In Proceedings of 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware, Los Angeles, CA, August 3-4, 1997, pp. 35-44

Memory Access Patterns of Occlusion-Compatible 3D Image Warping

William R. Mark Gary Bishop

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

Abstract

McMillan and Bishop's 3D image warp can be efficiently implemented by exploiting the coherency of its memory accesses. We analyze this coherency, and present algorithms that take advantage of it. These algorithms traverse the reference image in an occlusion-compatible order, which is an order that can resolve visibility using a painter's algorithm. Required cache sizes are calculated for several one-pass 3D warp algorithms, and we develop a two-pass algorithm which requires a smaller cache size than any of the practical one-pass algorithms. We also show that reference image traversal orders that are occlusion-compatible for continuous images are not always occlusion-compatible when applied to the discrete images used in practice.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation – Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Visible line/surface algorithms; I.3.1 [Computer Graphics]: Hardware Architecture – Graphics processors

Additional Keywords: image-based rendering, 3D image warp, occlusion-compatible warp order.

1 Introduction

Image-based rendering produces realistic-looking 3D graphics at a relatively low cost. The 3D image warp developed by McMillan and Bishop [12, 13] is particularly appropriate for low-cost implementation, because it requires a relatively small input data set. The 3D image warp has been used to explore natural scenes acquired using cameras [13] and to accelerate the display of computer generated imagery [5, 8, 9]. The alternative Lumigraph / Light Field approach to image-based rendering [4, 6] may produce higher quality output, but requires a very large input data set.

All implementations of the 3D warp to date have been software-based, and none has been faster than a few frames per second for 640 x 480 images. In order to inexpensively achieve 60 Hz frame rates in the near future while maintaining good image quality, the 3D warp will need to be supported by graphics hardware.

*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}

In this paper, we take the first step towards hardware support by analyzing the memory access patterns of the 3D warp. We discuss the cache size requirements of several one-pass 3D warping algorithms, and we present a new two-pass 3D warp algorithm which requires a smaller cache than most of the one-pass algorithms.

2 3D Warp

The 3D warp re-projects an image using a new view position, compensating for changes in both view direction and in view position. In this paper we refer to the original image as a *reference image*, and the computed image as an *output image*. The 3D warp requires per-pixel depth information from the reference image, as well as the usual color information. Simpler projective image warps do not use per-pixel depth information, and thus for general images can compensate only for changes in view direction, ignoring changes in view position. Wolberg's book [14] provides a good overview of projective warps, and of image warping in general.

The view interpolation system developed by Chen and Williams [3] used a 3D warp during pre-processing, but used a simpler interpolation algorithm at run-time. McMillan and Bishop developed a real-time 3D warp [12, 13]. Their system uses incremental evaluation of the 3D warp equations and an occlusion-compatible image-traversal order to achieve real-time performance. The rest of our presentation assumes that the reader understands this earlier work.

The occlusion-compatible image-traversal order is an important part of McMillan and Bishop's work. Because a 3D warp can map multiple reference-image pixels to a single output-image pixel, some means of arbitrating between the pixels is needed. Conventional Z-buffering can be employed, but it turns out to be unnecessary. Instead, occlusion ambiguities can be resolved by traversing the reference image in an appropriate order and using a painter's algorithm.

In addition to eliminating the need for Z-buffering, the occlusion compatible order can be used to provide a front-to-back or back-to-front order which allows for efficient anti-aliasing of the 3D warp. We have implemented Carpenter's A-buffer algorithm [1] in a software test-bed using a front-to-back order, providing super-sampled quality with only one bit of storage per sub-pixel. Gortler *et al.* [5] use a back-to-front order to composite pixels into the output image using a splat footprint with fractional alpha values.

In order to maintain the occlusion compatible order, certain restrictions must be followed in traversing the reference image. McMillan's dissertation [11] and papers [10, 12] discuss this point in detail. We provide a quick summary of these results here.

The projection of the output-image view position onto the reference-image plane defines a point, referred to in the computer vision literature as an epipole. There are two types of reference-image epipoles, positive and negative. The epipole type is determined by the relative positions of the reference-image view position, the output-image view position, and the reference-image view plane. If the epipole is negative, the back-to-front occlusion compatible order warps the epipole first, and moves radially outward. For a positive epipole, the order moves inward from the edges of the image, warping the epipole last (Figure 1).

More intuitively, if the output-image view position (the viewer) moves away from the reference-image view position while facing forward, objects will move away from the epipole. In this case, the back-to-front occlusion-compatible order must move inward from the edge of the reference image, towards the epipole.

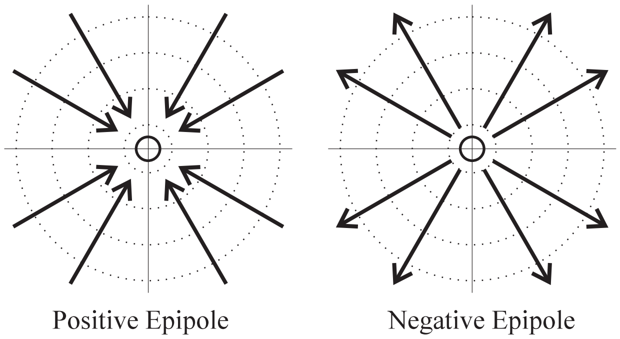

Figure 1: *The back-to-front occlusion compatible order moves towards a positive epipole and away from a negative epipole.*

The constraint on possible occlusions is even stronger than that implied by the inward or outward moving occlusion-compatible order just described. For a planar reference image, point A can only occlude point B if they lie on the same epipolar line (The epipolar lines are the lines radiating outward from the epipole). Thus, a warping order is occlusion compatible as long as it preserves the correct order along all epipolar lines.

This more flexible requirement allows occlusion-compatible orders that traverse large areas of the reference image in scan-line order. In the general case, the epipole divides the reference image into four *sheets* which can be traversed in scan-line order (Figure 2). If the epipole falls outside the borders of the reference image, then only one or two sheets will result.

