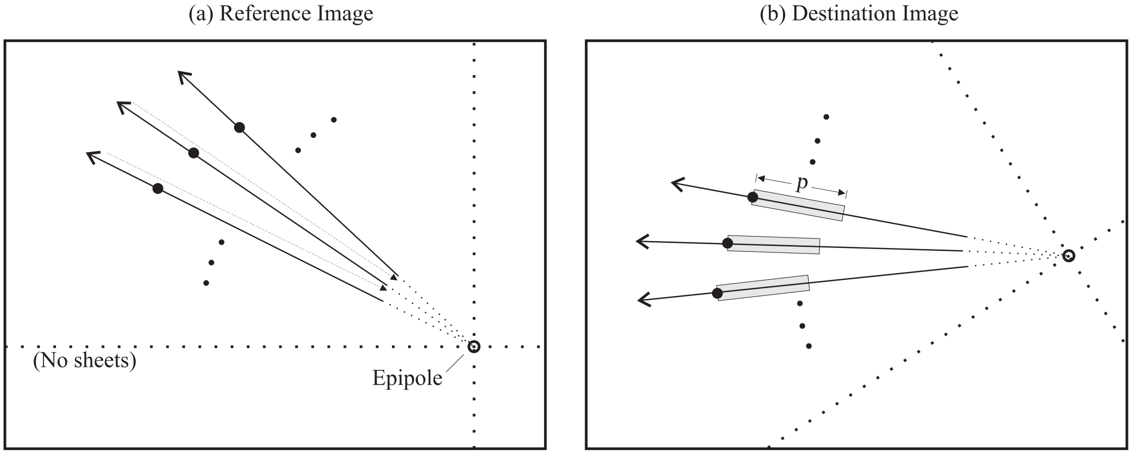


Figure 5.9: *Epipolar-line traversal. The destination image shows the potentially-written areas corresponding to the indicated reference-image points.*

The epipolar-line traversal pattern satisfies the goal that each destination-image pixel should enter and exit the cache exactly once. When the warp of a particular reference-image epipolar line is complete, the corresponding destination-image epipolar line can be removed from the cache, and it will never need to be reloaded.

The discrete nature of the reference and destination images introduces some complications into this strategy. In particular, the algorithm must insure that each reference-image pixel is only warped once. Since the reference-image epipolar lines get "closer" to each other near the reference-image epipole, some pixels must be skipped in each epipolar line as the epipole is approached. A more practical algorithm would warp blocks of reference-image pixels in an approximation of the epipolar line traversal. The destination-image cache then needs to be big enough to hold a "fat" epipolar line. Because destination-image pixels are grouped in blocks, we must hold an entire "fat" epipolar line in the cache. Holding only the subset of the epipolar line indicated by the value p is insufficient if we wish to avoid reloading pixel blocks.

A detailed calculation of the cache size required to hold a fat epipolar line is quite messy. It depends on a number of details, including the geometry of the pixel blocks in the reference and destination images and the strategy for loading them from memory. So, I will only provide an estimate of the cache size. I assume square, 16-pixel blocks in both images; an image width greater than the image height; destination-image pixels that require 12 bytes of storage (see Appendix C); and a one-to-one scaling from source to destination image. A Q-pixel-wide line can touch up to three QxQ pixel blocks in its "narrow" direction. So, the width of the fat line is three blocks. The length of the fat line (with the measurement made along the x or y axis) can be as much as *destxres* pixels, if we assume that $destxres > destyres$. So, the worst-case fat-line cache size is 3 block-widths $\cdot$ 4 pixels/block-width $\cdot$ *destxres* pixels. For a 640x480 destination image, this works out to approximately 92 KBytes.

Ideal traversal for hole fill

If a post-rendering warping system implements the hole-filling algorithm described in Chapter 4, then it must do so by making an occlusion-compatible traversal of the destination image. The simplest way to perform this traversal is to make a separate pass through the destination image after each reference-image warp. A destination-image cache is required for this traversal.

Chapter 4 discussed two variants of an occlusion-compatible eight-sheet traversal of the destination image that are appropriate for hole filling. The first variant traversed one sheet at a time. The second variant traversed all eight sheets approximately simultaneously, by visiting them in a round-robin fashion. This second, simultaneous, variant avoids the possibility of introducing artifacts at sheet boundaries, unlike the first variant.

Unfortunately, the simultaneous eight-sheet traversal requires a cache size that is typically eight times as large as the non-simultaneous eight-sheet traversal (although only about four times as large when comparing worst cases). The working set for the simultaneous traversal is illustrated in Figure 5.10. The required cache size is approximately $6 \cdot 2(destwidth + destheight)$ pixels. For a 640x480 image with 12-byte pixels, this size is equal to 161 KBytes. The non-simultaneous traversal requires $6 \cdot destwidth$ pixels, or 46 KBytes. Again, these figures may vary some depending on the geometry of pixel blocks in the destination image.

Warp and hole fill together?

It seems wasteful to completely separate the warp and the hole fill, because this separation requires two separate passes through the destination image. An alternative is to choose the warp's reference-image traversal pattern so that it is compatible with the hole-filling algorithm. In such an approach, the hole-filling algorithm operates on destination-image pixels just before they are discarded from the destination-image warping cache. The approach works because the discarding of pixels can be done in a destination-image occlusion-compatible order, as required by the hole-filling algorithm.

This approach requires an eight-sheet occlusion-compatible traversal in the reference image for the 3D warp. Unfortunately, for a worst-case reference image, this traversal requires a large cache.

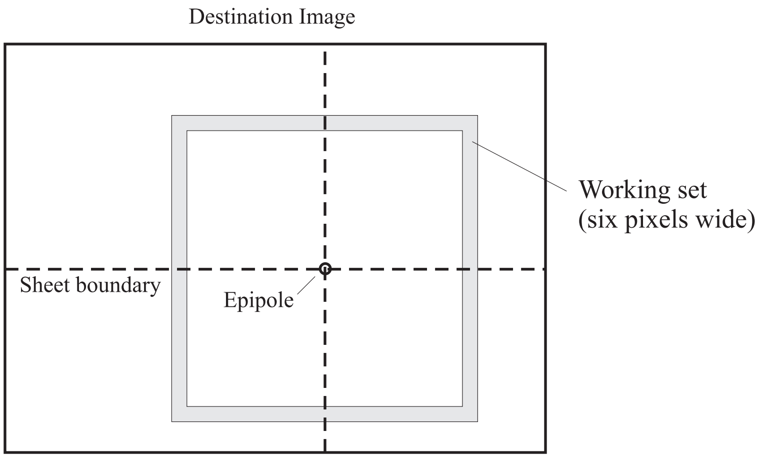


Figure 5.10: *Working set for the hole-filling algorithm, when using the approximately-simultaneous eight-sheet traversal of the image.*

Figure 5.11 illustrates the traversal order and its cache requirements. The cache must be able to hold the entire region of the destination image that is potentially touched by the pixels from a single scan line of the reference-image sheet. The reason for this requirement is that almost all of this potentially-touched region is revisited when the next scan line of the sheet is warped. The area of this region, and thus the minimum thrash-proof cache size, is $sheetWidth \cdot p$ pixels, where *sheetWidth* is measured in the destination image. Since a sheet can potentially occupy the entire destination image, the maximum sheet width is equal to $destdiag$, where $destdiag$ is the diagonal size of the destination image. Therefore the required destination-image-cache size is approximately $destdiag \cdot p$ pixels. For a 640x480 image with about 12 bytes per pixel, and $p = 100$, this works out to 960 KBytes of cache. This cache is large enough to be quite expensive, especially since it is intended to be a first-level cache.

Traversal discussion

The large cache size required by the one-pass warp/hole-fill traversal makes it unattractive. I also believe that implementing this traversal would be complicated, because of the need to run the hole-filling algorithm on parts of the image just before they leave the cache.

Another alternative, which I have not discussed here, is to use a two-pass occlusion-compatible traversal to perform the 3D warp. This traversal, described in [Mark97a], uses small caches (7 KBytes for a 640x480 image, neglecting pixel-blocking effects). One might suppose that hole-filling could be performed on pixels as they left the cache for the second time. But, although this traversal is occlusion compatible, it would not adequately insure that most precursor pixels are ready when needed for the hole filling. The algorithm is also quite complicated.

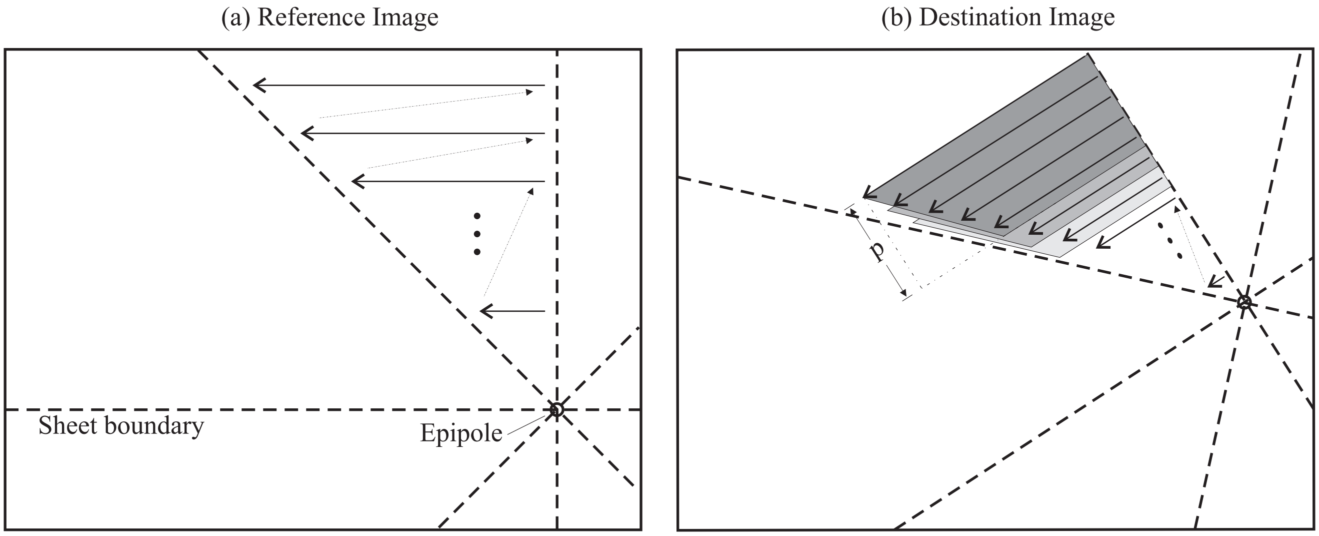


Figure 5.11: *Non-simultaneous eight-sheet occlusion-compatible traversal for the 3D warp. The potentially-written areas in the destination image that are produced by different reference-image scan lines overlap almost completely. I show these areas using their worst-case rectangular approximations, with the edges cut off slightly to allow the overlap to be more easily seen.*

Therefore, I believe that the best approach is to use one pass through the destination image for the warp, followed by a second pass for the hole filling. The warping pass uses the epipolar traversal, and the hole-filling pass uses the eight-sheet occlusion-compatible traversal. Both of these traversals were discussed earlier in this chapter. For a 640x480 destination image, this approach requires approximately 92 KBytes of cache for the first pass, and 46-161 KBytes of cache for the second pass (depending on the variant of eight-sheet traversal that is used). These values are independent of the maximum translational movement distance p. Approximately 11 MBytes of destination-image memory transfers are required for a warp and hole-fill of a 640x480 destination image ($640{\cdot}480{\cdot}(6{+}12{+}12{+}6)$).

