Efficient Reconstruction Techniques for Post-Rendering 3D Image Warping

TR98-011
March 21, 1998

William R. Mark
Gary Bishop

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

UNC CS Technical Report #TR98-011, March 21, 1998

Efficient Reconstruction Techniques for Post-Rendering 3D Image Warping

William R. Mark   Gary Bishop

Department of Computer Science[*]
University of North Carolina at Chapel Hill

Abstract

Image-based rendering replaces the difficult physical simulation problem of conventional computer graphics with a difficult reconstruction problem. In this paper, we introduce efficient methods for eliminating common reconstruction artifacts in 3D image warping. These methods are suitable for hardware implementation. We study the reconstruction problem in the context of the 3D warp's application to post-rendering warping. This application provides us with perfectly controlled reference images and allows us to compare generated images against a gold standard provided by conventional rendering.

We characterize the reconstruction problem as one of reconstructing 2D manifolds (surfaces) in 3-space, followed by a projection, resampling, and compositing of these surfaces. We present two reconstruction and resampling algorithms. The first algorithm is a general one. The second algorithm is an efficient micro-polygon technique that uses flat-shaded, axis-aligned rectangles and super-sampled anti-aliasing to avoid explicit interpolation between re-projected samples. We also present an algorithm, based on epipolar geometry, to estimate the color of any unknown areas of the warped image. This algorithm's work is strictly bounded by the resolution of the warped output image. Finally, we discuss the effectiveness of our post-rendering warping system.

1 Introduction

Image-based scene representations provide an alternative to conventional geometrical representations for rendering. We are particularly interested in image-based representations consisting of planar images with per-pixel depth or disparity values. These depth images are compact and can be acquired using computer vision techniques or active depth measuring devices. Conventional rendering techniques that preserve the Z-buffer can also generate depth images.

One can generate new views of a scene by re-projecting ("3D warping") the pixels from one or more depth images to a new viewpoint and view plane [20]. We refer to the original image as a *reference* image, and the new image as a *destination* or *derived* image. This paper examines the reconstruction and resampling problem encountered in such 3D image warping, with a focus on developing algorithms suitable for hardware implementation. Conceptually, the reconstruction problem consists of projecting the depth samples into 3-space, reconstructing the 3D world, and then re-projecting and resampling this 3D world to create a new image.

Our work has been driven by the application of 3D warping to post-rendering warping. Several characteristics of this application make it ideal for studying the reconstruction problem. First, we have a gold standard by which to judge our results: a conventionally rendered image. Second, our reference samples are well-controlled point samples, with accurate depth information. We can also obtain additional accurate information, such as per-pixel surface orientation. Finally, the distance between the reference and destination image viewpoints is relatively small, limiting the number and severity of occlusion artifacts.

An earlier paper of ours [17] describes the basic concept of post-rendering 3D image warping. We will summarize the important ideas now.

1.1 Post-Rendering 3D Warping

The frames generated by interactive 3D graphics applications exhibit enormous frame-to-frame coherence. This coherence can be exploited to completely avoid conventional rendering of most frames. Every Nth frame is conventionally rendered, and the in-between frames are generated with an image warp that extrapolates from the nearby conventionally rendered frames. This image warp can also be used to compensate for latency in the conventional rendering stage. Because the cost of an image warp is largely independent of scene complexity, the technique provides a considerable speedup for complex scenes.

Post-rendering image warping was first used in the form of image shifting by Breglia et al. [3]., where it was used for latency reduction. Hofmann [12], Regan and Pose [25] and Torborg and Kajiya [27] used 2D perspective or affine image warps to increase a rendering system's frame rate. Because these 2D image warps can not by themselves produce motion parallax, these systems must segment the scene into different layers which are independently rendered and warped.

By using a 3D warp, we eliminate the scene segmentation problem imposed on software by the 2D post-rendering warping systems. For scenes without moving objects or partially transparent surfaces, our approach places almost no burden on the application programmer. Moving objects and transparent surfaces require some special treatment, but application-controlled scene segmentation based on depth is still unnecessary.

The 3D warp can introduce visibility artifacts, if a portion of the scene that should be visible in the derived frame is not visible in the reference frame. We address this problem by always warping two reference frames to produce each derived frame (Figure 1). If the reference frames are properly chosen, this technique eliminates

[*]CB #3175, Sitterson Hall; Chapel Hill, NC 27599-3175 USA.
email: {billmark | gb}@cs.unc.edu
www: http://www.cs.unc.edu/~{billmark | gb}
phone: +1.919.962.{1917 | 1886}

most visibility artifacts. Any regions of the derived frame which are not visible in either reference frame are filled using an approximation technique. Throughout this paper, we refer to these regions needing approximation as *holes* in the derived frame. Plate 1a shows a frame output from our system.

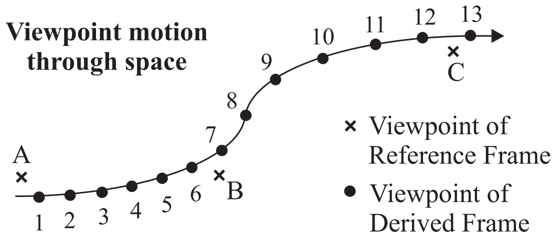
Viewpoint motion through space
1
2
3
4
5
6
7
8
9
10
11
12
13
A
B
C
× Viewpoint of Reference Frame
● Viewpoint of Derived Frame

Figure 1: *Derived 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. 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 (as shown here).*

Figure 2 is a conceptual diagram of a 3D post-rendering warping system. Although the figure shows two image warpers, an actual system would use one warper to warp both reference frames. 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.

2 Contribution

This paper concentrates on the problem of efficient and high-quality reconstruction for the 3D image warp.

Much of the previous work in image-based rendering has focused on obtaining real-time performance with existing hardware. Existing hardware places considerable constraints on the algorithms that can be considered. We also believe that existing hardware will not be able to simultaneously achieve the following three desirable goals for arbitrary scenes:

1. **High Quality**: 640x480 or higher resolution, anti-aliased, minimal artifacts.
2. **Fast**: 30–70 Hz frame rate.
3. **Cheap**: $1000 or less with today's technology.

With these goals in mind, we have taken a different path in our work. Rather than constraining ourselves to algorithms that can achieve real-time performance with existing hardware, we have explored algorithms that are suitable for implementation in new hardware that is specifically designed to support image-based rendering. We have designed our algorithms to use simple and efficient reconstruction, and to minimize memory-bandwidth requirements. We evaluate the image quality of these algorithms with a software test-bed that produces frames off-line for transfer to videotape.

Although we do not provide a detailed design for hardware that uses our algorithms, we do describe the type of hardware design that we envision. We believe that the strategy we present will be valuable to anyone considering a design of 3D image-based rendering hardware.

The 3D warping algorithm described in our earlier paper [17] has several important shortcomings that this paper addresses:

1. The technique for filling holes in the derived frame required the rasterization of potentially very large triangles during the reconstruction process. As a result, the work for the warp was not strictly bounded by image size. The potentially large triangles also precluded the use of a fixed-footprint rasterizer for reconstruction.