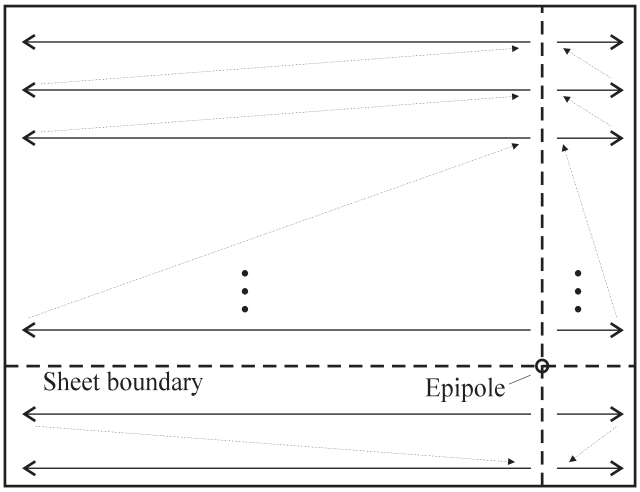

Figure 2: *The reference image can be divided into four occlusion-compatible sheets. Each sheet is traversed in a raster-like order.*

Traversing the reference image in sheets is much simpler and more efficient than trying to directly traverse the epipolar lines. Other sheet-like traversal orders are possible, a point we will return to later.

3 Coherency of 3D Warp

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 output 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 disparity value as well as the reference-image location. If these per-pixel disparity values do not change smoothly, then the mapping does not change smoothly either.

However, if we place bounds on disparity values and on the distance between reference-image and output-image centers of projection (the translation distance), then we can make the 3D warp behave more like a perspective warp. Discontinuities in the mapping are bounded in magnitude, so that a given reference-image pixel can potentially map to only a small set of output-image pixels. By characterizing this bound on discontinuity, we can design algorithms with efficient memory access properties.

In an actual system, the bound on disparity values comes from knowledge about the scene, and restrictions on the user's travel in the scene. Alternatively, one may think of the bound on disparity as the equivalent of a near clip plane. The bound on translation is determined in one of several different ways depending on the application. When reference images are pre-stored, the translation bound is calculated from the distance between reference images, and possibly from restrictions on the user's movement. In a post rendering warping system [8], it comes from a bound on the user's speed.

3.1 Bounded discontinuities

The bound on discontinuity in the 3D warp can be mathematically formulated in terms of the bounds on object distance and view-position translation. We begin this mathematical formulation by restating McMillan's 3D warp equation [11]:

$$\bar{x}_2 \doteq \delta(\bar{x}_1)\mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2) + \mathbf{P}_2^{-1}\mathbf{P}_1\bar{x}_1, \qquad (1)$$

where $\bar{x}_1$ is the reference-image location, $\bar{x}_2$ is the output-image location, and $\delta(\bar{x}_1)$ is the generalized disparity associated with the reference-image pixel. $\mathbf{P}$ and $\dot{C}$ represent the pinhole camera viewing parameters and center of projection respectively for the images. When both $\mathbf{P}_1$ and $\mathbf{P}_2$ represent planar images, then $\mathbf{P}_1$ and $\mathbf{P}_2^{-1}$ are 3x3 matrices, $\bar{x}_1$ and $\bar{x}_2$ are represented in projective coordinates, and $\doteq$ indicates projective equivalence. A divide is necessary to obtain 2D image coordinates from $\bar{x}_2$.

The second term in Equation 1 is a pure projective warp. Points $\bar{x}_1$ at infinite distance have $\delta(\bar{x}_1) = 0$, and thus are warped purely projectively. For these points at infinity, a coherent traversal of the reference image during warping will guarantee coherent accesses to the output image.

The first term in Equation 1 expresses the 3D warp's perturbation from a projective warp—in other words, the translational component of the 3D warp. It is this term that results in partially incoherent accesses to the output image during the warp even though we are coherently traversing the reference image. To examine this term alone, we can re-express Equation 1 as follows:

$$\bar{x}_2 \doteq \delta(\bar{x}_1)\mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2) + \bar{x}'_2, \qquad (2)$$

where $\bar{x}'_2$ is the location of $\bar{x}_1$ in the output image due to a pure projective warp ($\bar{x}'_2 = \mathbf{P}_2^{-1}\mathbf{P}_1\bar{x}_1$).

The perturbation to the pure projective warp is proportional to both the generalized disparity $\delta(\bar{x}_1)$ and the magnitude of the baseline, $\left\|\dot{C}_1 - \dot{C}_2\right\|$. 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 output-image pixels. We define $p = \max(\|\bar{x}_2 - \bar{x}'_2\|)$, where the magnitude is taken in image space (after the homogeneous divide). But first, we will determine a bound on the angle ϕ between $\bar{x}'_2$ and $\bar{x}_2$ interpreted in 3-space. Note that ϕ is independent of the projection manifolds $\mathbf{P}_1$ and $\mathbf{P}_2$, but p is not.

If T is the view-position translation distance ($T = \left\|\dot{C}_1 - \dot{C}_2\right\|$), and d is the distance from $\dot{C}_1$ to the closest object in the scene, then we can see from Figure 3 that

$$\phi = \sin^{-1}\left(\frac{T\sin(\alpha)}{\sqrt{d^2 + T^2 - 2dT\cos(\alpha)}}\right). \quad (3)$$

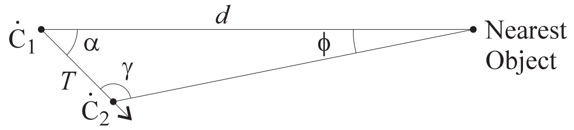

Figure 3: *Angular object movement across the field-of-view is a function of initial object distance and the distance between reference-image and output-image view positions.*

For a given d and T, ϕ is maximized when $\gamma = 90°$. In this worst case we have the simpler expression

$$\phi = \sin^{-1}\left(\frac{T}{d}\right) \quad (4)$$

and therefore,

$$\phi_{max} = \sin^{-1}\left(\frac{T_{max}}{d_{min}}\right) \quad (5)$$

For a planar output 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:

$$p = a_{max} \cdot \phi_{max}. \quad (6)$$

For on-center projections $\mathbf{P}_2$, this worst-case occurs at the corners of the display. If A represents the $horizontal/vertical$ aspect ratio, v represents the vertical field-of-view in radians, and V represents the vertical pixel count, then

$$a_{max} = \frac{V}{2}\cot\left(\frac{v}{2}\right)\left[\left(A^2+1\right)\left(\frac{1-\cos(v)}{1+\cos(v)}\right)+1\right], \quad (7)$$

which as $v \to 0$, approaches V/v as expected.