In discussing memory-access properties so far, I have assumed that the goal is to design a warper that can fit its entire working set into a cache. I have performed a worst-case analysis, so that the warper can provide guaranteed performance, regardless of the contents of the reference image. This type of worst-case analysis has another benefit as well: It enables the use of a software-managed cache, which is simpler to implement than a hardware-managed cache. My analysis of memory-access patterns (if completed in detail for a particular implementation) provides the information necessary to load a software-managed cache with exactly the right cache blocks at the appropriate time.

However, other types of warper designs are possible. One alternative is to use a smaller, hardware-managed cache. Because most reference images have good local coherence, such a system would typically have better performance than is indicated by a worst-case analysis. If guaranteed performance is not important, then simulations could be used to design such a system with good average performance.

I have assumed that it is important to use large-sized transfers between cache and main memory (on the order of 200 bytes). If this requirement is not important to obtain high bandwidth from a particular memory technology, then it might be possible to implement a warper that does not use a cache at all. However, such a warper would undoubtedly have to use a pixels-to-be-composited buffer to hide memory-access *latency*.

5.4 Hardware-oriented reconstruction and hole-filling algorithms

The hole filling and reconstruction algorithms described in Chapters 3 and 4 were designed to be efficiently implementable in hardware. I will briefly describe the hardware-friendly attributes of these algorithms here.

The first example of hardware friendliness is provided by the hybrid mesh/splat reconstruction algorithm. This algorithm uses a screen-space distance threshold for surface segmentation. This threshold insures a maximum size for the triangles used by this algorithm for color interpolation. In the system configuration used for the kitchen walkthrough sequence, all such triangles fit inside a 4x4 pixel grid. As a result, it is possible to use a fixed-footprint hardware rasterizer to rasterize these triangles (for example, a 4x4 Pixel-Planes-style SIMD array [Fuchs85]).

The reconstruction algorithm that I designed for use in conjunction with anti-aliasing is even simpler to implement in hardware. It saves computation costs because it requires no explicit color interpolation at all. The flat-shaded, axis-aligned rectangles that it uses for reconstruction are easily rasterized in hardware.

In some systems, the reference-image resolution is inadequate to achieve acceptable displayed-image quality by using axis-aligned rectangle reconstruction directly. In such cases, it might be possible to dice the reference image into finer pieces before using the axis-aligned rectangle reconstruction. The reference-image dicing does require interpolation, which adds some expense. However, this interpolation is axis-aligned in the reference image, so it is much cheaper than the displayed-image interpolation that is implemented by using triangles for reconstruction.

The hole-filling algorithm described in Chapter 3 provides another example of hardware friendliness. The algorithm performs blurred hole-filling with only a single pass over the destination image. The cost of the hole-filling does not grow as hole sizes increase. Most blurring algorithms do not share this property.

5.5 A-buffering for anti-aliasing

Super-sampled anti-aliasing for PRW requires a large amount of memory (and memory bandwidth), because both the reference and displayed images must be stored at super-sampled resolution ([Chen93] has previously made this observation about depth-based warping). To reduce these memory requirements, I have experimented with A-buffer storage [Carpenter84] for the reference and displayed images, as Max has done [Max96]. My A-buffer stores a maximum of three distinct surfaces per pixel. I chose to store three surfaces because no more than three surfaces can generically intersect at a single point. A generic intersection is one that persists after small perturbations.

My system stores a bit-mask with each of the three surfaces at a pixel to indicate which sub-pixels the surface represents. Depth information is used to merge sub-pixels into one of the three surfaces. More specifically, the depth and screen-space derivatives of depth are used to identify sub-pixels that belong to the same surface. Derivatives of depth must therefore be stored with each of the three surfaces at a pixel. This information is already needed in the reference image, but must be added to the displayed image. Max [Max96] does not use derivative information—his system uses constant Z across a pixel.

When the reference image is A-buffered, the results of the warp are not quite identical to those obtained without A-buffering, even if only a single surface is present at each reference-image pixel. The reason is that the destination-image pixel grid is not generally aligned with the reference-image pixel grid. So, even for a flat object in the scene, the sub-pixels that belong to a single reference-image pixel may belong to two (or even four) adjacent destination-image pixels. The color blending that occurs when consolidating reference-image sub-pixels into a single A-buffer surface thus incorrectly blends sub-pixels that will eventually belong to adjacent destination-image pixels. In practice, this effect does not seem to be a problem. It is equivalent to a slight blurring of the image (but the blurring only occurs *within* each surface, not between different surfaces).

My implementation of A-buffered super-sampling produces good results. The super-sampled video sequences on my demonstration videotape use this implementation of A-buffering, with a 3x3 coverage mask. However, there is one aspect of my current implementation of A-buffering which should be improved: The displayed image is not stored in A-buffered format between the warping and hole-filling stages. The displayed image is only stored in A-buffered format *after* the hole-filling is completed for each reference image. This strategy is only appropriate in a system that performs hole-filling in the same pass as warping, as pixels leave the displayed-image warping cache. Such a system can maintain its cache in expanded format, and change to A-buffer format when pixels were moved to or from main memory.

As discussed earlier, I now believe that separate warping and hole-filling passes provide the best cache performance. Thus, the displayed image should be stored in A-buffered format in between the warp and hole-fill stages (as well as after the hole-fill stage). I do not believe that there is any reason why this would be particularly difficult, but I have not implemented it at this time.

5.6 Clipping

In a PRW system, the reference images' field of view (FOV) is typically significantly larger than the destination image's FOV to allow for changes in view direction. As a result, for any particular warp much of the reference image will fall outside the destination-image FOV. By appropriately clipping the reference image prior to warping, we can avoid unnecessarily warping all of its pixels. This clipping requires an upper bound on the disparity value, δ_{max} (which corresponds to a lower bound on depth, $z_{min})$.

Taking the 3D warp from Equation 1.11, rearranging, and substituting $\vec{B} \equiv \dot{C}_1 - \dot{C}_2$, we get:

$$\bar{u}_2 = \frac{\delta(\bar{u}_2)}{\delta(\bar{u}_1)} \left[\mathbf{P}_2^{-1}\mathbf{P}_1\bar{u}_1 + \delta(\bar{u}_1)\mathbf{P}_2^{-1}\vec{B} \right] \quad (5.57)$$

Consider a scan-line in the reference image. This scan-line is represented by a range of values of $\bar{u}_1$, from $\bar{u}_{1,start}$ to $\bar{u}_{1,end}$. Points on this line that are at infinity ($\delta = 0$) also form a line in the *displayed* image's projective coordinate system. I have designated this line as q in Figure 5.12a. Points that are closer to infinity are perturbed from this line. Thus, the set of possible warped locations forms a parallelogram in the displayed image's projective coordinate system, as shown in Figure 5.12a. For simplicity, Figure 5.12 ignores the scale factor $\delta(\bar{u}_2)/\delta(\bar{u}_1)$.

The parallelogram formed by the possible locations of the warped points is clipped against the displayed-image view frustum, producing an arbitrary quadrilateral (Figure 5.12b). Next, the system must use the clipped parallelogram to compute the subset of the reference-image scan-line which needs to be warped. To compute this subset, each vertex of the quadrilateral is projected back to the line q, using the vector $\delta_{max}\vec{B}$ to perform the projection (Figure 5.12c). The extremal projected vertices on the line q define the portion of the reference-image scan-line which must be warped.

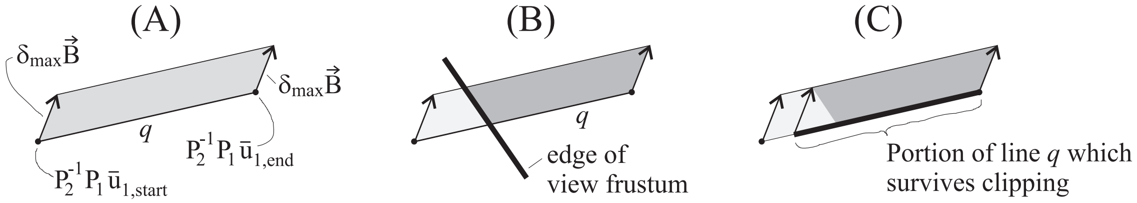

(A)
(B)
(C)
$\delta_{max}\vec{B}$
$\delta_{max}\vec{B}$
q
$P_2^{-1} P_1 \bar{u}_{1,end}$
$P_2^{-1} P_1 \bar{u}_{1,start}$
q
edge of view frustum
Portion of line q which survives clipping

Figure 5.12: *Clipping for the 3D warp. (a) Possible locations of warped pixels from a reference-image scan line. These locations form a parallelogram in the displayed image's projective coordinate system. (b) The parallelogram is clipped against the displayed image's view frustum. (c) The clipped vertices are then used to determine the extent of the reference image line which must be warped.*

I implemented this clipping technique in the PRW remote display system described in the next chapter (and in [Mark96]), with one further optimization. The server half of the remote display system determines the value δ_{max} independently for each reference image line, rather than for the image as a whole. This per-line δ_{max} allows tighter clipping than an per-image δ_{max}. The server computes the δ_{max} values by examining each reference-image scan line after rendering is complete.

5.7 Summary

The designer of a PRW system has to decide which representations, algorithms, and optimizations to use. This chapter provides a starting point for making these decisions, by discussing desirable attributes of image warping relative to normal polygon rendering. In particular, I showed that the 3D warp's transformation equations can be formulated as a fixed-point computation. I also discussed the memory access properties of the 3D warp, and proposed a cache-friendly reference-image traversal. Finally, I summarized the desirable features of my reconstruction algorithms, and described my A-buffering and reference-image clipping algorithms.

CHAPTER 6

REAL-TIME REMOTE DISPLAY

Consider a system which renders images at one location, and displays them at another location for a user who controls the viewpoint. The 3D warp can be used to compensate for network latency in such a *remote-display system*, as shown in Figure 6.1. Without the 3D warper, a change in the user's viewpoint or view direction would take one network round-trip time (plus rendering time) before it is seen in the display. For transcontinental or satellite networks, this latency is significant, and has a lower bound imposed by the speed of light. But with the 3D warper added to the system, viewpoint and view direction changes can be processed locally, and seen immediately in the display.

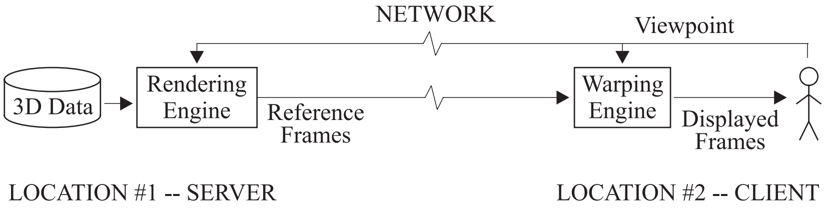


Figure 6.1: *Remote display system.*

This chapter discusses the use of post-rendering 3D warping for remote display. I begin by discussing some of the advantages of using the 3D warp in a remote-display system. Next, I describe the real-time remote display system that I built, and that Tom Hudson has since enhanced. Finally, I discuss possibilities for future improvement in these systems.