   This paper describes a new post-processing hole-filling technique that bounds the work for the warp and allows the use of a 4x4 footprint rasterizer for reconstruction.

2. The reconstruction technique required expensive color- and depth-interpolated triangles.

   One of our new methods uses flat-shaded, axis-aligned rectangles.

3. The reconstruction technique was prone to certain types of artifacts. In particular, one-pixel-wide features would disappear, and pinholes would appear under certain circumstances.

   The algorithms described here correctly reconstruct one-pixel-wide features, and eliminate most instances of pinholes.

4. The algorithm for segmenting the reference images into different surfaces ("discontinuity detection") was unreasonably complicated and expensive.

   The method described in this paper is extremely simple, yet more effective at avoiding artifacts.

5. The super-sampling technique required a memory bandwidth that scaled linearly with the number of super-samples per pixel.

   Here, we discuss an A-buffer algorithm to efficiently super-sample.

This paper describes our algorithms and provides a theoretical justification for them. In addition, this paper demonstrates post-rendering 3D warping using a more faithful simulation of real-world conditions than we have used previously. Our previous work relied on simulated user motions and simulated motion-prediction error. Our new system uses actual user motions acquired from a head position tracker, and an actual motion-prediction algorithm.

Although we focus primarily on the application of our 3D warping techniques to single-layer post-rendering 3D image warping, most of our techniques are applicable to other uses of 3D warping. Examples include imposter/portal-type 3D image warping [24], and warping of depth-images acquired from the real world using computer vision techniques [21]. A simple taxonomy (Table 3) can be used to organized image-based rendering applications. Applications which are closest to post-rendering warping in this taxonomy will benefit most from our techniques.

	Acquired Images (Real-World)	Rendered Images (Synthetic)
Changing Reference Images	3D Teleconference Systems	Post-Rendering Warping
Fixed set of Reference Images	Viewing real objects and scenes.	View high-quality rendered scenes

Figure 3: **[TABLE]** *Image-based rendering applications can be organized into a simple taxonomy based on the type of reference images and whether or not the reference images change in real time.*

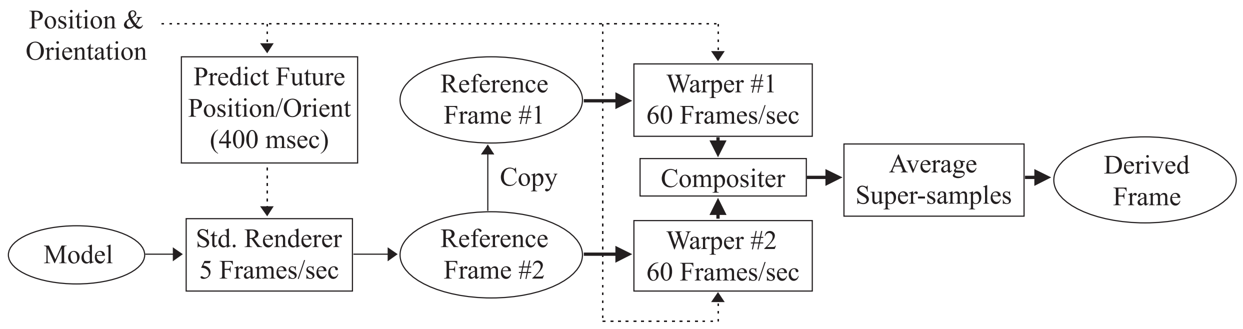

Figure 2: *Conceptual diagram of a post-rendering 3D warping system. The frame rates are in simulated time for our test-bed.*

3 Related Work

Blinn and Newell's environment mapping [2] is an early example of image-based rendering. Levoy and Whitted used points as an intermediate representation when rendering surfaces [15]. Chen and Williams [5] developed a hybrid 2D/3D image warp. McMillan and Bishop describe the 3D warp for planar reference images [20] and for cylindrical reference images [21]. McMillan's dissertation [19] introduces the reconstruction problem for 3D warping. Ward developed a 3D warp algorithm for generating animations from a few ray-traced images [28]. Max [18] and Gortler et al. [11] have used depth images with multiple layers at each pixel for 3D warping. Schaufler [26] describes an enhanced layered 2D perspective image warp that uses depth values to resolve inter-layer occlusion. Darsa et al. [8] render multiple adaptively-triangulated depth images to produce new images.

Gortler et al. [10], and Levoy and Hanrahan [14] have developed a different type of image-based rendering and addressed the reconstruction problem in that context. In the most general sense, this light-field approach represents all possible light rays, but their systems store only a discretized 4D subset of rays with depth information used to aid in the reconstruction. Pulli et al. [23] use what we would refer to as a type of 3D warp to combine and blend multiple depth images, but develop their technique from the light-field framework.

4 3D Warp Artifacts

Simple techniques for 3D warp resampling produce undesirable artifacts. McMillan's dissertation discusses some of these techniques and their shortcomings [19]. The simplest technique is to transform reference image samples to the destination image, and use a one-pixel reconstruction kernel. This strategy results in pinholes in any surfaces that are slightly under-sampled (Figure 4).

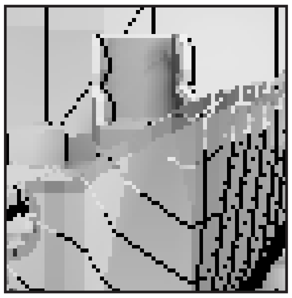

Figure 4: *The simplest form of resampling uses a one pixel reconstruction kernel for each reference pixel and causes holes to appear in surfaces that are slightly under-sampled.*

By using a larger (for example 3x3 pixel) fully-opaque reconstruction kernel, most of the pinholes can be eliminated. But, surface edges enlarge and become blocky, and some image detail is lost. Also, the interaction of the large kernel with the traversal order of the warp causes most image details to be incorrectly shifted by a pixel or two in the image plane.

If a per-pixel normal vector is available, then a variable-sized reconstruction kernel can be used. For image areas containing planar surfaces, this approach works well, but for curved surfaces the pinhole problem returns.

The same tradeoff between pinholes and loss of detail occurs when using partially transparent splat-type reconstruction kernels, as in [11]. However, in this case the pinholes are "fractional pinholes", in which the colors of background objects that should be fully occluded are incorrectly blended with foreground object colors. The fractional pinholes can be eliminated by enlarging the opaque portion of the reconstruction kernel, but blocky and/or blurry images result.

An alternative type of resampling assumes continuity between neighboring samples. Color is linearly interpolated between the transformed sample locations. In the images produced by this mesh-like approach, the internal regions of surfaces in a scene are reproduced well, without pinholes or blockiness. However, "rubber sheets" appear between foreground and background objects (Figure 5).