In summary, we can calculate the worst-case screen-space movement (p) of objects due to view-position translation from five parameters. Three parameters, A, V, and v, describe the display and its field-of-view. The fourth parameter, d_{min}, is the minimum distance from the reference-image view position to the nearest object. The final parameter, T_{max}, is the maximum distance between the reference-image view position and the output-image view position. Table 1 shows calculated values of p for several different sets of conditions, based on Equations 5 through 7 and the use of square pixels.

3.2 Direction of pixel movement

We are interested in knowing the *direction* of pixel movement in the output image as well as the magnitude of the movement. Equation 2 describes both the direction and magnitude of movement caused by view-position translation. If we define the *output-image* epipole $\bar{e}_o$ in homogeneous coordinates as

$$\bar{e}_o = \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2),$$

then the translational pixel movement can be expressed in terms of $\bar{e}_o$. The movement is always either directly towards or directly away from e_o, where e_o represents $\bar{e}_o$ after the homogeneous divide. The movement is towards e_o if the z component of $\bar{e}_o$ is positive, and away from e_o if the z component is negative.

Display	vFOV	T_{max}	d_{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 1: *Worst-case screen-space movement of objects due to view-position translation.*

Figure 4 shows how the 3D warp maps reference-image points of unknown depth to the output image. Each point in the reference image has a corresponding point in the output image which would result from a purely projective warp. This purely projective warp is equivalent to a 3D warp on a point at infinite distance ($\delta = 0$). Extending from each of these points in the output image is a line segment which shows the potential 3D warp mappings caused by point-distances less than infinity ($\delta > 0$), but greater than d_{min}. The line thus represents the 3D warp's perturbation from the projective warp. This perturbation is caused by view-position translation, and is larger for points that are closer. Its maximum magnitude anywhere in the output-image is p, as discussed earlier. The output-image epipole depicted in Figure 4 is a positive epipole, so the movement is towards this epipole.

The maximum output-image movement, p can only occur at few locations in the output image. The possible locations occur where $\alpha \approx 90°$, i.e. when the pixel is far away in output-image space from both output-image epipoles. For the maximum movement to occur, the pixel must also be in a corner of the output image. For illustration purposes, most of the figures in this paper show the maximum movement distance of p even when these conditions do not completely hold. We always use p in our cache size calculations because we are interested in the worst-case cache size.

Figure 5 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 output image. In our cache size calculations, we use a worst-case rectangular estimate of this area.

Note that Figures 4 and 5 (and the rest of the figures in this paper) show an approximately 1-to-1 scaling between reference-image pixels and output-image pixels. Because the fields of view and resolutions of the reference and output images can be different, other scalings are possible (with appropriate reconstruction). Our equations make no assumptions about the scaling.

4 One-Pass 3D Warps

To implement the 3D warp inexpensively, we want to maintain an occlusion compatible order, while achieving the following three goals:

- A small working set size, and thus a small required cache size.
- A slowly changing working set, thus minimizing cache to main memory traffic.
- A large size for each cache to main memory transfer (especially important for block-transfer oriented memories like RAMBUS).

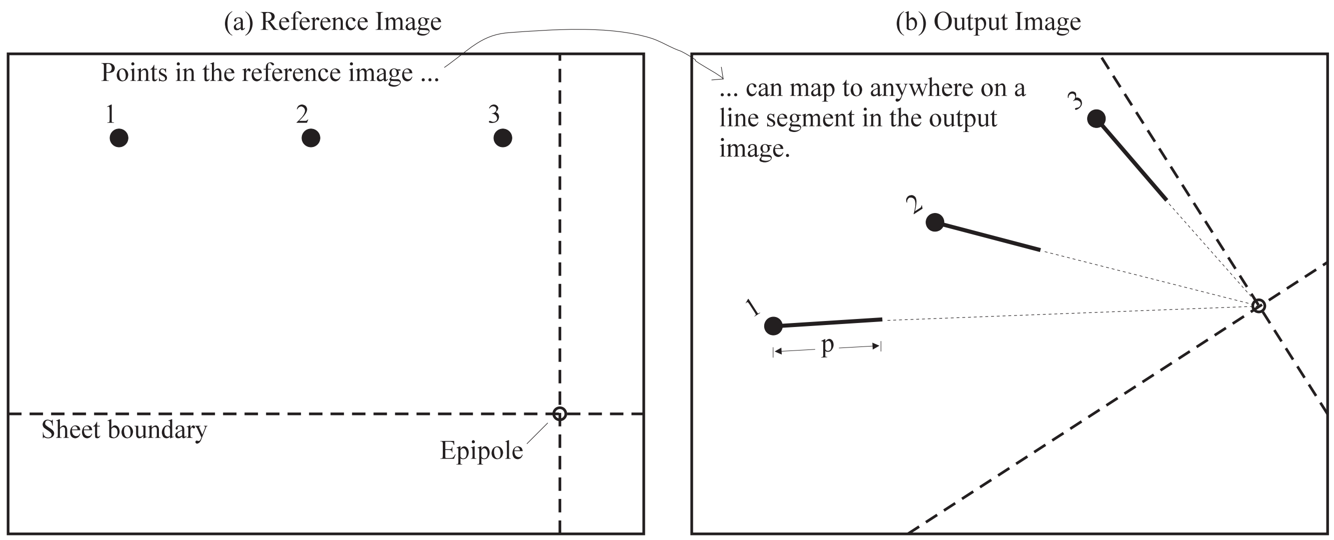

Figure 4: *3D warp of points. A point in the reference image can map to anywhere on a line segment in the output image. The actual location on the line depends on the point's disparity value. The point at one end of these output-image line segments shows the mapping that would result from a purely projective image warp.*