6.1 Advantages of 3D warping for remote display

A 3D warp-based remote display system capitalizes on many of the advantages of image-based rendering. The network bandwidth required by the system is independent of model complexity—it depends only on the displayed-frame resolution and the reference-frame update rate. The performance and memory of the 3D warper do not have to increase with scene complexity either. Thus, the 3D warper can function as a network client capable of displaying 3D models of arbitrary complexity. And, unlike systems such as those based on the Distributed Interactive Simulation (DIS) standard [Zyda93], the remote-display capability does not need to be integrated into the scene-graph level of an application.

If the remote-display system's reference-frame rate is less than the displayed-frame rate, then the 3D warp has two roles—to compensate for latency, and to increase the frame rate. However, if the reference frames are updated at the displayed-frame rate, then the 3D warp's sole role is to compensate for latency. In this situation, the remote warping system can display moving objects, deforming objects, and view-dependent lighting without these features jumping about at a (slower) reference-frame rate. For this reason, PRW is more generally applicable when it is used for remote display than when it is used for local rendering acceleration.

Although PRW can eliminate latency for user motions, it can not eliminate latency for changes to the scene. So, although moving objects will be appear correctly, they will appear in time delay. This latency for scene changes will not be noticeable unless the user is interacting with objects in the scene.

6.2 Real-time system

I have constructed a real-time system to demonstrate the use of PRW for remote display. More recently, UNC graduate student Tom Hudson has enhanced this system with new capabilities. I will first describe the original system ([Mark96, Mark97b]), then discuss Hudson's enhanced system ([Hudson98]). Both systems incorporate optimized software-based 3D warpers. Our timings of these warpers provide information about the performance that can be expected from software-based 3D warpers.

6.2.1 Original system

My remote display system consists of two Silicon Graphics Onyx computers connected via a 155 Mbit/sec ATM link. One computer is the rendering server, and the other is the warping client. The machines are physically adjacent to each other, so I simulate network latency by buffering data on the client computer.

The rendering server generates reference frames using SGI performer and an SGI RE/2 graphics subsystem. The reference frames are generated as groups of four depth images. The four images represent the four sides of a cube, and share in common the most recent viewpoint received from the client. The top and bottom of the cube are not rendered. Thus, each group of four images has a $360°$ horizontal FOV, and a $90°$ vertical FOV.

Each group of four reference images is transmitted uncompressed over the ATM network to the client. Each reference image has a resolution of 416 x 421 pixels. The images consist of RGB and δ at every pixel, as well as upper bounds on δ for each row of the image. The upper bound on δ is used by the client for clipping, as described in Chapter 5. It takes one second to render and transmit each group of images.

The client-side computer uses three 200 MHz MIPS R4400 processors to produce displayed images using software 3D warping. A fourth R4400 processor is used to receive and buffer data from the server. The client warps the most recently received group of four reference images to produce the displayed image. Warping the group of four reference images is only slightly more expensive than warping just a single reference image, because the system use the view-frustum clipping algorithm described in Chapter 5, which is very effective. The warper runs at 7 displayed frames/sec when producing displayed images of one-half NTSC resolution (320x240 pixels) [1]. The warper's reconstruction technique uses splats computed from per-pixel normal vectors.

By adding the appropriate position offsets for each eye to the displayed-image viewpoint, this system can produce stereo imagery for a head-mounted display. Producing stereo imagery does not require any additional information from the rendering server, but the displayed-frame rate drops to 3.5 frames/sec.

This system warps only one group of reference images to produce each displayed frame. These reference images all share a single center of projection, so in effect they function as one reference frame with a very large FOV. Because only one reference-frame viewpoint is used, the system is very prone to visibility artifacts. Whenever the user is moving, the reference frame viewpoint does not match the displayed frame viewpoint, and visibility artifacts appear. The system does not use any type of hole filling, so the visibility artifacts resemble large shadows cast from a light at the reference-frame viewpoint. One of the Tom Hudson's enhancements to the system, described in the next section, is to warp images from more than one reference-frame viewpoint.

As I mentioned earlier, the system uses three processors to perform the image warping. I experimented with two approaches to parallelization. The first approach used McMillan's occlusion-

[1] The displayed-image resolution is actually a hybrid between 320x240 and 640x480. The final resolution is 640x480, but the corresponding portion of the reference image (the visible region) has approximately a 320x240 resolution. Most of the warping cost is determined by this 320x240 resolution.

compatible warping order (see Chapter 1). The system assigned different occlusion-compatible sheets to different processors. Unfortunately, this approach sometimes resulted in poor load balancing, when one or two sheets required almost all of the computation. Popescu *et al.* have since developed a occlusion-compatible warping algorithm that has better load-balancing properties [Popescu98].

My second approach to parallelization uses Z-buffering to resolve visibility. With this approach, the reference image can be divided into equal-sized pieces, so that good load-balancing is achieved. Strictly speaking, an atomic read/modify/write should be used for accesses to the color and Z buffers, but neglecting to use such atomic accesses produces no visible artifacts.

6.2.2 Tom Hudson's enhanced system

Hudson [Hudson98] enhanced my original system in two major ways. First, his system produces displayed frames by warping reference frames from more than one reference-frame viewpoint. This change greatly reduces visibility artifacts. Second, Hudson's system uses newer hardware, and thus achieves better warping performance.

Hudson's system always warps three groups of reference images. As in my system, the reference images in a single group share a common viewpoint and form four sides of a cube. However, in Hudson's system, each of the three groups has a different reference-image viewpoint. In effect, his system warps three $360°$ horizontal FOV reference frames located at different viewpoints. The system attempts to maintain these reference-frame viewpoints in a triangle configuration surrounding the user's viewpoint (see [Hudson98] for more details). The plane containing the triangle is parallel to the ground plane, and at the same height as the user's eye. This reference-viewpoint configuration is a generalization of my point-on-line condition (discussed in Chapter 3) to a point-in-plane condition.

Hudson's system uses a four-processor SGI Onyx2 (195 MHz MIPS R10K processors) as the warping client. Three processors perform warping while the fourth communicates with the rendering server. The warper produces six displayed frames/sec. Because this warper uses three reference-frame viewpoints rather than one, this frame rate represents an approximately three-fold improvement in warping performance over my system. The displayed frames and reference frames are the same resolution that they are in my system.

6.3 Discussion

The performance of the software warpers used in these remote display systems is not yet adequate. Minimum performance in a remote display system should be approximately 12 displayed frames/sec at 640x480 resolution. I believe that achieving this performance requires approximately a four-fold increase in computational performance (it is not eight-fold, because we could regain approximately a factor of two improvement from the combination of the smaller splat sizes required for a true 640x480 warp and a switch to a fixed-size splat). An additional factor of 2.5 in CPU and memory performance (a factor of 10 overall) would be required to reach 30 displayed frames/sec.

For a 30 Hz *local* PRW system that uses two reference frames and only a single CPU for warping, at least a 20-fold per-CPU performance improvement is required. The factor of 20 is probably too small, since it neglects the cost of hole-filling, and the need to use some CPU time for generating the reference frames. Thus, if we rely on software-based warpers, Moore's Law indicates that it will be about five years before *local* PRW is a viable rendering-acceleration technique for single-CPU machines, even at 640x480 resolution.

6.4 Summary

This chapter described the application of PRW to the problem of remote display of rendered images. I described two such real-time PRW remote-display systems. The first system was built by me, and the second was built by Tom Hudson. The performance of these systems indicates that single-CPU local PRW at 30 displayed frames/sec (with 640x480 resolution) will not be achievable for about five years.

CHAPTER 7

DISCUSSION AND CONCLUSION

This chapter concludes this dissertation. The chapter begins by discussing the viability of post-rendering warping as a rendering-acceleration technique. Part of this discussion lists the characteristics that an application should satisfy in order to benefit from PRW. Next, I summarize this dissertation's results. Finally, I describe some possible directions for future PRW research.

7.1 Viability of post-rendering warping

I am convinced, based on the work discussed in this dissertation, that building a real-time post-rendering warping system that produces good quality imagery is *possible*. But, there is another important question – is it *desirable* to use post-rendering warping both now and in the future? The answer depends in part on characteristics of the particular application under consideration, and in part on more general trends in CPU, display, and graphics hardware. The answer also depends on whether post-rendering warping is used for rendering acceleration, for latency compensation, or for both purposes.

7.1.1 Application characteristics

Certain types of applications are not appropriate for post-rendering warping. If the majority of objects in the scene are deforming, moving, or appearing/disappearing, then post-rendering warping will not be an effective acceleration technique. Applications that extensively use highly view-dependent lighting such as reflection maps are also inappropriate for post-rendering warping, especially if the objects that are view-dependently lit are flat or have low surface curvature. In contrast, applications with entirely static, diffusely-illuminated scenes are the best fit for post-rendering warping. However, a PRW system can accommodate applications with small numbers of deforming, moving, or highly specular objects if the application can be modified to render these objects in a separate pass for each displayed frame.

Applications with rapid changes in view direction (e.g. rapid rotation of a head-mounted display), benefit less from rendering acceleration by PRW, especially if occasional clipping of the displayed image is unacceptable. The reason is that larger rotation rates require larger FOV's for the PRW system's reference frames. Oversized reference frames are expensive to render and to store, thus reducing the performance gain from post-rendering warping. This problem might be alleviated by real-time adjustment of the FOV of the reference frames, based on prediction of future view-direction rotation rates. If the goal of PRW is latency compensation rather than rendering acceleration, then rotation rates are not as important.

For similar reasons, applications which require a 30-60 Hz frame rate are better suited to rendering acceleration by PRW than applications for which a 10-15 Hz displayed-frame rate is acceptable. At lower displayed-frame rates, reference frames are generated less often (2-3 Hz) and thus must have greater over-sizing to account for changes in view direction. Furthermore, as the reference-frame rate drops, the user's motion between reference frames becomes less linear, and the motion prediction needed to choose reference frames becomes less accurate. As a result, visibility artifacts are worse. Applications requiring (or improved by) a 60 Hz or greater frame rate benefit the most from post-rendering warping.

Applications with an *extremely* low *reference*-frame rate can also benefit from rendering acceleration by post-rendering warping. Once the reference-frame rate is substantially below 1 Hz, it becomes necessary to render a full 4π steradian set of reference frames. There is then no further penalty for a further decrease in the reference-frame rate. However, such applications are likely to suffer from more severe visibility artifacts, even if four or more reference frames are warped to produce each displayed frame.

Good predictability of viewpoint motion is another important characteristic of applications that are suitable for PRW acceleration. Good predictability is a function of both the quality of the motion prediction sub-system, and of the typical motion patterns associated with the application. It is clear from my experiments that the quality of motion prediction is a crucial factor in determining the severity of visibility artifacts generated by a two-reference-frame PRW system.