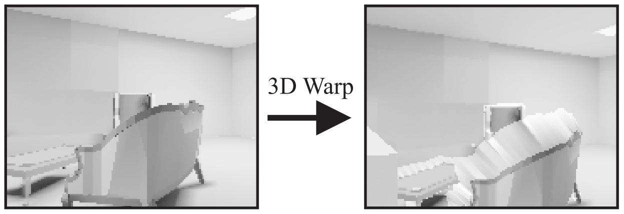

Figure 5: *Assuming continuity between all neighboring samples causes "rubber sheets" to stretch between foreground and background objects. The right-side image shows these sheets, in a lighter shade.*

5 Reconstruction and Resampling Strategy

By considering the 3D world that the reference and destination images represent, we can improve the quality of the reconstruction and resampling. The reference images depict the projection onto a plane of a set of 2D manifolds (surfaces) which form the boundaries of objects in the 3D world. We would like these same object boundaries to be faithfully represented in the destination image.

An ideal 3D warp would reconstruct these surfaces in 3-space *before* attempting to re-project and resample them. The most important characteristic of such a reconstruction is the segmentation of the reference image into sample sets representing *different* surfaces in 3-space. Once this segmentation is complete, each surface in 3-space is independently reconstructed. This reconstruction strategy for 3D warping shares much in common with systems for building models from laser range images [7].

Many reconstruction and resampling algorithms do not truly perform a full reconstruction before resampling, and ours follows this pattern. The reconstruction in 3-space is conceptual rather than actual. Our 3D warping 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.

Because our goal is to develop an algorithm for inexpensive real-time systems, we are restricted to working with one reference image at a time. Thus, the steps above are performed for each reference image in turn (and really for local neighborhoods of pixels within each image in turn). The compositing step (#4) is therefore incremental. 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.

By working with only one reference image at a time, we discard some useful global information. In particular, step #2a can profitably use this global information. In a 3D warping system which works with pre-calculated or pre-acquired reference images, step #2a should be at least partially performed in a preprocessing step which works simultaneously with all reference images.

Our reconstruction and resampling approach is distinguished from earlier algorithms primarily by step #2 above. Earlier approaches applied a uniform reconstruction strategy to all transformed samples. We treat a neighborhood of samples differently depending on whether the samples belong to a single surface in 3-space or to multiple surfaces in 3-space. Our earlier system's technique [17] can be cast into this framework, but it was not as well-justified theoretically, and produced unnecessary artifacts. Systems that adaptively triangulate depth images [8] can also be considered to be using this framework.

6 Segmentation into Surfaces

Our reconstruction and resampling strategy requires that we conceptually group the transformed reference-image samples into sets corresponding to different 3-space surfaces. From a practical point of view, we just need to answer a relatively simple question: For a pair of samples which are adjacent in the reference image, do the two samples represent the same surface? If the two samples do not represent the same surface, then we say that there is a *discontinuity* between the two samples.

Figure 6 shows two examples of sample pairs to be checked for a discontinuity. 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 surfaces as well as the samples. However, in Figure 7a, which does not show the true surfaces, it is more difficult to distinguish the discontinuities from the non-discontinuities. The samples in Figure 7a could belong to either the surface configuration shown in Figure 7b, or the configuration shown in Figure 7c. In one case there is a discontinuity between the middle pair of samples, and in the other case there is not.

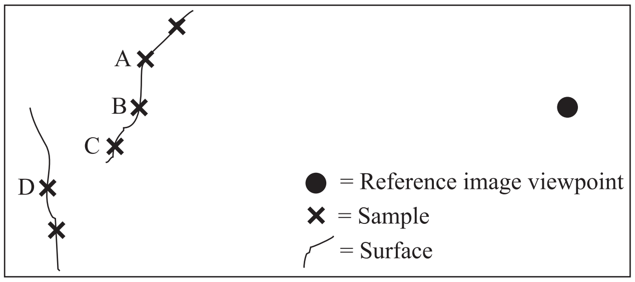

Figure 6: *Samples A and B clearly represent the same surface. Samples C and D clearly represent different surfaces.*

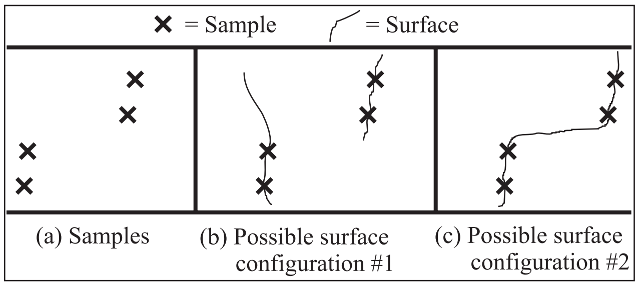
= Sample
= Surface
(a) Samples
(b) Possible surface configuration #1
(c) Possible surface configuration #2

Figure 7: *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.*

Unfortunately, there is no perfect solution to this problem. In general, the surface segmentation problem is unsolvable with the information we have available. We will briefly sketch out our argument for this assertion, using signal theory.

If a signal is locally band-limited in most regions, then we can consider a discontinuity to be a region of the signal with significant energy above this band-limit. Thus, if we sample substantially above the Nyquist rate for the 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 our case, the depth function that we are sampling can have energy content at or above the Nyquist frequency even in regions without discontinuities. Thus, we can not distinguish the discontinuities from the rest of the signal.

Our earlier system [17] used first derivative information (surface normals at sample points) as well as depth values to find discontinuities. But, such first derivative information is still not sufficient, since having this extra information is only equivalent to doubling the sampling rate [13]. Even additional information such as object-id numbers at each sample will not solve our problem, unless all objects are convex and non-intersecting.

Despite this theoretically impossible situation, we can do quite well in practice by picking an algorithm that satisfies two criteria. First, it should work well for the common cases. In particular, when there is an ambiguity between an extremely under-sampled surface and a discontinuity, the algorithm should categorize the configuration as a discontinuity. Second, in borderline cases the algorithm should make the decision that, if wrong, will produce the smallest error in the perceptual sense.

To re-cap, the segmentation algorithm must determine, for each pair of reference-image samples, whether or not the two samples are likely to represent the same surface. The algorithm we use is very simple. First, it projects the two reference-image samples into the destination-image plane, using the 3D warp transformation. 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.

This algorithm is dependent on the derived-frame viewpoint. This dependence may seem inappropriate, but it is crucial to satisfying the goal of minimizing perceived error. Plates 2a-c illustrate this point using magnified images. Our old approach was destination-view-independent, using reference image depths and surface orientations to find discontinuities. This old approach determined that a discontinuity existed between the red foreground object and the small, dark, middle-depth object in Plate 2a. As a result, the light-colored background shows through the discontinuity gap. Our new approach (Plate 2b) 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 (Plate 2c). 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 shows up as an extremely disturbing flickering. 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 threshold is exceeded). The error is usually completely imperceptible.

If a surface is sufficiently under-sampled, then our 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, our algorithm is getting the common case correct, since such samples usually belong to two distinct surfaces rather than a single under-sampled surface. Furthermore, if the samples *do* represent an under-sampled surface, the surface will typically be represented at a higher sampling density in another reference image, so that no artifact at all will be visible.

Our algorithm works well with a wide range of values for the destination-image plane distance threshold. This threshold value is expressed as a percentage of the “expected” distance (in pixels) between adjacent transformed samples. The expected distance is defined as the distance between transformed samples for a surface facing the reference image center of projection. We have successfully used thresholds between 140% and 230% of the expected distance. Higher values cause slight blurring at some foreground/background edges; lower values occasionally allow pinholes of the type shown in Plate 2a. Most of the videotape was produced using a threshold of 230% of the expected distance, but we came to the conclusion that this value was larger than necessary. Most of the plates in the paper (except Plate 1) were produced using a threshold of 140% of the expected distance.