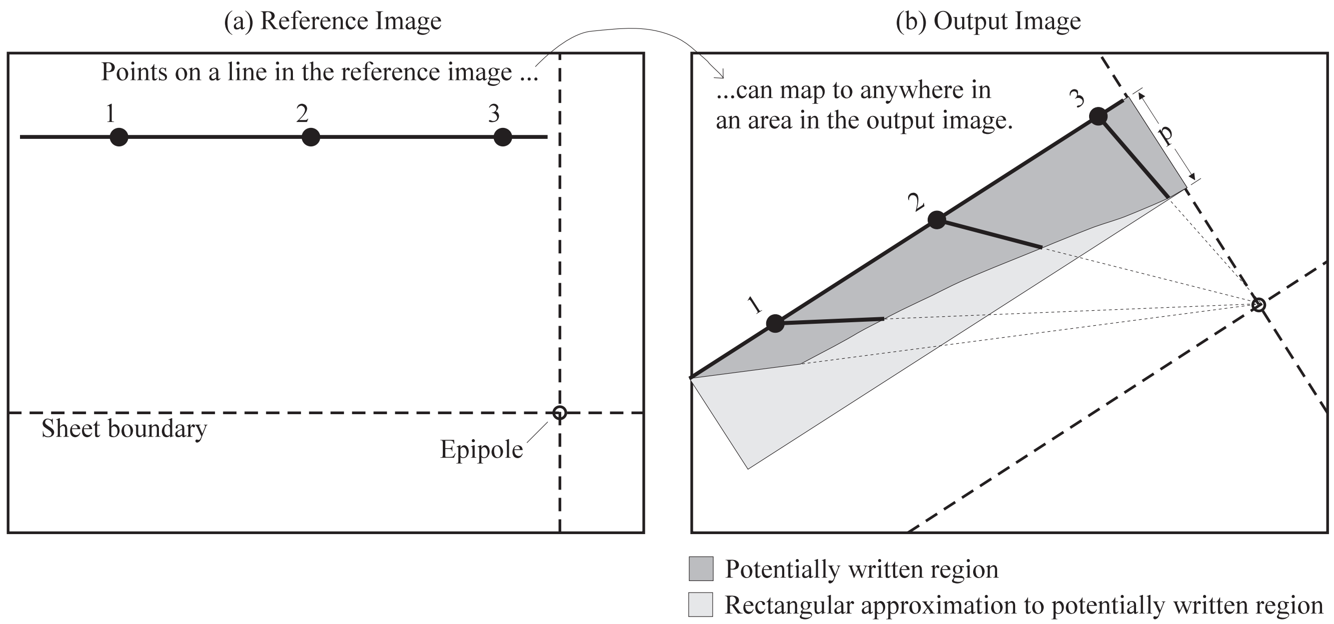

Figure 5: *3D warp of a line. The points on a reference-image line can map to anywhere in an area in the output image. In our analysis, we approximate this area with a worst-case rectangular region.*

If disparities and view-position translation are bounded (thus bounding p), we would like to guarantee a minimum image warping rate, for *any* reference images which conform to the bounds. In order to provide such a performance guarantee, the bound p must be sufficient to in turn imply a bound on the number of cache misses and memory accesses. We determine this implied bound by performing a worst-case analysis of the memory access patterns of the 3D warp algorithms as a function of p.

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 can not use the standard raster organization for storing the output image in main memory. In any particular warp, the raster organization is inefficient for those portions of the output image where the epipolar lines are nearly perpendicular to the raster lines. In these portions of the output image, a single reference pixel could map to any one of many scattered memory locations. Therefore, we organize the output image into square blocks, and store all pixels from each block in adjacent locations in main memory (Figure 6). These blocks are most naturally chosen to be same size as a cache block. Thus, two pixels which are near each other in the 2D image are likely to reside in the same cache block.

To improve clarity, most of the rest of our discussion does not explicitly deal with the fact that cache blocks are greater than one pixel in size. We do however implicitly assume this fact by arguing that the entire area of the output 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 in any particular warp (there are more potentially touched 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 as 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.

Neglecting the size of cache blocks can also cause us to underestimate the amount of traffic between the cache and main memory. The reason is that cache blocks are always transferred in their entirety, even if only a few pixels in the block belong to the working set. However, when an algorithm's working set in the output image is approximately square and $p \gg \sqrt{cacheblocksize}$, this simplification has minimal impact on our memory-traffic analysis. It is acceptable to make the assumption that $p \gg \sqrt{cacheblocksize}$, since reasonable values for these variables are $p = 100$ and $\sqrt{cacheblocksize} = \sqrt{256}$ bytes $= 8$ pixels. When these assumptions are violated (in particular if the working set is very long and thin), we will discuss the issue further.

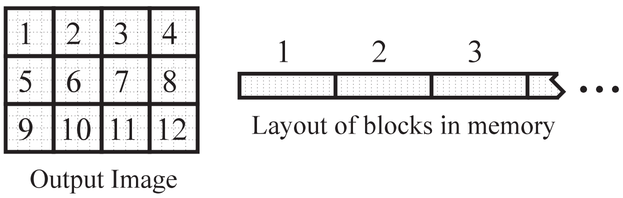

Figure 6: *Pixels belonging to 2D blocks of the output image are stored contiguously in memory. A standard raster organization of the output-image pixels in memory would result in poor utilization of the cache.*

4.1 Standard Order

The usual occlusion-compatible sheet-traversal order is a line-at-a-time raster scan of the sheet [12]. For a worst-case reference image that conforms to our pixel-movement bounds, this traversal order requires a large cache to avoid thrashing. Figure 7 illustrates this traversal order and its cache requirements. The cache must be able to hold the entire region of the output 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 output image. Since a sheet can potentially occupy the entire output image, the maximum sheet width is equal to S, where S is the diagonal size of the output image. Therefore, the required cache size is $S \cdot p$ pixels. With a cache of this size, each pixel of the output image is transferred from main memory to the cache and back no more than once.