Applications which are very intolerant of certain types of reconstruction and aliasing artifacts may not be suitable for rendering acceleration by PRW, especially if the PRW system uses a simple reconstruction algorithm. For example, flight simulators require very accurate display of distant aircraft and point-source lights in order to avoid providing cues to the pilot which are not present in the real world. Many PRW reconstruction algorithms will incorrectly enlarge or shrink such distant point-like objects. Such PRW systems would probably be unacceptable for flight simulators, unless the system directly rendered the troublesome objects into each displayed frame in a separate pass. However, if latency reduction is an important goal (as in remote display systems), then the latency-reduction benefits of PRW may outweigh the drawbacks of any artifacts.

7.1.2 Relationship to other rendering acceleration techniques

Post-rendering warping can be thought of as a simple and approximate method to automatically perform view-dependent geometry simplification and visibility culling. The reference frames represent the portions of the scene which are likely to be visible in the displayed frames, at approximately the correct level of detail (one reference-frame pixel per displayed-frame pixel). The relationship of post-rendering warping to other acceleration techniques follows naturally from these characteristics.

If an application aggressively uses visibility culling and view-dependent simplification techniques, then PRW becomes largely redundant (for rendering acceleration). In this situation, PRW loses most of its computational advantage over the conventional rendering, although it still retains some memory-access-coherency advantages. In particular, PRW does not require texture map lookups. Conversely, PRW is most useful for applications which do not or can not use aggressive geometry simplification and visibility culling approaches. PRW is more likely to be used in these applications because of its simplicity—in particular, because it requires no integration with the scene-graph level of the application.

If PRW is implemented in software, then PRW may contend with conventional simplification and visibility techniques for CPU time. The application writer must decide whether the benefits of PRW justify the cost in CPU time. In making this decision, the indirect costs of using PRW must be considered. If PRW lowers the conventional rendering rate (due to stealing of CPU time), then the cost-per-frame of the conventional simplification techniques may increase. The reason is that some conventional simplification techniques (e.g. [Luebke97]) rely on frame-to-frame coherence. As the conventional rendering rate drops, this coherence is reduced. In essence, the coherence can not be used twice – it is either used by the PRW, or by the conventional simplification techniques, but not by both.

7.1.3 Software performance

Current PRW performance on desktop and workstation systems is inadequate for rendering acceleration. Current PRW performance is borderline for remote-display latency compensation, where lower frame rates and resolutions may be acceptable. I consider the minimum acceptable performance for rendering acceleration to be 30 Hz at 640x480 display resolution, for two reference frames. In Chapter 6, I argued that CPU performance needs to improve by at least a factor of 20 to achieve this performance on a single-CPU machine. An even larger improvement would be required in order to add anti-aliasing, increase the display resolution, or incorporate better reconstruction and resampling algorithms.

The performance of software image warpers will improve as CPU performance, including memory bandwidth, improves. However, display resolutions (both desktop and head-mounted) are also steadily increasing. Already, 1280 x 1024 displays are standard on desktop PC's. Since the cost of image warping is proportional to display resolution, larger displays require increased warper performance even for a fixed frame rate. It is not clear that CPU performance will improve quickly enough to simultaneously accommodate higher-resolution displays *and* increase the PRW frame rate to 30 Hz in the near future. As a result, the near-future use of software-based PRW may be restricted to applications for which relatively low resolution displays are acceptable.

7.1.4 Hardware outlook

The poor performance of software-based PRW led me to explore hardware acceleration of PRW, as discussed in Chapter 5. It is perhaps not surprising that algorithm-specific hardware acceleration would be required to obtain acceptable PRW performance. PC's already use algorithm-specific hardware to accelerate many common algorithms, including 2D window operations, polygon rasterization, and, more recently, polygon setup and transformation.

I have no doubt that hardware could be designed and built to efficiently accelerate PRW. By relying on the particular properties of PRW calculations, such hardware would have a much better cost/performance ratio than either the general-purpose CPU or the conventional polygon-rendering hardware.

An important question remains: Will anyone ever include such hardware in a commercial product (and should they)? More specifically, does the cost of the PRW hardware justify its inclusion in the product? History argues strongly against the likelihood of such inclusion—most ideas for hardware acceleration of complex graphics algorithms are never incorporated into mainstream products. The utility of the idea must outweigh its cost. An example of this difficulty is provided by the Talisman architecture [Torborg96]. Although the Talisman architecture implements a number of promising techniques, it does not currently appear that it will become a commercial product.

Given the limited set of applications for which PRW is well suited, I believe that PRW hardware will not be included in mainstream products in the near future, unless the same hardware also accelerates other useful operations. For example, such hardware might be used to accelerate other image-based rendering operations. In particular, hardware which could accelerate both PRW and image-based rendering of acquired scenes would be a strong candidate for inclusion in commercial products. Unfortunately, the precise form that such hardware should take will not become clear until image-based rendering algorithms for acquired imagery mature further.

7.1.5 Viability summary – Rendering acceleration

I will now summarize my thoughts about the viability of post-rendering warping as a rendering-acceleration technique. I do not believe that the cost of PRW hardware can be justified without support for other image-based-rendering techniques. Thus, in the near future, PRW will be implemented in software rather than in hardware. Applications which can gainfully use PRW will have the following characteristics:

- They can use a 640x480 pixel non-anti-aliased display.
- The view direction changes relatively slowly.
- There are few (or no) moving objects; low-curvature, highly-reflective objects; or deforming objects.
- Users' interactions with the scene do not need to be accelerated (this is a special case of the previous characteristic).
- Viewpoint motion can be accurately predicted.
- The quality of the application is improved by increasing the frame rate to 30 or 60 Hz.
- Minor reconstruction artifacts are tolerable.
- The application programmer wants to use an acceleration technique that is simple to implement and does not require changes at the scene-graph level of the application.

7.1.6 Viability summary – Latency compensation

For systems in which the primary purpose of PRW is latency compensation (for either local or remote display), some of the restrictions listed above no longer apply. That is because PRW does not have to provide an improvement in rendering performance in a latency-compensation system. Therefore, it is acceptable to render reference frames with very large fields-of-view. As a result, latency-compensation systems can be used with applications that do not necessarily benefit from a 30 or 60 Hz frame rate. Lower frame rates allow higher resolutions with constant software performance, so that resolutions higher than 640x480 are feasible.

If reference frames are not reused, and if the reference-frame rate matches the displayed-frame rate, then moving and deforming objects will be undistorted by the 3D warp. However, the 3D warp is still unable to hide the latency in movement or deformation. As long as this latency is acceptable, these objects may be included in the scene.

Therefore, applications that can gainfully use PRW for latency compensation should have the following (smaller) set of characteristics:

- The display is 640x480 pixels, *or* the frame rate is low.
- There are few (or no) highly reflective, low-curvature objects
- Minor reconstruction artifacts are tolerable.
- High latency for interaction with objects in the scene (e.g. modifications to objects or movement of objects) is acceptable.

7.2 Results

At a high level, the results from this dissertation are:

A. I have shown, using an off-line test-bed, that a PRW system can produce excellent-quality images. A major advantage of my PRW system is that it requires no integration with the scene-graph level of the application.

B. I have developed conceptual frameworks for studying the visibility and reconstruction problems encountered when warping multiple reference images. I have also developed specific algorithms to address these problems in the context of post-rendering warping.

C. I have shown that there are a number of properties of 3D warping that could be used to efficiently accelerate it in hardware.

D. I have characterized the advantages and disadvantages of PRW as a rendering-acceleration technique, and described the characteristics of applications most suitable for acceleration.

At a more detailed level, the major results from this dissertation are:

1. I demonstrated that two source images are sufficient to eliminate almost all visibility artifacts when accurate motion prediction is available. I explained this result theoretically by showing that visibility artifacts are eliminated when the point-on-line and single-occluder conditions are satisfied.

2. I developed a hole-filling algorithm with several desirable properties. The algorithm is efficient, requiring only a single pass for each warped reference image. The algorithm is based on well-stated assumptions, incorporates blurring, and produces good results.

3. I mathematically described the image-space behavior of visibility holes produced by the 3D warp for the case of two reference images.

4. I described and discussed the implications of the dependence of the reference-frame field of view on three variables: the rate of change in view direction, the reference-frame rate, and the maximum rotational prediction error.

5. I was among the first researchers to thoroughly study the visibility, sampling, and reconstruction issues associated with a 3D warp of more than one reference image. As part of this work, I developed a theoretical framework for the multiple-reference-image 3D warp reconstruction and resampling problem. This framework considers the problem in 3D, and shows that the key question in reconstruction is whether or not adjacent image-space samples belong to the same 3D surface. I showed that post-rendering warping using typical polygonal models has insufficient information available for perfectly correct reconstruction, and that the goal of a reconstruction algorithm should be to minimize perceived artifacts.

6. I developed and implemented a reconstruction and resampling algorithm for multi-reference-image 3D warping. The algorithm uses an image-space, view-dependent test to decide whether or not adjacent image-space samples belong to the same 3D surface.

7. I showed that the REYES flat-shaded micro-polygon strategy can be used for 3D warping reconstruction and resampling. This strategy uses super-sampled anti-aliasing to implicitly interpolate between flat-shaded regions associated with each sample.

8. I analyzed a variety of alternative reconstruction and resampling algorithms, and explained their advantages and disadvantages in terms of my theoretical framework. My normal-vector splat algorithm was the first to show that per-pixel normal vectors can be used to assist 3D warp reconstruction.

9. I showed that there are a number of properties specific to 3D warping that distinguish it from more general polygon rendering, and allow for more efficient hardware implementations. In particular, I showed that the 3D warp's memory accesses are partially coherent, so that 3D warping can be implemented using a software-managed cache. I also developed a fixed-point formulation of the 3D warp's transformation equations.

10. I showed how anti-aliasing for the 3D warp can be implemented using an A-buffer.

11. I developed clipping and parallelization algorithms for efficient real-time implementation of 3D warping.

12. I built a real-time remote-display system that uses 3D warping for latency compensation.

I believe that this dissertation's approaches to the visibility, reconstruction, and performance problems will prove to be useful for other applications of 3D warping, including the warping of depth images acquired from the real world. The algorithms developed in this dissertation are most useful for warping depth images that are both acquired and warped in real time. In such 100% real-time systems, there is no opportunity to extensively pre-process depth images.

7.3 Future work

There are a number of potential areas for future work that are particularly worthy of discussion. Some of these ideas have the potential to overcome important restrictions of post-rendering warping. Others are attractive on the surface but have potentially significant difficulties that I will discuss.

I have concentrated on a PRW approach that warps two reference frames to produce each displayed frame. This approach eliminates visibility artifacts for a single, convex occluder only if the point-on-line condition from Chapter 3 is satisfied. By warping *four* reference frames to produce each displayed frame, the point-on-line requirement could be eliminated. The key development needed to adopt this approach is an algorithm for incrementally choosing new reference-frame viewpoints. Hudson's work with three reference frames [Hudson98] provides a starting point.

There are, however, several important drawbacks to the four-reference-frame approach. First, it requires twice as much computation, storage, and memory bandwidth as the two-reference-frame approach. Second, it suffers from PRW's problem with high rotation rates. In a four-reference-frame system, each reference frame is used for twice as long as in a two-reference-frame system. Thus, the reference frames must be oversized more than they are in a two-reference-frame system. This problem might be overcome by designing such a system to accept some FOV clipping for the older reference frames during periods of fast rotation. During such periods, the system would temporarily become, in effect, a two-reference-frame system (but with differently chosen reference-frame viewpoints).