Given the theoretical difficulty of segmenting a reference image into different surfaces using only the information available in that image, our simple algorithm works quite well. Systems which are willing to use inter-image information to aid in the segmentation could do considerably better. For example, the information from an appropriately placed second reference image can disambiguate the cases in Figure 7a and 7b. However, when the reference images are generated in real-time, preprocessing is not a possibility, and real-time inter-image analysis is expensive.

7 Two Algorithms for Reconstruction

We have developed two different approaches to surface reconstruction. The first approach is generally applicable, but somewhat expensive. It relies on explicit interpolation between most transformed samples. The second approach is designed for use 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.

7.1 Surface Reconstruction–General Algorithm

The first approach to reconstruction and resampling is our general technique, which we use when we are not performing super-sampled anti-aliasing. This technique is shown in Figure 8.

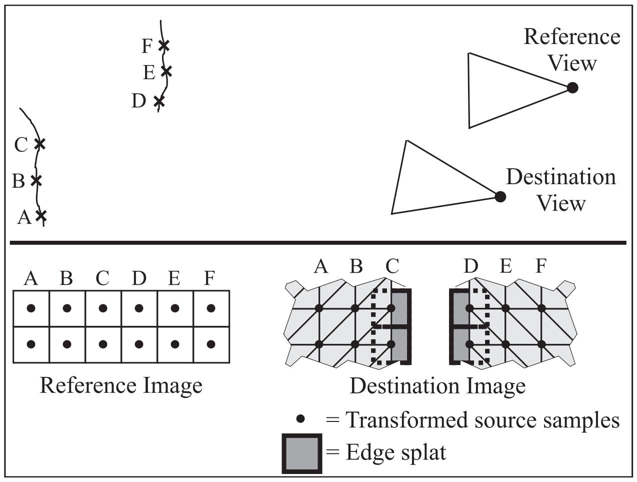

Figure 8: *Our general reconstruction and resampling technique. The top half of the figure is a top view of a scene. The bottom half shows the corresponding reference and destination images. The destination image depicts the transformed samples, the triangle mesh used for reconstructing the interiors of surfaces, and the edge splats used for reconstructing the edges of surfaces.*

For interior regions of surfaces, we reconstruct by treating the surface as a triangle mesh. The source-image samples form the vertices of the mesh. These vertices are warped to destination image coordinates, and the mesh is then rasterized and composited into the destination image. Colors and depths are linearly interpolated in the destination-image space.

Our earlier system reconstructed surfaces in essentially this same manner. Unfortunately, this technique shaves $1/2$ of a pixel off the edge of all surfaces. The most extreme example is a one-pixel-wide line, which disappears completely. Plate 3a illustrates this problem in a magnified image, and Plate 3b illustrates our solution, which performs extra reconstruction at surface edges.

This extra reconstruction treats pixels at the edge of a surface specially. We define an edge pixel as one which has a discontinuity between it and at least one of its neighbors. For edge pixels, we perform a splat-like quadrilateral reconstruction. We compute the corners of this *edge splat* using the sample’s location and surface orientation, as described in the Appendix. Because the splat color is not interpolated, we want to avoid overwriting the adjacent mesh-reconstruction with the splat color. A bit at each pixel in the destination image indicates whether the pixel originated from mesh or splat reconstruction. A splat pixel is never allowed to overwrite a mesh pixel at the same depth (within a tolerance). Thus, the edge splat only contributes to the portion of the surface outside the meshed interior. Figure 8 shows 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 best-sampled source image. Normally, our compositing algorithm (discussed later) takes care of this problem, by arbitrating between pixels from different images based on their sampling density. However, an edge splat from a poorly-sampled reference image can produce a rough edge that will stick out further in the destination image than the same edge from the better-sampled image. These samples will never have the opportunity to "lose" to better samples, and will thus remain in the final image.

We avoid this problem by suppressing an edge splat when both of the following two conditions are met:

1. The surface is poorly sampled. We designate the surface as poorly sampled if the density of transformed samples in the destination image (determined from surface orientation) is less than $1/2$ of the typical 1.0 density.

2. Another source image will sample the surface better (this assumes that the surface is visible in the other source image).

This algorithm always achieves the desired result, except when the "better" image does not in fact sample the surface at all, due to occlusion. In this instance, we effectively have fallen back to the previous technique, in which $1/2$ of a pixel is shaved off surface edges. We could detect this case by back-projecting our poor sample into the other source image to verify that the surface is actually sampled by the other image. However, this back-projection would be expensive in a hardware implementation, since it would require the hardware to have access to all source images simultaneously.

7.2 Surface Reconstruction–For Anti-Aliasing

When performing super-sampled anti-aliasing with a 3D warp, both the reference images and destination image should be at super-sampled resolution. The warp works with the images at the super-sampled resolution—final averaging of super-samples occurs after the warp. In this section, when we refer to "pixels", we are actually discussing the super-samples.

If our source images and our destination image are super-sampled, then we can greatly simplify our reconstruction algorithm. Our approach in this case was inspired by the REYES system's flat-shaded micro-polygons [6]. Because we can rely on the averaging of super-samples to implicitly perform our color interpolation, there is no longer any need to explicitly interpolate colors. We can also simplify the geometrical portion of the reconstruction—we use axis aligned rectangles rather than triangles. This approach is designed to be easily and cheaply implemented in hardware.

Although we no longer interpolate colors or depths between adjacent samples, we do not completely ignore the mesh relationship for connected surfaces. If we did ignore this relationship, by independently splatting each source pixel into the destination image, we would return to the tradeoff between loss of detail and creation of pinholes.

We begin the reconstruction process by transforming each source-image sample to determine its location in the destination image (Figures 9a and b). The extent of the reconstruction kernel for each sample is computed next. Initially, this reconstruction kernel is a general quadrilateral (Figure 9c). For 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 9c, corner #1's location is computed by averaging the coordinates of points A, B, C, and D. Figure 9c shows only the reconstruction kernel for sample D, but corner #1 is shared with the reconstruction kernels for samples A, B, and C, thus guaranteeing that there are no cracks in the mesh (Figure 9d). Finally, we convert the reconstruction kernel's extent from an arbitrary quadrilateral to an axis-aligned rectangle. We perform this conversion by taking the arbitrary quadrilateral's axis-aligned bounding box as the reconstruction kernel (Figure 9e). All pixel centers inside the axis-aligned kernel are filled with the sample's color and transformed 1/Z value.