4.2 Improved Order

We can reduce the cache size slightly by changing to a different occlusion-compatible traversal order of the sheet. Figure 8a shows this traversal order, and Figure 8b shows the resulting memory access pattern in the output image. The cache must still hold the region of the output image touched by a "line" from the reference image, but these lines are now shorter, of width W. The required cache size would seem to be $(W + p) \cdot p$ pixels. Unfortunately, even this size isn't sufficient to hold the overlap between adjacent *columns* of the traversal pattern.

There are two alternatives to addressing this problem. One is to simply pay the price of reloading the overlap region as we traverse each column. If $W = p$, then each output-image pixel is potentially loaded and flushed from the cache an extra time. If $W \gg p$, then most pixels are only loaded and flushed once, but the cache size approaches that required for the standard pattern. The second alternative is to increase the cache size to cover the entire overlap region. The extra cache storage required is $(sheetheight) \cdot p$. In summary, if we tolerate reloads, then the required cache size is $O(p^2)$, and if not, then it is $O(Sp)$, where S is diagonal size of the output image.

4.3 Epipolar line traversal

In theory, we could use an output-pixel cache size of p if we traversed the reference image along epipolar lines (Figure 9). In practice, this strategy is very troublesome. In a discrete reference image, all epipolar lines converge to a single pixel at the epipole. Independently traversing all lines between pixels at the edge of the image and the epipole would warp most pixels more than once. We would need a complicated line drawing algorithm that determines which pixels near the epipole belong to which of the discrete lines, and thus only warps each pixel once.

More seriously, traversing epipolar lines one at a time in a *discrete* image can actually introduce occlusion errors. Figure 10 illustrates how this type of error can occur. Pixels from later-drawn lines can incorrectly overwrite pixels from earlier-drawn lines, due to the compression of the epipolar lines near the positive epipole. The problem is especially acute if a splat-type reconstruction algorithm [5, 7] is used, which can expand one reference pixel into several output pixels. This overwriting also indicates that a cache size of p pixels is not in fact sufficient—the potentially overwritten pixels need to be held in cache as well. To avoid repeated trips by pixels to the cache, the cache would have to be somewhere between p and $S\sqrt{2} + p$ pixels in size.

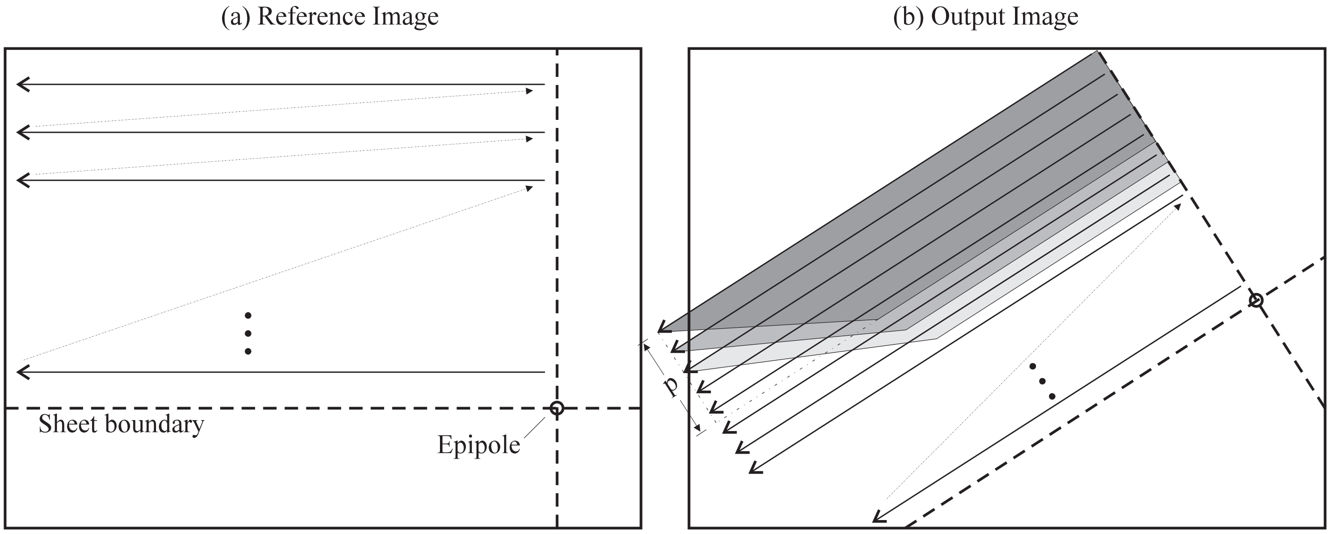

Figure 7: *Standard one-pass occlusion-compatible order. The potentially-written areas in the output image which are produced by different reference-image scan lines overlap almost completely. We show these areas using their worst-case rectangular approximations, with one corner cut off to allow the overlap to be more easily seen.*