An important limitation of PRW is its inability to handle moving objects, unless the moving objects are separately rendered into each displayed frame. This restriction could be relaxed by associating motion information with each pixel in the reference frame. For example, each pixel could have an associated velocity and acceleration, as proposed by [Costella93]. The warp equations would be modified to incorporate the velocity and acceleration vectors. This approach would only work for objects with constant or nearly constant acceleration. Visibility artifacts would become more severe, because the point-on-line condition is not sufficient to eliminate visibility artifacts for scenes with moving objects. Finally, providing the velocity and acceleration data would require integration with the scene-graph level of the application.

Theoretically, post-rendering warping only produces correct results for diffuse surfaces. In practice, it also provides acceptable results for mildly specular surfaces. However, highly specular lighting effects, such as reflection maps, still jump about at the reference frame rate. One possible solution to this problem is to use deferred or partially-deferred shading [Whitted82, Deering88]. The reference frame contains shading parameters at each pixel rather than surface colors. These parameters are used to evaluate the shading function at each displayed-frame pixel.

I believe that deferred shading is becoming less attractive as time progresses. There is a pronounced trend in new rendering hardware towards increasingly complex shading calculations, with a corresponding increase in the number of shading parameters and in the computational effort devoted to shading. The large number of shading parameters unreasonably enlarges the reference frame's memory size. Deferring the expensive shading calculation also reduces the computational benefit of PRW, and makes the warper substantially more complicated, especially if texture-map lookups are deferred.

A number of important questions about post-rendering warping could be answered by building a real-time PRW system. Such a system could be tested with real applications, in order to evaluate the many application-dependent questions about the usefulness of PRW. With today's technology, a large multi-processor system (e.g. 16 or 32 processors) would be required to obtain adequate performance and displayed-frame resolution. Such a system could serve as a vehicle to evaluate the feasibility of restricting the head-rotation rate, and to assess the severity of visibility artifacts on task performance. A real-time system would provide the most useful results if it was coupled with a state-of-the-art user-motion prediction system. A motion-prediction system of this type was not available to me during my research, but is projected to be completed soon at UNC Chapel Hill by Hans Weber.

I have focused exclusively on using PRW to warp reference images of the entire scene. However, there is the potential to use 3D warping in conjunction with a layered system like Talisman. One of the major advantages of PRW—independence from the scene-graph management—would be forfeited, but problems with visibility artifacts would be reduced. I believe that many of the results in this dissertation could be applied to a Talisman-like system. However, the fact that Talisman has yet to succeed commercially casts some doubt on the commercial viability of any hardware-based layering system.

When algorithms for image-based rendering of acquired imagery have matured further, there will be an opportunity to design hardware to accelerate both these algorithms and PRW. I consider this area of research to be a particularly promising one.

7.4 Summary

For certain applications, post-rendering warping can be used to accelerate the display of complex models, with only minor degradation in image quality. Such applications must satisfy a number of characteristics, including high frame-to-frame coherence even at peak head-rotation rates. Today's CPU's do not provide sufficient performance to build inexpensive software-based post-rendering warping systems for rendering acceleration. Hardware acceleration of post-rendering warping has the potential to overcome this problem, but I believe that this hardware must also accelerate other image-based rendering tasks in order to be commercially viable. Performance is less of an issue when using PRW for latency compensation, because the benefits of latency compensation may be large enough that lower frame rates or lower resolutions are acceptable.

APPENDIX A

DERIVATION OF SOLID ANGLE FORMULA

In this appendix, I derive an expression for the solid angle subtended by a rectangle (i.e. image) as seen from a particular center of projection (i.e. viewpoint). This expression was used in Chapter 3. The rectangle is described by its horizontal and vertical fields of view. It is assumed that the rectangle is centered about the view direction, and that the image plane is orthogonal to the view direction.

I actually compute the angle subtended by one quadrant of the image, then multiple by four. The computation could thus be easily adjusted for images that are not centered about the view direction, by making separate calculations for each quadrant.

Solid angle is measured as area on the unit sphere surrounding the viewpoint. Thus, I compute the solid area by integration using spherical coordinates. The spherical coordinates (θ, ϕ, ρ) that I use are defined by the relations:

$$x = \rho \cos\theta \cos\phi, \qquad y = \rho \sin\theta \cos\phi, \qquad z = \rho \sin\phi \tag{A.1}$$

Assume that the image plane touches the unit sphere at the point $(x = 1, y = 0, z = 0)$, or equivalently, $(\rho = 1, \theta = 0, \phi = 0)$.

If we define u and v as image-plane coordinates, with the origin at the point where the image plane touches the sphere, then

$$u = \tan\theta \qquad v = \frac{\tan\phi}{\cos\theta} \tag{A.2}$$

Then,

$$QuadrantSolidAngle = \int_0^{RightSide} \int_0^{TopSide} dA \tag{A.3}$$

$$= \int_0^{RightSide} \int_0^{TopSide} \cos\phi \; d\phi \, d\theta \tag{A.4}$$

The *RightSide* bound, $u = u_{max}$ in image coordinates, remains fixed in spherical coordinates. The *TopSide* bound, $v = v_{max}$ in image coordinates, does not—it varies as a function of θ. This asymmetry in the behavior of the integration bounds is due to the asymmetry in the definition of the spherical coordinates. So,

$$QuadrantSolidAngle = \int_0^{\theta_{max}} \int_0^{\phi_{max}(\theta)} \cos\phi \; d\phi \, d\theta. \tag{A.5}$$

If we define $\alpha \equiv \frac{1}{2}\,HorizFOV$, then $\theta_{max} = \alpha$. We also define $\beta \equiv \frac{1}{2}\,VertFOV$. Then, from Equation A.2, $v_{max} = \tan\beta$. Also using Equation A.2,

$$\begin{aligned}\phi_{max}(\theta) &= \arctan\left(v_{max}\cos\theta\right) && \text{(A.6)}\\ &= \arctan\left(\tan\beta\cos\theta\right). && \text{(A.7)}\end{aligned}$$

So,

$$\begin{aligned}\frac{1}{4}\,SolidAngle &= \int_0^{\alpha}\int_0^{\phi_{max}(\theta)} \cos\phi \; d\phi \, d\theta && \text{(A.8)}\\
&= \int_0^{\alpha} \sin\phi|_0^{\phi_{max}(\theta)} \, d\theta && \text{(A.9)}\\
&= \int_0^{\alpha} \sin\left(\phi_{max}(\theta)\right)\, d\theta && \text{(A.10)}\\
&= \int_0^{\alpha} \sin\left(\arctan\left(\tan\beta\cos\theta\right)\right)\, d\theta && \text{(A.11)}\\
&= \int_0^{\alpha} \frac{\tan\beta\cos\theta}{\sqrt{1+\left(\tan\beta\cos\theta\right)^2}}\, d\theta && \text{(A.12)}\\
&= \tan\beta\int_0^{\alpha} \frac{\cos\theta}{\sqrt{1+\tan^2\beta\left(1-\sin^2\theta\right)}}\, d\theta && \text{(A.13)}\\
&= \frac{\tan\beta}{\sqrt{1+\tan^2\beta}}\int_0^{\alpha} \frac{\cos\theta}{\sqrt{1-\frac{\tan^2\beta}{1+\tan^2\beta}\sin^2\theta}}\, d\theta && \text{(A.14)}\\
&= \frac{\tan\beta}{\sqrt{1+\tan^2\beta}}\left[\frac{1}{\sqrt{\frac{\tan^2\beta}{1+\tan^2\beta}}}\arcsin\left(\sin\theta\sqrt{\frac{\tan^2\beta}{1+\tan^2\beta}}\right)\right]_{\theta=0}^{\theta=\alpha} && \text{(A.15)}\\
&= \left[\arcsin\left(\sin\theta\sin\beta\right)\right]_{\theta=0}^{\theta=\alpha} && \text{(A.16)}\\
&= \arcsin\left(\sin\alpha\sin\beta\right) && \text{(A.17)}\end{aligned}$$

The integral above (Equation A.14) appears on page 162 of [Gradshteyn80], a table of integrals. In summary,

$$\boxed{SolidAngle = 4\arcsin\left(\sin\left(\tfrac{1}{2}\mathrm{HorizFOV}\right)\sin\left(\tfrac{1}{2}\mathrm{VertFOV}\right)\right)} \tag{A.18}$$

APPENDIX B

LINEARIZATIONS FOR SECTION 3.3

This appendix derives several linearized expressions that are used in Section 3.3.

In several places, we make use of a first order approximation of the following form (where H is any desired expression):

$$\frac{1}{H+dH} \approx \frac{1}{H} - \frac{dH}{H^2}, \quad \text{for } |dH| \ll |H|. \tag{B.1}$$

From this approximation, we can derive a second approximation, in which H and G are any desired expressions:

$$\frac{G+dG}{H+dH} \approx \frac{G}{H} + \frac{dG}{H} - \frac{G \cdot dH}{H^2} - \frac{dG \cdot dH}{H^2}, \quad \text{for } |dH| \ll |H|. \tag{B.2}$$

B.1 Linearization of diverging epipoles expression

In this section, I will linearize the expressions for the image-space locations of diverging epipoles. The epipoles diverge from a common location as the point-on-line condition is violated (i.e. as $\vec{d}$ becomes non-zero). This perturbation of the epipoles away from their initial coincident location is described by Equations 3.26, repeated here:

$$\begin{aligned} e_{u,2A} &= \frac{b'_x + \frac{\beta}{t}\hat{d}'_x}{b'_z + \frac{\beta}{t}\hat{d}'_z} & \qquad e_{v,2A} &= \frac{b'_y + \frac{\beta}{t}\hat{d}'_y}{b'_z + \frac{\beta}{t}\hat{d}'_z} \\ e_{u,2B} &= \frac{b'_x - \frac{\beta}{1-t}\hat{d}'_x}{b'_z - \frac{\beta}{1-t}\hat{d}'_z} & \qquad e_{v,2B} &= \frac{b'_y - \frac{\beta}{1-t}\hat{d}'_y}{b'_z - \frac{\beta}{1-t}\hat{d}'_z} \end{aligned} \tag{3.26}$$