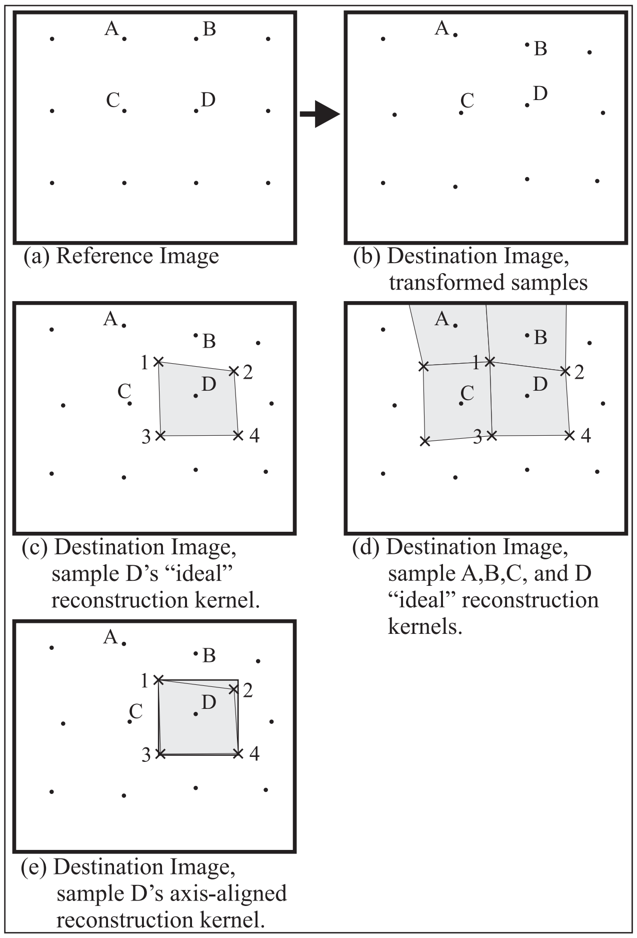
A
B
C
D
(a) Reference Image
(b) Destination Image, transformed samples
1
2
3
4
(c) Destination Image, sample D's "ideal" reconstruction kernel.
(d) Destination Image, sample A,B,C, and D "ideal" reconstruction kernels.
(e) Destination Image, sample D's axis-aligned reconstruction kernel.

Figure 9: *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 kernel is formed using the transformed reference-image samples.*

For a corner of a reconstruction kernel that lies on the edge of a surface, the above algorithm is modified. We say that a corner of a reconstruction kernel lies on an edge when the four reference-image samples surrounding the corner do not all belong to the same surface. We make this same-surface test using our discontinuity-detection algorithm. When a reconstruction-kernel corner is on an edge, the corner's location is computed using *only* information from the sample to which the kernel belongs. The computation is the same one used for computing a corner of an edge splat in our first reconstruction technique. The formulas, given in Appendix A, use the sample's position, depth, and surface orientation.

We compute a kernel corner in this same manner in one other instance. This other instance is when all four samples surrounding a corner have passed the discontinuity test, but fall in a *fold-over* configuration with respect to each other. Figure 10 illustrates this case, which usually results 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 kernel corner by averaging the four adjacent sample values would result in a misshapen kernel. Such a misshapen kernel might not even cover the transformed position of its own sample, clearly an undesirable result. We detect a fold-over configuration by examining the quadrilateral formed by the transformed positions of the four samples surrounding the corner. 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 we use is adapted from a Graphics Gem [9].

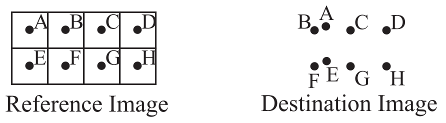

Figure 10: *A fold-over configuration. Samples A and E are on the "wrong" side of samples B and F.*

The conversion of the kernel from an arbitrary quadrilateral to an axis-aligned bounding box can result in slight overlap of adjacent reconstruction kernels. Because rotations about the view direction in a fraction of a second are generally very small, the overlap is minimal in practice—it is rare for the center of a destination pixel to fall in the overlapping region. We have not observed artifacts caused by this occasional overlap. For post-rendering warping, the tradeoff is worthwhile, since it is much cheaper to rasterize an axis-aligned rectangle than it is to rasterize an arbitrary quadrilateral. In some other 3D warping applications where more rotation occurs, it might be necessary to rasterize the quadrilaterals.

8 Compositing

As surfaces are reconstructed and then re-sampled at destination-image pixel centers, *candidate pixels* are produced. When warping two or more reference images, more than one candidate pixel may contend for the same pixel in the derived frame. Even when warping only a single reference image, multiple candidate pixels may contend for the same derived-frame pixel, if their original surfaces are at different depths and one is folded over the other by the 3D warp.

Our compositing algorithm must achieve two goals. First, it must arbitrate between candidate pixels that are at different depths. This goal is achieved by Z-buffering. Second, it 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.

Our compositing is performed incrementally, as part of the warping 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 each 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. 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 we do not do so. 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 compositer checks the bit in the new candidate pixel's bit-mask that corresponds to the already-stored pixel's source-image number. (Full disclosure: The code used for the videotape differs very slightly from this description, but we will be modifying the code to use the above algorithm, which is functionally equivalent to our current code but more memory efficient).

9 Hole Filling

When warping any fixed number of source images, some areas of the scene that should be visible in the destination image may not be visible in any source image. These *holes* in the destination image are usually quite small, but it is crucial that they be filled in a manner that minimizes their perceptual impact.

Typically the holes occur where a foreground object has moved with respect to a background surface (Plate 4a). The most likely "correct" way to fill the hole is thus to extend the background surface into the hole.

Our previous system approached this problem by treating the *entire* reference image as a triangle mesh. Elements of the mesh that bridged discontinuities between two true surfaces were treated specially. They were designated as "low-confidence" mesh elements, and would lose a composition test with any true surface. The color of these mesh elements was not interpolated; instead, the color was set to the color of the farthest-away vertex. Plate 4b shows the results of this approach.

There were two problems with our old approach. First, and most seriously, the low-confidence mesh elements could be quite large. These mesh elements were always rasterized, even when the hole between a foreground and background object was eventually filled by another source image. As a result, the computational work for the warp (measured in rasterized pixels) was no longer truly independent of image complexity. The arbitrarily-large low-confidence mesh elements also precluded the use of any type of inexpensive fixed-footprint hardware rasterizer. We viewed this restriction as a major obstacle to hardware implementation of a fast, low-cost warper.

The second problem with our old approach was that it introduced highly noticeable artifacts. The stretched-out low-confidence mesh elements form parallel stripes across the gap that they bridge. These stripes are clearly visible in Plate 4b. They are even more pronounced when super-sampled anti-aliasing is not being used.

Our new approach, illustrated in Plate 4c, solves both of these problems. Instead of filling holes during the warp itself using special mesh elements, we fill holes in a post-process step. The algorithm is based on the epipolar geometry of the 3D warp. Figure 11 illustrates this epipolar geometry for an example case.