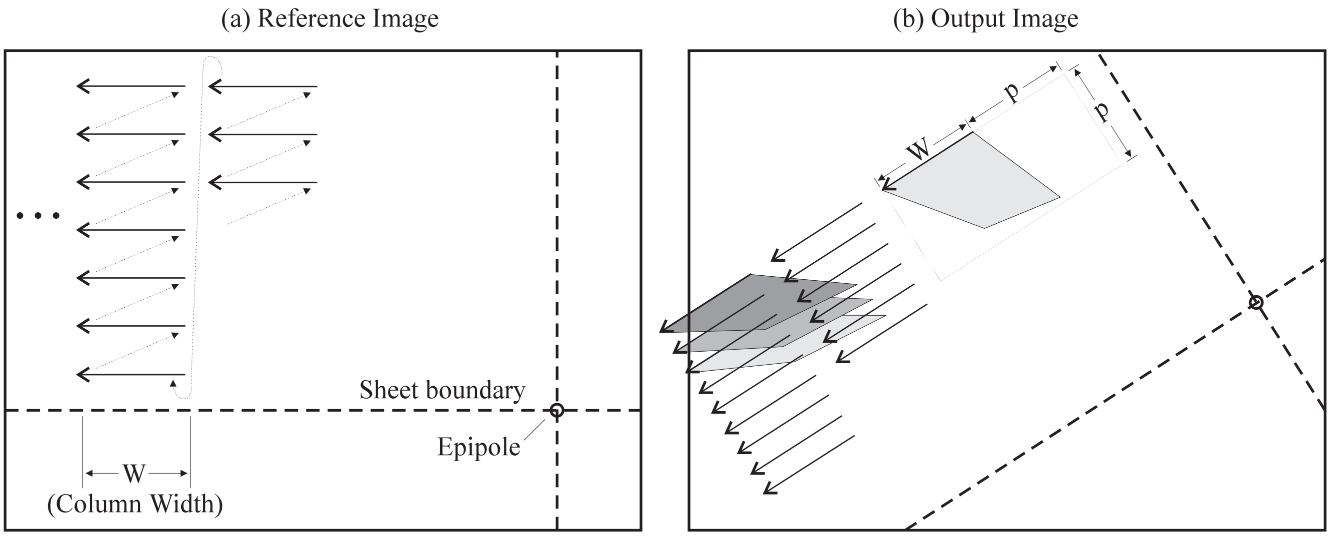

Figure 8: *Revised one-pass occlusion-compatible order. The reference image is traversed one column at a time, with each column traversed in a raster-like order. The overlapping potentially-written areas which correspond to the raster lines are shown superimposed on the output image. Note that these areas overlap both* within *a column and* between *adjacent columns.*

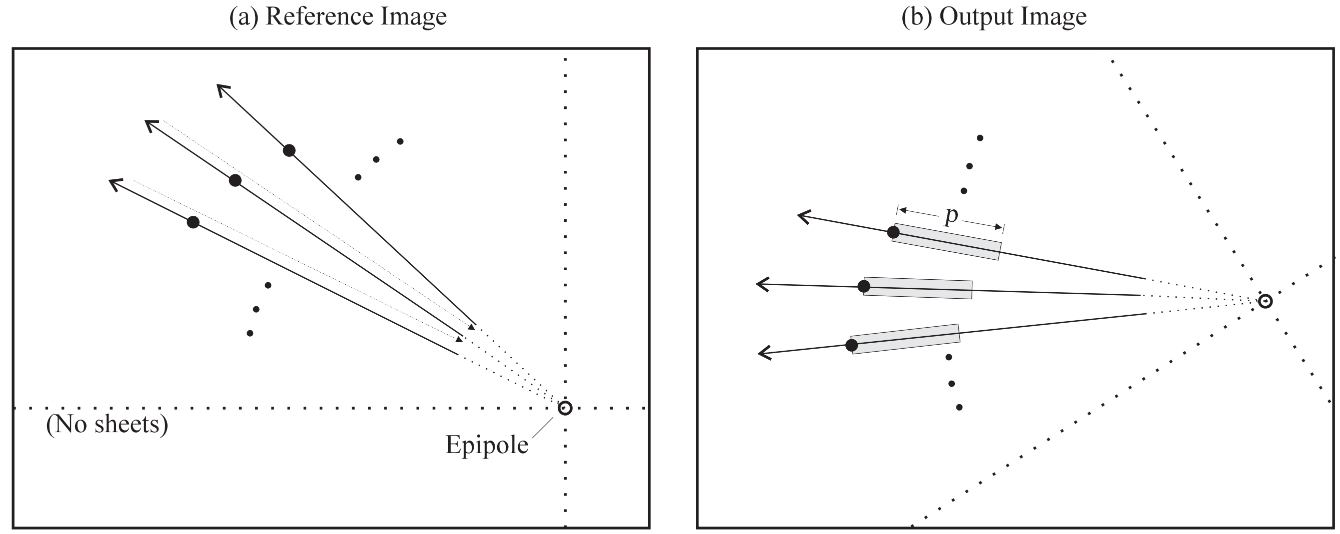

Figure 9: *Epipolar one-pass occlusion compatible order. The output image shows the potentially written areas (really just lines) corresponding to the indicated reference-image points.*

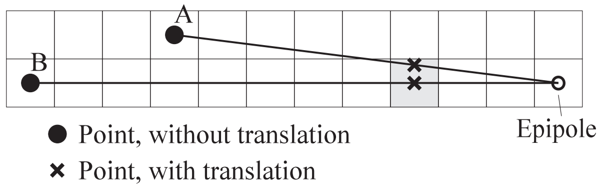

Figure 10: *Warping by epipolar lines is occlusion compatible for continuous images, but is not necessarily occlusion compatible for discrete images. The 3D warp maps both point A and point B to the grey pixel. If point A's epipolar line is traversed after point B's epipolar line, then A will incorrectly occlude B.*

This same type of occlusion error can also occur with the conventional raster-scan traversal of discrete image sheets. Errors can occur near the edge of the sheet where the minor traversal direction becomes nearly parallel to the epipolar lines. But with a raster-scan traversal, the problem is at least restricted to a small portion of the reference image, instead of occurring everywhere. It could be avoided almost entirely by dividing the image into eight sheets instead of four sheets, so that the minor traversal direction is always at least 45° degrees away from the epipolar direction (Figure 11). This change can be made to any of the 3D warp algorithms we present in this paper, with the exception of the epipolar line traversal algorithm.

An additional problem with the epipolar order is caused by the fact that cache blocks are normally larger than one pixel. The narrow lines through the output image would hit only a few of the pixels in each accessed cache block. This problem could be mitigated, at the expense of additional complexity, by traversing several adjacent epipolar lines simultaneously—i.e. traversing "wide" epipolar lines.

5 Two-pass 3D Warp

The problem with the one-pass occlusion compatible techniques (except for the epipolar line technique) is that their working set protrudes in a direction other than the direction of traversal. This protruding portion of the working set has to be held in the cache until the algorithm revisits that area of the image, or else has to be flushed from the cache and then reloaded later. The first alternative results in a large cache size, and the second results in increased main memory traffic.