If the perturbation of the epipole (due to $\vec{d}$) described these equations is small, we can apply approximation B.2 to Equations 3.26. In doing so, the $|dH| \ll |H|$ assumption becomes $\left|\frac{\beta}{t}\hat{d}'_z\right| \ll |b'_z|$, or $\left|\frac{\beta}{1-t}\hat{d}'_z\right| \ll |b'_z|$. This assumption becomes invalid as the destination-image center of projection approaches a source image center of projection, causing the $\frac{1}{t}$ or $\frac{1}{1-t}$ factor to explode. Otherwise, the assumption is valid for any image field-of-view significantly less than 180 degrees, for the following reason: If the epipoles are to appear in the image, then $\vec{b}'$ must have a significant z component. Since $\|\vec{d}\| \ll \|\vec{b}\|$, the significant z component means that $\|\vec{d}\| \ll |b'_z|$, and thus that $|\beta\hat{d}'_z| \ll |b'_z|$. Applying the approximation to equations 3.26 gives:

$$\begin{aligned}
e_{u,2A} &\approx \frac{b'_x}{b'_z} + \frac{\frac{\beta}{t}\hat{d}'_x}{b'_z} - \frac{b'_x\frac{\beta}{t}\hat{d}'_z}{(b'_z)^2} - \frac{\left(\frac{\beta}{t}\right)^2\hat{d}'_x\hat{d}'_z}{(b'_z)^2} \\
e_{v,2A} &\approx \frac{b'_y}{b'_z} + \frac{\frac{\beta}{t}\hat{d}'_y}{b'_z} - \frac{b'_y\frac{\beta}{t}\hat{d}'_z}{(b'_z)^2} - \frac{\left(\frac{\beta}{t}\right)^2\hat{d}'_y\hat{d}'_z}{(b'_z)^2} \\
e_{u,2B} &\approx \frac{b'_x}{b'_z} - \frac{\frac{\beta}{(1-t)}\hat{d}'_x}{b'_z} + \frac{b'_x\frac{\beta}{(1-t)}\hat{d}'_z}{(b'_z)^2} - \frac{\left(\frac{\beta}{1-t}\right)^2\hat{d}'_x\hat{d}'_z}{(b'_z)^2} \\
e_{v,2B} &\approx \frac{b'_y}{b'_z} - \frac{\frac{\beta}{(1-t)}\hat{d}'_y}{b'_z} + \frac{b'_y\frac{\beta}{(1-t)}\hat{d}'_z}{(b'_z)^2} - \frac{\left(\frac{\beta}{1-t}\right)^2\hat{d}'_y\hat{d}'_z}{(b'_z)^2}.
\end{aligned} \tag{B.3}$$

The fourth term in each of these approximating expressions is a second-order term that can be dropped. The reason is the same one stated earlier, the expectation that $|\beta\hat{d}'_z| \ll |b'_z|$. Since for similar reasons we also expect that $|\beta\hat{d}'_x| \ll |b'_z|$ and $|\beta\hat{d}'_y| \ll |b'_z|$, the fourth term is always a second order term. Even when $b'_x = 0$ or $b'_y = 0$, which *can* occur, the second term in these equations will dominate the fourth term. Incorporating this approximation, the equations are:

$$\begin{aligned}
e_{u,2A} &\approx \frac{b'_x}{b'_z} + \frac{\frac{\beta}{t}\hat{d}'_x}{b'_z} - \frac{b'_x\frac{\beta}{t}\hat{d}'_z}{(b'_z)^2} & e_{v,2A} &\approx \frac{b'_y}{b'_z} + \frac{\frac{\beta}{t}\hat{d}'_y}{b'_z} - \frac{b'_y\frac{\beta}{t}\hat{d}'_z}{(b'_z)^2} \\
e_{u,2B} &\approx \frac{b'_x}{b'_z} - \frac{\frac{\beta}{(1-t)}\hat{d}'_x}{b'_z} + \frac{b'_x\frac{\beta}{(1-t)}\hat{d}'_z}{(b'_z)^2} & e_{v,2B} &\approx \frac{b'_y}{b'_z} - \frac{\frac{\beta}{(1-t)}\hat{d}'_y}{b'_z} + \frac{b'_y\frac{\beta}{(1-t)}\hat{d}'_z}{(b'_z)^2}.
\end{aligned} \tag{B.4}$$

These equations can be rearranged to yield:

$$\begin{aligned}
e_{u,2A} &\approx \frac{b'_x}{b'_z} + \frac{\beta}{t}\left(\frac{1}{b'_z}\right)\left(\hat{d}'_x - \frac{b'_x\hat{d}'_z}{b'_z}\right) & e_{v,2A} &\approx \frac{b'_y}{b'_z} + \frac{\beta}{t}\left(\frac{1}{b'_z}\right)\left(\hat{d}'_y - \frac{b'_y\hat{d}'_z}{b'_z}\right) \\
e_{u,2B} &\approx \frac{b'_x}{b'_z} - \frac{\beta}{1-t}\left(\frac{1}{b'_z}\right)\left(\hat{d}'_x - \frac{b'_x\hat{d}'_z}{b'_z}\right) & e_{v,2B} &\approx \frac{b'_y}{b'_z} - \frac{\beta}{1-t}\left(\frac{1}{b'_z}\right)\left(\hat{d}'_y - \frac{b'_y\hat{d}'_z}{b'_z}\right).
\end{aligned} \tag{B.5}$$

When $\beta = 0$ (that is, $\vec{d} = 0$), we have, as expected:

$$\bar{e}_{2A} = \bar{e}_{2B} = (e_{u,2}, e_{v,2}) = \left(\frac{b'_x}{b'_z}, \frac{b'_y}{b'_z}\right), \quad \text{if } \vec{d} = 0. \tag{B.6}$$

We can express the image-space perturbation of the epipole away from the location given by equation B.6, due to a non-zero $\vec{d}$, as:

$$\Delta e = (\Delta e_{u,2}, \Delta e_{v,2}) \tag{B.7}$$

So,

$$\begin{aligned}
\Delta e_{2A} &= \left[\frac{\beta}{t} \cdot \frac{1}{b'_z}\right] \left(\hat{d}'_x - \frac{b'_x \hat{d}'_z}{b'_z} \quad , \quad \hat{d}'_y - \frac{b'_y \hat{d}'_z}{b'_z}\right) \\
\Delta e_{2B} &= -\left[\frac{\beta}{1-t} \cdot \frac{1}{b'_z}\right] \left(\hat{d}'_x - \frac{b'_x \hat{d}'_z}{b'_z} \quad , \quad \hat{d}'_y - \frac{b'_y \hat{d}'_z}{b'_z}\right).
\end{aligned} \tag{B.8}$$

Using the substitution $\vec{d'} = \beta \hat{\vec{d'}}$ and rearranging, we can express these last equations a little more cleanly as:

$$\begin{aligned}
\Delta e_{2A} &= \left[\frac{1}{t} \cdot \frac{d'_z}{b'_z}\right] \left(\frac{d'_x}{d'_z} - \frac{b'_x}{b'_z} \quad , \quad \frac{d'_y}{d'_z} - \frac{b'_y}{b'_z}\right) \\
\Delta e_{2B} &= -\left[\frac{1}{1-t} \cdot \frac{d'_z}{b'_z}\right] \left(\frac{d'_x}{d'_z} - \frac{b'_x}{b'_z} \quad , \quad \frac{d'_y}{d'_z} - \frac{b'_y}{b'_z}\right).
\end{aligned} \tag{B.9}$$

B.2 Linearization of 3D warp translation

In this section, I derive a linearization of the expression describing the translation component of the 3D warp. The translation component is described by Equation 3.34. This equation is repeated here:

$$\bar{u}_2 - \vec{u}'_2 = \frac{S_2}{z_2} \mathbf{P}_2^{-1} (\dot{C}_1 - \dot{C}_2) \tag{3.34}$$

Making the appropriate substitutions for each reference image (A and B) into equation 3.34 and rearranging slightly, we get:

$$\begin{aligned}
\bar{u}'_{2A} &= \bar{u}_2 - \frac{S_2}{z_2} \mathbf{P}_2^{-1} (\dot{C}_A - \dot{C}_2) \\
\bar{u}'_{2B} &= \bar{u}_2 - \frac{S_2}{z_2} \mathbf{P}_2^{-1} (\dot{C}_B - \dot{C}_2)
\end{aligned} \tag{B.10}$$

We can use the definition of $\vec{e}_{2A}$ and $\vec{e}_{2B}$ from Equation 3.13 to make additional substitutions:

$$\begin{aligned}
\bar{u}'_{2A} &= \bar{u}_2 - \frac{S_2}{z_2} \vec{e}_{2A} \\
\bar{u}'_{2B} &= \bar{u}_2 - \frac{S_2}{z_2} \vec{e}_{2B}
\end{aligned} \tag{B.11}$$

Then we can re-express the first of these two equations (and similarly for the second) as:

$$\begin{bmatrix} u'_{2A,x} \\ u'_{2A,y} \\ u'_{2A,z} \end{bmatrix} = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} - \frac{S_2}{z_2} \begin{bmatrix} e_{2A,x} \\ e_{2A,y} \\ e_{2A,z} \end{bmatrix} \tag{B.12}$$

Then, in image coordinates, for reference image A (and similarly for B):

$$\begin{aligned} u'_{2A} &= \frac{u'_{2A,x}}{u'_{2A,z}} = \frac{u_2 - \frac{S_2}{z_2} e_{2A,x}}{1 - \frac{S_2}{z_2} e_{2A,z}} = \frac{z_2\, u_2 - S_2 e_{2A,x}}{z_2 - S_2 e_{2A,z}} \\ v'_{2A} &= \frac{u'_{2A,y}}{u'_{2A,z}} = \frac{v_2 - \frac{S_2}{z_2} e_{2A,y}}{1 - \frac{S_2}{z_2} e_{2A,z}} = \frac{z_2\, v_2 - S_2 e_{2A,y}}{z_2 - S_2 e_{2A,z}} \end{aligned} \tag{B.13}$$

We can approximate this expression (using Approximation B.2) by taking advantage of the fact that $|S_2 e_{2A,z}| \ll |z_2|$. As before, the assumption of a large $|z_2|$ is valid, because a small value would indicate either an object that is extremely close, or an object that is off to the side and thus outside of any field-of-view substantially less than 180 degrees. The approximate expression for u'_{2A} is:

$$u'_{2A} \approx u_2 - \frac{S_2 e_{2A,x}}{z_2} + \frac{u_2 S_2 e_{2A,z}}{z_2} - \frac{(S_2)^2\, e_{2A,x} e_{2A,z}}{(z_2)^2} \tag{B.14}$$

But, as with similar earlier approximations, it turns out that we can drop the fourth term. The reason is that we expect that $|S_2 e_{2A,z}| \ll |z_2|$. So, we have:

$$u'_{2A} = u_2 + \Delta u_{2A}, \quad \text{with} \tag{B.15}$$

$$\Delta u_{2A} \approx \frac{S_2}{z_2} \left(-e_{2A,x} + u_2 e_{2A,z} \right) \tag{B.16}$$

The complete set of the delta equations for the epipoles resulting from reference images A and B is:

$$\begin{aligned} \Delta u_{2A} &\approx \frac{S_2}{z_2} \left(-e_{2A,x} + u_2 e_{2A,z} \right) & \Delta v_{2A} &\approx \frac{S_2}{z_2} \left(-e_{2A,y} + v_2 e_{2A,z} \right) \\ \Delta u_{2B} &\approx \frac{S_2}{z_2} \left(-e_{2B,x} + u_2 e_{2B,z} \right) & \Delta v_{2B} &\approx \frac{S_2}{z_2} \left(-e_{2B,y} + v_2 e_{2B,z} \right) \end{aligned} \tag{B.17}$$

We can rewrite these equations in $(\Delta u, \Delta v)$ form to get:

$$\begin{aligned} \Delta \bar{u}_{2A} &\approx \tfrac{S_2}{z_2} \left(-e_{2A,x} + u_2 e_{2A,z} \quad , \quad -e_{2A,y} + v_2 e_{2A,z} \right) \\ \Delta \bar{u}_{2B} &\approx \tfrac{S_2}{z_2} \left(-e_{2B,x} + u_2 e_{2B,z} \quad , \quad -e_{2B,y} + v_2 e_{2B,z} \right) \end{aligned} \tag{B.18}$$

A further reorganization (and substitution from Equation 1.18) clearly shows that the direction of these delta vectors is exactly towards or away from the appropriate epipole:

$$\begin{aligned}\Delta\bar{u}_{2A} &\approx \tfrac{S_2 e_{2A,z}}{z_2}\left(u_2 - e_{u,2A} \quad , \quad v_2 - e_{v,2A}\right) \\ \Delta\bar{u}_{2B} &\approx \tfrac{S_2 e_{2B,z}}{z_2}\left(u_2 - e_{u,2B} \quad , \quad v_2 - e_{v,2B}\right)\end{aligned} \tag{B.19}$$

This result can be written more concisely as:

$$\begin{aligned}\Delta\bar{u}_{2A} &\approx \tfrac{S_2 e_{2A,z}}{z_2}\left(\bar{u}_2 - \bar{e}_{2A}\right) \\ \Delta\bar{u}_{2B} &\approx \tfrac{S_2 e_{2B,z}}{z_2}\left(\bar{u}_2 - \bar{e}_{2B}\right)\end{aligned} \tag{B.20}$$

APPENDIX C

PER-PIXEL DATA

This appendix describes the per-pixel contents of the reference frames and displayed frame. These contents support the hole-filling algorithm described in Chapter 3 and the hybrid mesh/splat reconstruction algorithm described in Chapter 4.