From inside a hole region, we can find the background-object pixels adjoining the hole by searching backwards along the epipolar lines. But it is much more efficient to fill all holes in the destination image with a single pass over destination image, like that depicted in Figure 12. For each pixel of this traversal that is a hole pixel, the algorithm looks backwards by one pixel along the epipolar line. Because this backwards pixel has already been touched by the

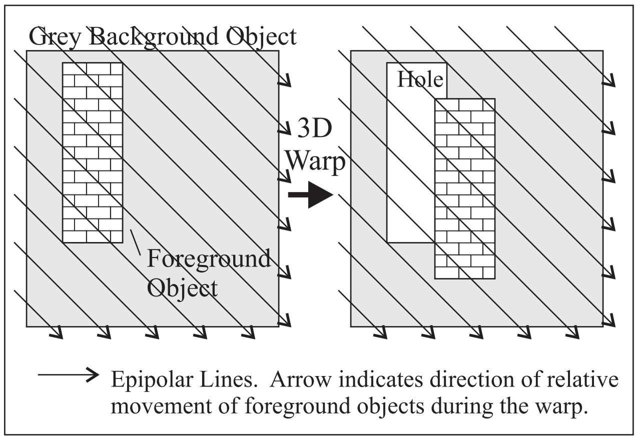
Grey Background Object
Hole
3D Warp
Foreground Object
Epipolar Lines. Arrow indicates direction of relative movement of foreground objects during the warp.

Figure 11: *Epipolar geometry for a warp of one source image to a destination image. In this particular case, the epipolar lines are nearly parallel, indicating that the epipole is far away. The 3D warp causes the foreground object to move relative to the background object, opening up a "hole" in the destination image.*

algorithm, it is either a true background pixel, or an already-filled hole pixel. In either case, this backwards pixel contains the background-object color that the algorithm needs to fill the current hole pixel.

The traversal order just described is actually an inverse occlusion-compatible traversal, in the *destination* image. In other words, if we were to consider warping the destination image to the source image, the inverse occlusion-compatible order for this warp would be the same order we use for hole filling. McMillan and Bishop introduced occlusion-compatible traversals for 3D warping of planar reference images in [20]. Our filling algorithm actually uses the eight-sheet variation their occlusion-compatible order that is described in [16].

The hole filling algorithm that we've just described is equivalent to filling every hole pixel with the nearest background object along the pixel's epipolar line. Unfortunately, this strategy produces the same stripe artifacts as our old algorithm. The colors along the edge of the background object propagate into the hole along epipolar lines. If the colors along this edge vary, then stripes in the direction of the epipolar lines will appear in the hole.

The solution to this problem is to blur the stripes together as they reach further into the hole. Our algorithm blurs by averaging several pixels to compute each new hole pixel, instead of copying a single pixel (Figure 13). If the hole pixel is close to a background surface, then three nearby pixels are averaged to compute the fill color (Figure 13b). The blurring caused by averaging three close pixels levels off as the distance from the true 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 13c). This blurring technique is not very sophisticated, but it is inexpensive to implement and succeeds in greatly reducing the stripe effect, as Plate 4c shows.

9.1 Hole filling for multiple source images

As described so far, the hole-filling algorithm is designed for warping a single source image to produce a destination image. When warping more than one source image, there is no single epipolar direction to use for hole filling. Conceptually, we solve this problem by warping each source image to its own destination image, performing hole filling in this destination image, and then compositing the individual destination images to form the final

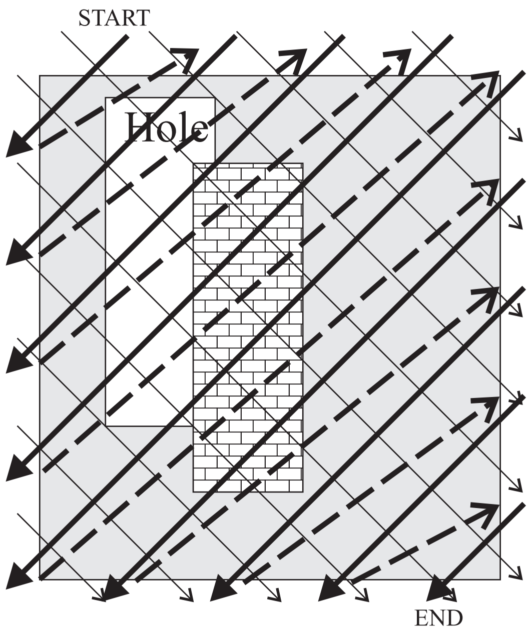

Figure 12: *To fill the holes, the destination image is traversed in the order indicated by the dark black lines. The background object color is thus propagated into the hole, one pixel at a time. The result is equivalent to that which would in theory obtained by traversing each epipolar line (thin lines) and copying the background color, but with fewer problems due to the discrete nature of the image.*

output image. The compositing process always discards hole-filled pixels in favor of non-hole-filled pixels. Thus, hole-filled pixels only appear in the final output image where hole-fill pixels appeared in *all* individual destination images. Ideally, all of these hole-fill pixels at the same location would all have the same color, but that is not always the case. So, the compositing averages the colors of the contributing hole-fill pixels to form the final hole-fill pixel. Each contribution is weighted by the reciprocal of that hole-pixel's distance (along its epipolar line) from a background pixel. This weighting 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.

In our system, the hole filling algorithm is not actually implemented by warping each source image to its own destination image. Instead, the entire process is performed incrementally in a single destination image that contains extra bit-planes for the hole-fill colors and flags. Appendix B briefly describes these extra bit-planes. This appendix also describes some additional details of the hole-filling algorithm.

10 Hardware

Our primary goal in this work has been to develop 3D warping algorithms that are suitable for implementation in inexpensive hardware. Figure 14 is a sketch of the type of system design we envision for anti-aliased post-rendering 3D warping. The 3D warper contains a flat-shaded rasterizer which can rasterize rectangles up to 4x4 pixels in size. Our rasterization algorithm guarantees that this size will not be exceeded for a $1 : 1$ ratio of source-to-destination angular pixel density. The destination-image cache in this design is large enough to hold the maximum-sized destination-image working set [16]. As a result, portions of the destination image only enter and exit the cache once during the warp of each source image. The hole-filling algorithm runs on portions of the image as they exit the cache.

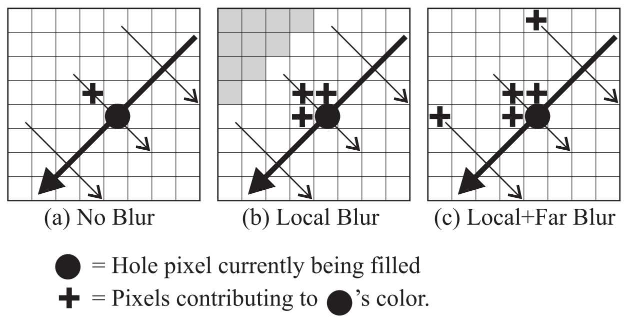
(a) No Blur
(b) Local Blur
(c) Local+Far Blur
● = Hole pixel currently being filled
✚ = Pixels contributing to ●'s color.