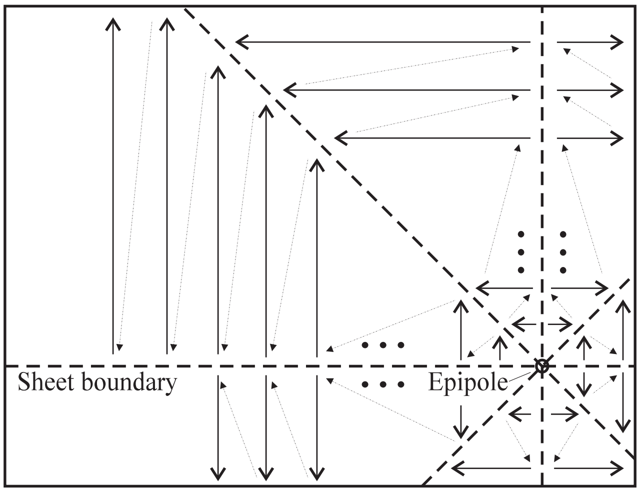

Figure 11: *Dividing into eight sheets rather than four almost entirely eliminates errors in the occlusion-compatible order due to image discretization.*

A solution to this dilemma is to use a two-pass technique, in which the first pass partially sorts pixels in one of the dimensions. We perform the partial sort by grouping pixels into bins. The first pass of our algorithm bin-sorts pixels by their post-translation, pre-rotation X-coordinate. This X-coordinate can be thought of as the reference image X-coordinate, but after translation has occurred. Each sort bin is b pixels wide. This first pass of the algorithm is described by Equation 8 and Figure 12. The choice of pre-rotation rather than post-rotation X-coordinates minimizes the rate at which the working set of bins changes during the first pass.

The first pass of the algorithm also evaluates Equation 1 for each reference-image pixel to obtain $\bar{x}_2$, but does *not* move the pixel accordingly. The value of $\bar{x}_2$ (after the homogeneous division) is stored with each pixel for use during the second pass.

$$bin = \left\lfloor \frac{\bar{x}_1 + \delta(\bar{x}_1)\mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2)}{b} \right\rfloor \qquad (8)$$

Algorithm	Occlusion Problems?	Cache size (Bytes)	Cache size O()	Example cache sizes (KB)	Mem BW (Bytes/pixel)	Example Mem BW's (MB/sec)
Standard one-pass	n	$4Sp$	$O(Sp) \geq O(p^2)$	257, 1100	12	221, 944
By-columns one-pass (A)	n	$4 \cdot 2p^2$	$O(p^2)$	51, 217	20	369, 1600
By-columns one-pass (B)	n	$4Sp$	$O(Sp) \geq O(p^2)$	257, 1100	12	221, 944
Epipolar lines one-pass	y	$4p$, at best	$O(p)$, at best	0.3, 0.7	12	221, 944
Two-pass	n	$4bp + Yp/b + 2Y$	$O(p)$	7, 14	24	442, 1900

Table 2: *Comparison of cache sizes and main-memory bandwidth for warping algorithms. The variable S is the diagonal size of the output image in pixels. The variable p is the maximum pixel movement due to view-position translation. Typically $S \gg p$. Two versions of the one-pass "by-columns" algorithm are listed. Version A uses a small cache, but requires two cache visits for each output pixel. Version B uses a larger cache which requires only one cache load for each output pixel. The table provides numerical cache sizes and memory bandwidths for two example output image sizes. The first entry in the example columns corresponds to a 640 x 480 output image, and the second entry corresponds to a 1280 x 1024 output image. For both example sizes, we set $T_{max}/d_{min} = 0.1$, the vertical field-of-view to $60°$, $b = 16$ pixels, and $Y = 256$ bytes.*

The second pass of the algorithm completes the 3D warp by moving each pixel to the previously computed location $\bar{x}_2$. The pixel bins from the first pass are read one at a time and warped into the output image (Figure 13). The pixels in each bin are warped in the same order that they were placed into the bin originally, so that the occlusion compatible order of the first pass traversal is maintained.

Our two pass algorithm was inspired by earlier two pass image-warping techniques. Catmull and Smith [2] developed a 2-pass technique for several classes of image warp, including planar-to-planar perspective warps. Many other researchers have extended this work; Wolberg's book [14] provides a good overview. Our two-pass technique differs from this earlier work in two ways. First, our algorithm is designed for the 3D warp rather than a projective warp. Second, the output of our first pass is a set of bins, rather than a complete image.

In our algorithm, each bin from the first pass can contain contributions from several of the raster-scanned reference image columns. The contributions from each such column are grouped together in the bin. Unfortunately, this grouping means that during the second pass of the algorithm, several traversals are made through the stripe in the output image corresponding to the bin. To avoid repeated trips by each output-image pixel to and from the cache, the cache would have to be big enough to hold an entire stripe of the output image.

If we require that $W \geq p + b$, then no more than two traversals are made though each stripe in the destination image. With a slight modification to our algorithm, we can perform both of these traversals simultaneously. During the first pass, we mark the point in the bin at which we start writing pixels from a second column into the bin. In the second pass, we simultaneously traverse the first and second portions of the bin. The occlusion-compatible order is maintained by requiring that the second-portion traversal always stay slightly behind the first-portion traversal (in output image space).

With this modification, the output-image cache need only be $b \cdot p$ pixels in size, and output pixels only enter and exit this cache once. For this two-pass algorithm, we also need to consider the first pass cache size. There will be p/b active bins at any one time. If the cache block size is Y *bytes*, then the required cache size is $(p/b) \cdot Y$ bytes.

Additional cache memory of $2 \cdot Y$ bytes is required by the input to the second pass of the algorithm. This cache memory holds the two active cache blocks of the bin which is being processed.

If $b = O(1)$ and $Y = O(1)$, then the first-pass cache size is $O(p)$, and the second-pass cache size is also $O(p)$. Thus, the cache sizes for the two-pass technique are asymptotically better than the $O(p^2)$ cache sizes required by most of the one-pass techniques.