For each variable stored at a pixel, I indicate the number of bits used to store it in my software test-bed (labeled “Bits (cur)”), and the minimum number of bits that I believe would be required to store it in an optimized implementation (labeled “Bits (min)”). The test-bed does not always use the minimum number of bits because it is designed to provide maximum flexibility to experiment with different algorithms.

C.1 Reference-frame per-pixel contents

Table C.1 lists the reference-frame per-pixel variables. RGB holds the pixel’s color, and $Z1$ holds the pixel’s depth as $\frac{1}{Z}$. The pixel’s surface orientation is stored as the partial derivatives of $\frac{1}{Z}$ with respect to u and v. These variables are listed as $dz1du$ and $dz1dv$ in the table. I believe that all very large values of $dz1du$ and $dz1dv$ (indicating extremely oblique surfaces) could be represented by a single large value, thus allowing a small number of bits to be used for these variables.

C.2 Displayed-frame per-pixel contents

Table C.2 lists the displayed-frame per-pixel variables. The top six entries in the table are required because of the hole-filling algorithm. Hole-filling must be performed on each warped source image, not on the composited image. RGB_{cursrc} and $Z1_{cursrc}$ implement this requirement by holding the results of the warp of the current source image. In contrast, $RGB_{composited}$ and $Z1_{composited}$ hold the results

	Variable	Bits (min)	Bits (cur)
1.	RGB	24	24
2.	$Z1$	16	32
3.	$dz1du$	4	32
4.	$dz1dv$	4	32

Table C.1: *Reference-frame per-pixel variables.*

of the incremental compositing (of multiple warped source images) described in Chapter 4. Both $Z1$ variables hold depth as $\frac{1}{Z}$.

	Variable	Bits (min)	Bits (cur)
1.	RGB_{cursrc}	24	24
2.	$Z1_{cursrc}$	16	32
3.	$cursrc$	2	2
4.	$HoleFillDist$	6	8
5.	$TruePixelFlag$	1	1
6.	$HoleFillWeight$	4	8
7.	$RGB_{composited}$	24	24
8.	$Z1_{composited}$	16	32
9.	$SplatFlag$	1	1
9.	$BetterMask$	2	2

Table C.2: *Displayed-frame per-pixel variables.*

$Cursrc$ indicates which source image RGB_{cursrc} and $Z1_{cursrc}$ come from, or zero if they are invalid. In effect, this functions as a flag to indicate whether or not RGB_{cursrc} has been written for the current source image. But, by storing an index value rather than a flag, the value does not need to be cleared before warping each source image—it only needs to be cleared before warping the first source image.

$HoleFillDist$ is only used during a hole-filling pass. If the hole-filling algorithm can be completed in a cache, then this variable does not need to be stored in main memory. For a hole pixel, $HoleFillDist$ indicates the distance to a background surface along the pixel's epipolar line. For non-hole pixels, the value is zero, indicating that they are a true surface.

$HoleFillWeight$ is used for blending the hole-fill contributions from multiple source images. If all source images warped so far have a hole at the current pixel ($TruePixelFlag$ is false), then $RGB_{composited}$ holds the consensus hole-fill color. $HoleFillWeight$ indicates the weight corresponding to the consensus color. This weight is used to weight the hole-fill-color contributions from additional warped source images.

$TruePixelFlag$ is set if $RGB_{composited}$ holds a warped candidate pixel. If the flag is not set, then this pixel is still a hole pixel, and $RGB_{composited}$ holds the consensus (weighted) hole-fill color computed so far.

$SplatFlag$ is set if $RGB_{composited}$ originated from a edge splat rather than a mesh element. Its use allows mesh elements to always win over edge splats of similar depth in the compositing process.

$BetterMask$ is used for combining the warps of different reference images. It is a bit-mask that specifies which reference images sample the surface held in $RGB_{composited}$ better than the reference image that produced the current contents of $RGB_{composited}$.

APPENDIX D

DERIVATIONS FOR CHAPTER 5

This appendix contains a derivation used in Chapter 5. I answer the question: “What is the maximum value of the expression $f = Ax + By + C$, where $A^2 + B^2 + C^2 = 1$, $|x| \leq x_{max}$, and $|y| \leq y_{max}$?”

Because the constraints on the expression are independent of sign, we know that:

$$\max\left(|Ax + By + C|\right) = \max((|Ax| + |By| + |C|)\,. \tag{D.1}$$

We compute the first of these two maxima in this section, but actually use the second in Chapter 5. We can incorporate the constraint $A^2+B^2+C^2 = 1$ into the expression $f = Ax+By+C$ as follows:

$$f = Ax + By + \left(1 - A^2 - B^2\right) \tag{D.2}$$

At $\max(f)$ with respect to A and B, we know that $\frac{\partial f}{\partial A} = 0$ and $\frac{\partial v}{\partial B} = 0$. Expanding the derivatives and simplifying the resulting equations, we get:

$$\begin{aligned} \left(1 + x_{max}^2\right) A^2 + x_{max}^2 \left(B^2 - 1\right) &= 0 \\ \left(1 + y_{max}^2\right) B^2 + y_{max}^2 \left(A^2 - 1\right) &= 0 \end{aligned} \tag{D.3}$$

Solving the second equation for B^2, substituting into the first, and simplifying, we get:

$$A = \sqrt{\frac{x_{max}^2}{x_{max}^2 + y_{max}^2 + 1}} \tag{D.4}$$

After more substitutions and simplifications,

$$\begin{aligned} B &= \sqrt{\tfrac{y_{max}^2}{x_{max}^2+y_{max}^2+1}} \\ C &= \sqrt{\tfrac{1}{x_{max}^2+y_{max}^2+1}} \end{aligned} \tag{D.5}$$

So,

$$\max\left(|Ax + By + C|\right) = \max\left(|Ax| + |By| + |C|\right) = \frac{x_{max}^2 + y_{max}^2 + 1}{\sqrt{x_{max}^2 + y_{max}^2 + 1}} \tag{D.6}$$

BIBLIOGRAPHY

[Adelson91] E. H. Adelson and J. R. Bergen. *The Plenoptic Function and the Elements of Early Vision*, chapter 1. MIT Press, 1991.

[Adelson93] Stephen J. Adelson and Larry F. Hodges. Stereoscopic ray-tracing. *The Visual Computer*, 10(3):127–144, 1993.

[Adelson95] Stephen J. Adelson and Larry F. Hodges. Generating exact ray-traced animation frames by reprojection. *IEEE Computer Graphics and Applications*, 15(3):43–52, 1995.

[Aliaga96] Daniel G. Aliaga. Visualization of complex models using dynamic texture-based simplification. In *Proceedings of IEEE Visualization 96*, pages 101–106, October 1996.

[Aliaga97] Daniel G. Aliaga. Architectural walkthroughs using portal textures. In *Proceedings of IEEE Visualization 97*, pages 355–362, October 1997.

[Aliaga98] D. Aliaga, J. Cohen, A. Wilson, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, E. Baker, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. A framework for the real-time walkthrough of massive models. Technical Report UNC-CH TR98-013, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, March 1998. Available at http://www.cs.unc.edu/Research/tech-reports.html.

[Amenta98] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 98)*, pages 415–421, Orlando, Florida, July 1998.

[Azuma94] Ronald Azuma and Gary Bishop. Improving static and dynamic registration in an optical see-through hmd. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 94)*, pages 197–204, Orlando, Florida, July 1994.

[Azuma95a] Ronald Azuma. *Predictive Tracking for Augmented Reality*. PhD thesis, University of North Carolina at Chapel Hill, 1995. Available as UNC-CH Computer Science TR95-007, at http://www.cs.unc.edu/Research/tech-reports.html.

[Azuma95b] Ronald Azuma and Gary Bishop. A frequency-domain analysis of head-motion prediction. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 95)*, pages 401–408, Los Angeles, CA, August 1995.

[Badt88] Sig Badt, Jr. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4(3):123–131, 1988.

[Blinn76] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, October 1976.

[Breglia81] Denis R. Breglia, A. Michael Spooner, and Dan Lobb. Helmet mounted laser projector. In *The 1981 Image Generation/Display Conference II*, pages 241–258, Scottsdale, Arizona, Jun 1981.

[Burbidge89] Dick Burbidge and Paul M. Murray. Hardware improvements to the helmet mounted projector on the visual display research tool (VDRT) at the naval training systems center. In *Proceedings SPIE*, volume 1116, pages 52–60, Orlando, Florida, Mar 1989.

[CAE84] CAE Electronics Ltd. Wide-field-of-view, helmet-mounted infinity display system development. interim report AFHRL-TR-84-27, US Air Force Human Resoures Laboratory, Operations Training Division, Dec 1984.

[Carpenter84] Loren Carpenter. The A-buffer, an antialiased hidden surface method. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):103–108, July 1984.

[Chen93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 93)*, pages 279–288, Anaheim, California, August 1993.

[Chen95] Shenchang Eric Chen. QuickTime VR — an image-based approach to virtual environment navigation. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 95)*, pages 29–38, Los Angeles, California, August 1995.

[Cook87] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering architecture. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):95–102, July 1987.

[Costella93] John P. Costella. Motion extrapolation at the pixel level. Unpublished paper available at http://www.ph. unimelb.edu.au/˜jpc, January 1993.

[Dally96] William J. Dally, Leonard McMillan, Gary Bishop, and Henry Fuchs. The delta tree: An object-centered approach to image-based rendering. MIT AI Lab Technical Memo 1604, MIT, May 1996. Available at http://www.ai.mit.edu/pubs.html.

[Darsa97] Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney. Navigating static environments using image-space simplification and morphing. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 25–34, Providence, RI, April 1997.