Figure 13: *Blurring techniques for hole filling. In (a), only a single already-filled pixel is examined to determine the hole-fill color of a new pixel. We do not use this approach, because it results in stripes. 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.*

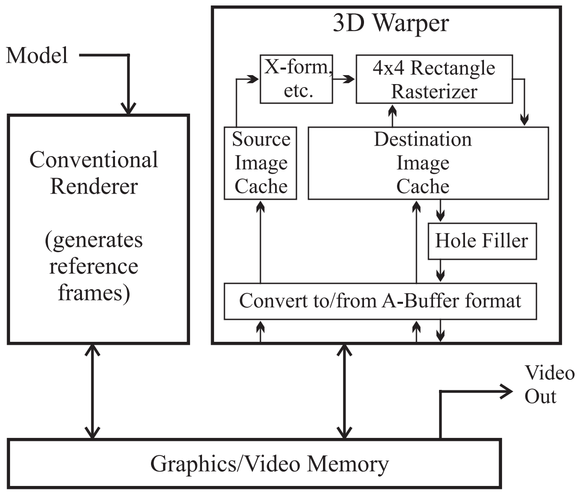

Figure 14: *A sketch of the 3D post-rendering warping system we envision. The 3D warper's source- and destination-image caches are entirely software managed. The destination-image cache is large enough to hold the maximum-sized working set.*

Traffic to and from the main graphics memory is in an efficient A-buffer format [4], greatly reducing the memory bandwidth required for the anti-aliased warping. We allow a maximum of three A-buffer fragments per pixel. If the renderer is a tiled renderer, then it can easily output images to memory in this A-buffer format. If not, then the 3D warper will have to convert each source image to this format the first time it is warped, and write it back to memory in the A-buffer format.

Our software test-bed includes code to demonstrate the visual effect of A-buffering. The test-bed represents the reference and destination frames in the A-buffer format in between the warp of each reference frame. The frames are expanded into super-sampled format for the actual warp. Our videotape and plates were produced with this A-buffering turned on, using a maximum of three fragments per pixel.

11 Post-Rendering Warp Testbed

We have evaluated our reconstruction, resampling, and hole-filling algorithms in a post-rendering 3D warping software test-bed. This test-bed conventionally renders reference frames at a (simulated) frame rate of five frames per second. It warps two reference frames to produce each derived frame. The derived-frame rate is 30 frames per simulated-second.

We used two models to demonstrate our system. The first model is a submarine's auxiliary machine room. For our display of this model we used a known spline-based path. Thus, reference image viewpoints could be chosen to lie exactly on the motion path. Even though this model is rendered using view-dependent per-pixel lighting, we do not observe any lighting artifacts introduced by the 3D warp. Although the 3D warp implicitly assumes diffuse surfaces, in post-rendering warping the reference-image viewpoint and derived-image viewpoint are generally close enough that artifacts do not appear unless highly specular lighting is used.

Our second model is the kitchen of a house. The path through this model was acquired by walking through a real-time display of the kitchen while wearing a motion-tracked head-mounted display. The motion tracker is an optical head tracker, with a Kalman filter that estimates velocity as well as position. We saved the position and velocity information in a file for our test-bed. Our test-bed used the velocity information to predict future reference-image viewpoints. The prediction interval is 400 msec [17]. The quality of the predicted positions is not very good—a system that used accelerometer data, such as [1], would be far superior. The poorly chosen reference images cause a significant number of holes to appear in the derived image, which our hole-filling algorithm must fill.

In a post-rendering warping system, the reference frames must be oversized to allow for change in the view direction. In the kitchen model, our display was 47 degrees by 60 degrees, and our reference frames were 68 degrees by 88 degrees, with a corresponding increase in the number of image pixels. These fields of view could be reduced somewhat with better-quality orientation prediction. The auxiliary machine room walk-through used a field of view for the reference frames of 52 degrees by 75 degrees, for the same display.

Our post-rendering warping test-bed does not currently support moving objects. The simplest approach to handling a small number of moving and partially transparent objects would be to conventionally render them into the derived frame after 3D warping has been completed.

12 Discussion

In our discussions about sampling density, we have been primarily concerned with the problem of under-sampling. It is also possible for oversampling 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 their reconstruction kernels map to the same destination-image sub-pixel. In most circumstances this is roughly equivalent to having sampled at a lower rate to begin with—it's what you'd get from conventional rendering from the new viewpoint. However, when the reference-image samples are pre-filtered (i.e. they come from rendering of MIP-mapped textures), discarding samples could introduce artifacts that would otherwise be preventable by the pre-filtering. The easiest solution to this problem, albeit an expensive one, is to increase the super-sampling of the destination image so that it is greater than that of the reference images. This problem could also be reduced by increasing the conventional renderer's texture filter kernel size slightly (changing the algorithm for choosing MIP-map levels). When super-sampling is already being used, a minor adjustment of this type would introduce very little blurring in the final image. Finally, the compositing algorithm's technique for arbitrating between competing samples of the same surface could be modified to discriminate against excessive oversampling as well as against under-sampling.

13 Conclusion

In this paper we have presented 3D image warping methods appropriate for hardware implementation. When working with super-sampled images, our algorithm performs reconstruction using flat-shaded, axis-aligned rectangles. The algorithm is able to enforce a maximum size on reconstruction kernel size, because holes in the destination image are filled using a new post-processing technique.

We have focused on the application of our techniques to post-rendering image warping of two or more single-layered reference images. In particular, we demonstrated for the first time the use of post-rendering 3D warping with actual user motions (and user motion predictions) collected from a head tracker. However, most of our 3D warping algorithms are applicable to other systems using 3D warping, including tele-presence systems and imposter or multi-layer post-rendering image warping systems.

14 Acknowledgements

We thank Leonard McMillan for many exciting and productive discussions, and for introducing us to the tool of epipolar geometry. We thank Steve Molnar for his comments on a draft of this report. In particular, his suggestions helped us improved the clarity of the "Reconstruction and and Resampling Strategy" section. We thank the members of the UNC image-based rendering research group for helpful discussions and comments.

The kitchen model was provided by the UNC Walkthrough group. The Auxiliary Machine Room model was provided to us by Greg Angelini, Jim Boudreaux and Ken Fast at Electric Boat, a subsidiary of General Dynamics. We thank Dan Aliaga for helping us with model conversion and management, and Todd Gaul for cheerful and efficient videotape editing.

Our overall image-based rendering research program has been sponsored primarily by DARPA and NSF, under DARPA contracts #63-93-C-0048 and #63-96-C-0041, and NSF grant #MIP-9612643. This particular work has been supported by a research grant from the Intel corportation, and by fellowships from the Microsoft Corporation and Link Foundation. We run our software test-bed on Pentium Pro workstations donated by Intel.

References

[1] 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.

[2] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, October 1976.

[3] 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.

[4] Loren Carpenter. The A-buffer, an antialiased hidden surface method. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):103–108, July 1984.

[5] 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.

[6] 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.

[7] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 96)*, pages 303–312, New Orleans, Louisiana, August 1996.

[8] 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.

[9] Jr. F. S. Hill. The pleasures of "perp dot" products. In *Graphics Gems IV*, pages 138–148. Academic Press, 1994.

[10] 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.

[11] Steven J. Gortler, Li wei He, and Michael F. Cohen. Rendering layered depth images. Technical Report #97-09, Microsoft Research, Mar 1997. Available at http://www.research.microsoft.com/pubs.

[12] 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.