[Deering88] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: A VLSI system for high performance graphics. *Computer Graphics (Proceedings of SIGGRAPH 88)*, pages 21–30, 1988.

[Faugeras93] Olivier Faugeras. *Three-dimensional computer vision: a geometric viewpoint.* MIT Press, 1993.

[Foxlin93] Eric Foxlin. Inertial head-tracking. Master's thesis, Massachusetts Institute of Technology (EECS dept.), 1993.

[Foxlin98] Eric Foxlin, Michael Harrington, and George Pfeifer. Constellation: A wide-range wireless motion-tracking system for augmented reality. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 98)*, pages 371–378, Orlando, Florida, July 1998.

[Fuchs85] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, John G. Eyles, and John Poulton. Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes. *Computer Graphics (Proceedings of SIGGRAPH 85)*, pages 111–120, 1985.

[Gortler96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 96)*, pages 43–54, New Orleans, Louisiana, August 1996.

[Gradshteyn80] I. S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, corrected and enlarged fourth edition, 1980.

[Greene86] Ned Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[Greene94] Ned Greene and Michael Kass. Approximating visibility with environment maps. Technical Report #41, Apple Computer, November 1994.

[Grossman98] J. P. Grossman and William J. Dally. Point sample rendering. In G. Drettakis and N. Max, editors, *Rendering Techniques '98: Proceedings of the Eurographics Rendering Workshop 1998*, pages 181–192, Vienna, Austria, June 1998.

[Hill94] F. S. Hill, Jr. The pleasures of “perp dot” products. In *Graphics Gems IV*, pages 138–148. Academic Press, 1994.

[Hofmann88] Georg Rainer Hofmann. The calculus of the non-exact perspective projection. In *Proceedings of the European Computer Graphics Conference and Exhibition (Eurographics '88)*, pages 429–442, Nice, France, Sep 1988.

[Holloway95] Richard Lee Holloway. *Registration Errors in Augmented Reality Systems*. PhD thesis, University of North Carolina at Chapel Hill, 1995. Available as UNC-CH Computer Science TR95-016, at http://www.cs.unc.edu/Research/tech-reports.html.

[Hudson98] Thomas C. Hudson and William R. Mark. Image warping for remote display of rendered images. Unpublished paper, October 1998.

[Jerri77] Abdul J. Jerri. The Shannon sampling theorm—its various extensions and applications: A tutorial review. *Proceedings of the IEEE*, 65(11):1565–1596, November 1977.

[Laveau94] S. Laveau and O. D. Faugeras. 3-D scene representation as a collection of images. In *Proc. of 12th IAPR Intl. Conf. on Pattern Recognition*, volume 1, pages 689–691, Jerusalem, Israel, October 1994.

[Levoy85] Marc Levoy and Turner Whitted. The use of points as a display primitive. Technical Report UNC-CH TR85-022, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, 1985.

[Levoy96] Marc Levoy and Pat Hanrahan. Light field rendering. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 96)*, pages 31–42, New Orleans, Louisiana, August 1996.

[Lippman80] Andrew Lippman. Movie-maps: An application of the optical videodisc to computer graphics. *Computer Graphics (Proceedings of SIGGRAPH 80)*, 14(3):32–42, July 1980.

[Luebke97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 97)*, pages 199–206, Los Angeles, California, August 1997.

[Maciel95] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 95–102, Monterey, CA, April 1995.

[Malzbender93] Tom Malzbender and Susan Spach. A context sensitive texture nib. In N. M. Thalmann and D. Thalmann, editors, *Communicating with Virtual Worlds (Proceedings of Computer Graphics International '93)*, pages 151–163. Springer-Verlag, 1993.

[Mark96] William R. Mark, Gary Bishop, and Leonard McMillan. Post-rendering image warping for latency compensation. Technical Report UNC-CH TR96-020, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, January 1996. Available at http://www.cs.unc.edu/Research/tech-reports.html.

[Mark97a] William R. Mark and Gary Bishop. Memory access patterns of occlusion-compatible 3D image warping. In *Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 35–44, Los Angeles, CA, August 1997.

[Mark97b] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3D warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 7–16, Providence, RI, April 1997.

[Max95] Nelson Max and Keiichi Ohsaki. Rendering trees from precomputed Z-buffer views. In Patrick M. Hanrahan and Werner Purgathofer, editors, *Rendering Techniques '95: Proceedings of the Eurographics Rendering Workshop 1995*, pages 45–54, Dublin, Ireland, June 1995.

[Max96] Nelson Max. Hierarchical rendering of trees from precomputed multi-layer z-buffers. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96: Proceedings of the Eurographics Rendering Workshop 1996*, pages 165–174, Porto, Portugal, June 1996.

[Mazuryk95] Tomasz Mazuryk and Michael Gervautz. Two-step prediction and image deflection for exact head tracking in virtual environments. *Computer Graphics Forum (Eurographics '95)*, 14(3):C29–C41, 1995.

[McAllister99] David K. McAllister, Lars Nyland, Voicu Popescu, Anselmo Lastra, and Chris McCue. Real time rendering of real world environments. Technical Report UNC-CH TR99-019, University of North Carolina at Chapel Hill, Dept. of Computer Science, 1999. Available at http://www.cs.unc.edu/Research/tech-reports.html.

[McMillan95a] Leonard McMillan and Gary Bishop. Head-tracked stereoscopic display using image warping. In S. Fisher, J. Merritt, and B. Bolas, editors, *Proceedings SPIE*, volume 2409, pages 21–30, San Jose, CA, Feb 1995.

[McMillan95b] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 95)*, pages 39–46, Los Angeles, CA, August 1995.

[McMillan97] Leonard McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997. Available as UNC-CH Computer Science TR97-013, at http://www.cs.unc.edu/Research/tech-reports.html.

[Molnar92] Steven Molnar, John Eyles, and John Poulton. PixelFlow: high-speed rendering using image composition. *Computer Graphics (Proceedings of SIGGRAPH '92)*, 26(2):231–240, July 1992.

[Molnar94] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[Pizer83] Stephen M. Pizer and Victor L. Wallace. *To Compute Numerically: Concepts and Strategies*. Little, Brown and Company, 1983.

[Popescu98] Voicu Popescu, Anselmo Lastra, Daniel G. Aliaga, and Manuel de Oliveira Neto. Efficient warping for architectural walkthroughs using layered depth images. In *Proceedings of IEEE Visualization 98*, pages 211–215, October 1998.

[Pulli97] Kari Pulli, Michael Cohen, Tom Duchamp, Hughes Hoppe, Linda Shapiro, and Werner Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In Julie Dorsey and Philipp Slusallek, editors, *Rendering Techniques '97: Proceedings of the Eurographics Rendering Workshop 1997*, pages 23–34, St. Etienne, France, June 1997.

[Rademacher98] Paul Rademacher and Gary Bishop. Multiple center-of-projection images. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 98)*, pages 199–206, Orlando, Florida, July 1998.

[Rafferty98] Matthew M. Rafferty, Daniel G. Aliaga, and Anselmo A. Lastra. 3D image warping in architectural walkthoughs. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 228–233, Atlanta, GA, March 1998.

[Ramachandran95] V. S. Ramachandran. *Perceptual Correlates of Neural Plasticity in the Adult Human Brain*, chapter 22. MIT Press, 1995.

[Rambus97] Rambus Inc. *Concurrent RDRAM User Guide*, 1997. available at http://www.rambus.com.

[Regan93] Matthew Regan and Ronald Pose. An interactive graphics display architecture. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 293–299, Seattle, Washington, September 1993.

[Regan94] Matthew Regan and Ronald Pose. Priority rendering with a virtual reality address recalculation pipeline. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 94)*, pages 155–162, Orlando, Florida, July 1994.

[Riner92] Bruce Riner and Blair Browder. Design guidelines for a carrier-based training system. In *Proceedings of IMAGE VI Conference*, pages 65–73, Scottsdale, Arizona, Jul 1992.

[Schaufler96a] Gernot Schaufler. Exploiting frame-to-frame coherence in a virtual reality system. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 95–102, Santa Clara, CA, Mar 1996.

[Schaufler96b] Gernot Schaufler and Wolfgang Stürzlinger. A three dimensional image cache for virtual reality. In *Proceedings of Eurographics '96*, pages C227–C235, Poitiers, France, Aug 1996.

[Schaufler97] Gernot Schaufler. Nailboards: A rendering primitive for image caching in dynamic scenes. In Julie Dorsey and Philipp Slusallek, editors, *Rendering Techniques '97: Proceedings of the Eurographics Rendering Workshop 1997*, pages 151–162, St. Etienne, France, June 1997.

[Shade96] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 96)*, pages 75–82, New Orleans, Louisiana, August 1996.

[Shade98] Jonathan W. Shade, Steven J. Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 98)*, pages 231–242, Orlando, Florida, July 1998.

[Sillion97] François Sillion, George Drettakis, and Benoit Bodelet. Efficient imposter manipulation for real-time visualization of urban scenery. In *Proceedings of Eurographics 97*, pages C207–C218, Budapest, Hungary, September 1997.

[Snyder98] John Snyder and Jed Lengyel. Visibility sorting and compositing for image-based rendering. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 98)*, pages 219–230, Orlando, Florida, July 1998.

[So92] Richard H. Y. So and Michael J. Griffin. Compensating lags in head-coupled displays using head position prediction and and image deflection. *Journal of Aircraft*, 29(6):1064–1068, Nov-Dec 1992.

[Szeliski96] Richard Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, March 1996.

[Torborg96] Jay Torborg and James T. Kajiya. Talisman: Commodity realtime 3D graphics for the PC. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 96)*, pages 353–364, New Orleans, Louisiana, August 1996.

[Ward90] Greg Ward. pinterp utility, part of RADIANCE v1.2 and above, January 1990. man page available from http://radsite.lbl.gov/radiance.

[Watt92] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice.* Addison-Wesley, 1992.

[Weber97] Hans Weber. Predictive head tracking using a body-centric coordinate system. Ph.D. Dissertation Proposal, available from http://www.cs.unc.edu/˜weberh, April 1997.

[Welch97] Greg Welch and Gary Bishop. SCAAT: Incremental tracking with incomplete information. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 97)*, pages 333–344, Los Angeles, California, August 1997.

[Wells84] Maxwell J. Wells and Michael J. Griffin. Benefits of helmet-mounted display image stabilization under whole-body vibration. *Aviation, Space, and Environmental Medicine*, 55(1):13–18, Jan 1984.

[Westover90] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):367–376, August 1990.

[Whitted82] Turner Whitted and David M. Weimer. A software testbed for the development of 3D raster graphics systems. *ACM Transactions on Graphics*, 1(1):43–58, January 1982.

[Wolberg92] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California, 1992.

[Zangemeister81] Wolfgang H. Zangemeister, Ashby Jones, and Lawrence Stark. Dynamics of head movement trajectories: Main sequence relationship. *Experimental Neurology*, 71(1):76–91, Jan 1981.

[Zyda93] Michael Zyda, David Pratt, John Falby, Paul Barham, and Kristen Kelleher. NPSNET and the naval postgraduate school graphics and video laboratory. *PRESENCE*, 2(3):244–258, Summer 1993.