[13] Abdul J. Jerri. The Shannon sampling theorm—its various extensions and applications: A tutorial review. *Proceedings of the IEEE*, 65(11):1565–1596, November 1997.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. InfiniteReality: A real-time graphics system. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 97)*, pages 293–301, Los Angeles, California, August 1997.

[23] 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.

[24] 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 ??–?? (To Appear), ??, ??, March 1998. (Note to reviewers: This paper uses 3D Warping for portals in architectural walkthroughs. Reconstruction issues are not addressed beyond the level in [17] and [11].).

[25] 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.

[26] 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.

[27] 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.

[28] Greg Ward. pinterp utility, part of RADIANCE v1.2 and above, January 1990. man page available from http://radsite.lbl.gov/radiance.

A Splat Formulas

The following formulas are useful when independently splatting a sample, as we do for our "edge splats". We adopt the notation and definitions used in McMillan and Bishop's planar-to-planar 3D warping paper [20]. You will need to refer to that paper to understand the following equations.

First, for any planar projection defined by $\vec{o}$, $\vec{u}$, and $\vec{v}$, we can calculate a scale factor, S, between disparity (δ) and $1/Z$. Thus, the two can be used almost interchangeably, as can their partial derivatives.

$$\delta \equiv \frac{S}{Z}, \qquad \text{where} \quad S \triangleq \frac{\vec{u} \times \vec{v}}{\|\vec{u} \times \vec{v}\|} \cdot \vec{o} \tag{1}$$

We need to be able to calculate the extents of splats which will exactly tile the destination image if the reference-image samples represent a planar surface. The sample point itself (center of splat) transforms to (x', y'). The four corners of such a splat are at:

$$\text{corners} = \left(x' \pm \frac{1}{2} \left(\frac{\partial x'}{\partial x} + \frac{\partial x'}{\partial y} \right) \quad , \quad y' \pm \frac{1}{2} \left(\frac{\partial y'}{\partial x} + \frac{\partial y'}{\partial y} \right) \right) \tag{2}$$

The partial derivatives are:

$$\frac{\partial x'}{\partial x} = \frac{(\partial\delta/\partial x)(c - ix') + a - gx'}{gx + hy + i\delta + m} \tag{3}$$

$$\frac{\partial x'}{\partial y} = \frac{(\partial\delta/\partial y)(c - ix') + b - hx'}{gx + hy + i\delta + m} \tag{4}$$

$$\frac{\partial y'}{\partial x} = \frac{(\partial\delta/\partial x)(f - iy') + d - gy'}{gx + hy + i\delta + m} \tag{5}$$

$$\frac{\partial y'}{\partial y} = \frac{(\partial\delta/\partial y)(f - iy') + e - hy'}{gx + hy + i\delta + m}. \tag{6}$$

As is usual for 3D warping, most of these computations are much less expensive than they look, since they can be performed incrementally.

In our system, the partial derivatives of δ with respect to source-image x and y are actually provided as partial derivatives of $1/Z$ with respect to source-image x and y, and then scaled. These partial derivatives represent surface orientation, in much the same way that a normal vector does. In any new hardware design, it is reasonable to expect that we can inexpensively obtain the partial derivatives of $1/Z$, since these values are already maintained internally in almost all conventional polygon rendering engines in order to interpolate $1/Z$.

B Additional Hole-Fill Algorithm Details

This appendix provides some additional (although terse) details about our hole-filling algorithm and its implementation.

B.1 Forward Edges

Our hole-fill algorithm as described earlier would not work well for the case depicted in Figure 15. When the filling algorithm looks in the inverse-occlusion compatible direction, it picks up the color of the non-convex foreground object rather than the color of the background object. But, a slight modification to the algorithm fixes this problem.

First, let us define a *forward edge* of an object. We make this definition in the source image, and with respect to the epipolar direction arrows (thin lines) in our diagrams. A source image pixel X is on a forward edge if the pixel, Y, that is adjacent to it in direction of our arrows has greater depth than X has. More precisely, the "direction of our arrows" is the inverse-occlusion compatible direction for the source-to-destination warp. Figure 15 shows the forward edges as thick lines.

During the warp, any pixel X on a forward edge is specially tagged, and the color of its adjacent pixel Y is stored with X in an auxiliary *hole-fill color buffer* associated with the destination image. During the hole-filling process, the color in this auxiliary buffer is used rather than the original color of the pixel. This algorithm eliminates the problem illustrated in Figure 15. For a forward-edge pixel, there is no need to *ever* use the original pixel color in the fill algorithm, since forward edges never cause gaps themselves. Instead, forward edges cause fold-over, which is correctly handled during compositing by the Z-buffering.

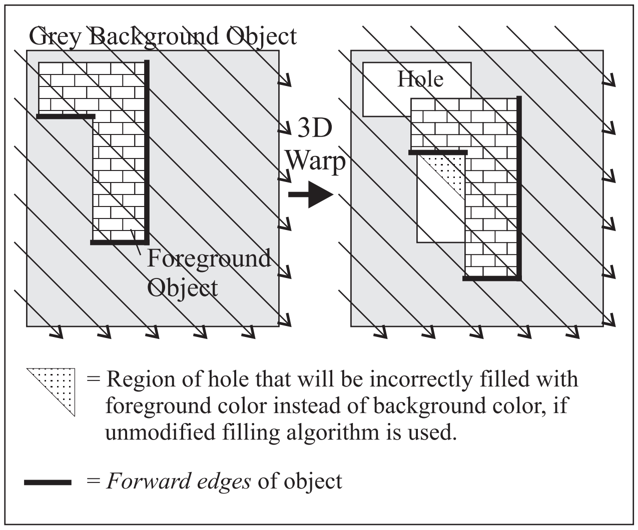
Grey Background Object
Hole
3D Warp
Foreground Object
= Region of hole that will be incorrectly filled with foreground color instead of background color, if unmodified filling algorithm is used.
= *Forward edges* of object

Figure 15: *The simple version of our hole filling algorithm will incorrectly fill some areas when the foreground object is non-convex. 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 overcome this problem.*

B.2 Hole-Fill Bit-Planes

In our current implementation, the destination image has the following values at each pixel while a warp is in progress:

1. Color (RGB)

2. Destination-image $1/Z$
3. $\frac{\partial(1/Z)}{\partial x}$ and $\frac{\partial(1/Z)}{\partial y}$
4. Source-image number that contributed color, $1/Z$ and derivatives.
5. Flag: Is Color (field #1) from a hole-fill?
6. Hole-fill Color (RGB)
7. Flag: Is Hole-fill Color valid?
8. Hole-fill-distance (0 indicates true surface)
9. Hole-fill-weight

As each source image is warped, colors are written to both the “Color” buffer, and the “Hole-fill Color” buffer. The special forward edge colors discussed in the previous section are written only to the “Hole-fill Color” buffer. After each source image warp is completed, the hole-filling algorithm is run using the “Hole-fill” buffers, with any filled values copied back into the “Color buffer”. Fields #6, #7, and #8 do not need to be preserved after each hole-fill pass is run (and thus need exist only in the cache). The “Hole-fill-weight” is used for the weighted averaging of the hole-fill contributions from different source images.