Although the cache size is smaller, we pay a penalty in the cache to main-memory bandwidth. The additional expense is that of writing the bins produced by pass one, and reading these bins during pass two. Note that it would be possible to use an entirely separate memory for these bins, which might be cheaper than trying to double the bandwidth of a single main memory that also holds the output image. The linear access to the bins makes it feasible to entropy encode their contents. However, a worst-case analysis must assume that this encoding is only moderately successful (some redundancy is expected in the output-image coordinates even in the worst case, due to the partial coherence of the warp).

We have yet to discuss how to choose the value of b, the bin widths, except to state that $b = O(1)$, rather than $O(p)$. If b is too small, then the bin's stripe in the output image is narrow, so that the discretization of the output image into cache blocks causes problems. In this case, most of the cache blocks that are touched by the thin stripe are only partially covered by it. Additionally, when a splat-type reconstruction is used that can map one reference pixel to several output-image pixels, excessively narrow bins require storing a large percentage of pixels into two bins.

If b is too large, then the second pass's cache size for the output-image pixels becomes too large. The appropriate tradeoff depends on variables such as the size of a cache block.

6 Algorithm Comparison

Table 2 summarizes the worst-case cache sizes and memory bandwidths required by the different occlusion compatible algorithms we have examined. The epipolar line one-pass traversal has the best combination of cache-size and memory bandwidth requirements, but is difficult to implement and in practice does not always correctly maintain an occlusion-compatible order. Our two-pass algorithm is the only other choice which allows an $O(p)$ cache size, but this small size comes at the expense of a higher memory bandwidth requirement, and a more complicated algorithm.

The preferred algorithm in any given instance will depend on a variety of parameters, including the relative expense of cache memory versus main-memory bandwidth.

Note that Table 2 makes a number of estimates and assumptions in calculating its numerical results. First, we assume a warping rate of 60 frames/sec. Second, we use a one-to-one scaling of input-image pixels to output-image pixels. Third, we make assumptions about the number of bytes required to store input and output pixels. Input-image pixels are eight bytes (RGB and depth) and output-image pixels are four bytes (RGBA). The intermediate "pixels" stored between the first and second passes of the two-pass warp are six bytes (RGB, and compressed $\bar{x}_2$). These pixel sizes are used to determine the bytes of memory bandwidth per pixel per warp ("Mem BW (Bytes/pixel)").

We make two additional assumptions in calculating Table 2. First, in calculating the numerical memory bandwidth values we assume that the reference image is perfectly clipped (only pixels that will be visible are read), and that the output image is completely

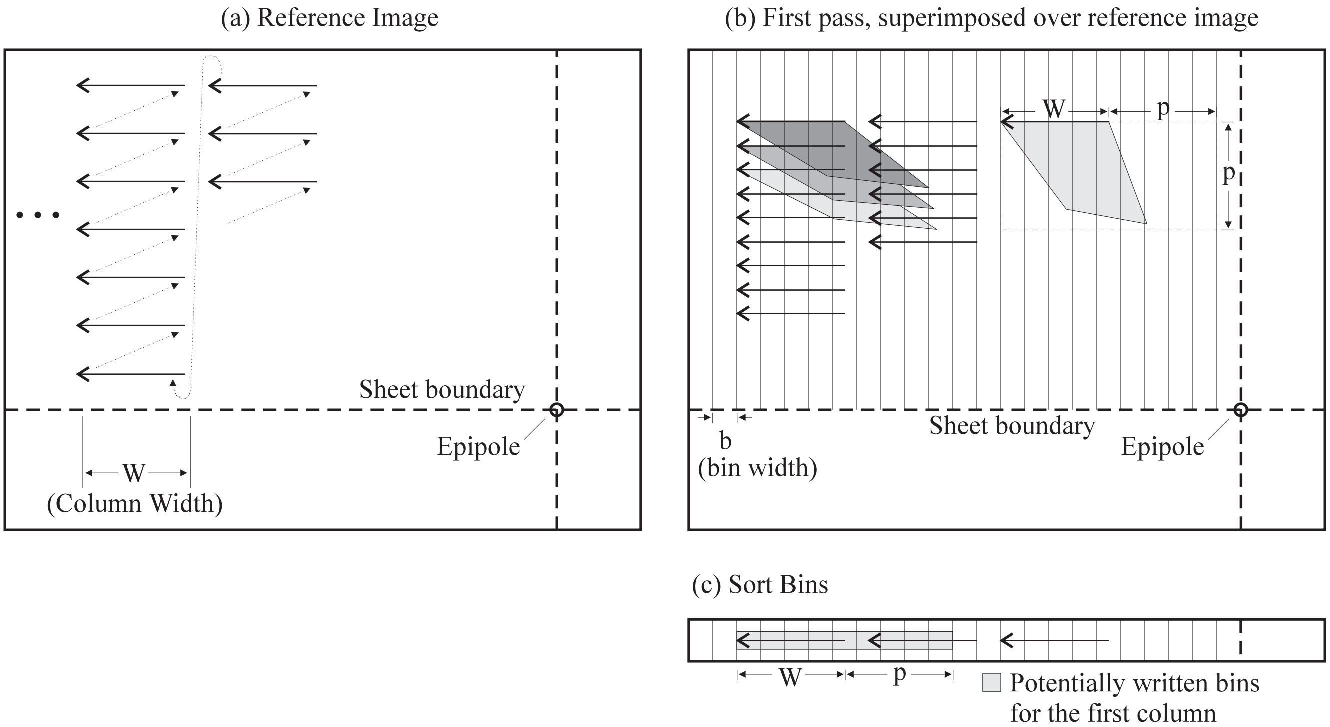

Figure 12: *First pass of the two-pass warp. Reference-image pixels are sorted into bins based on their post-translation, pre-rotation X-coordinate. (a) shows the reference image, (b) shows the regions of potential translation superimposed on the reference image, and (c) shows the bins which are filled during the first pass.*

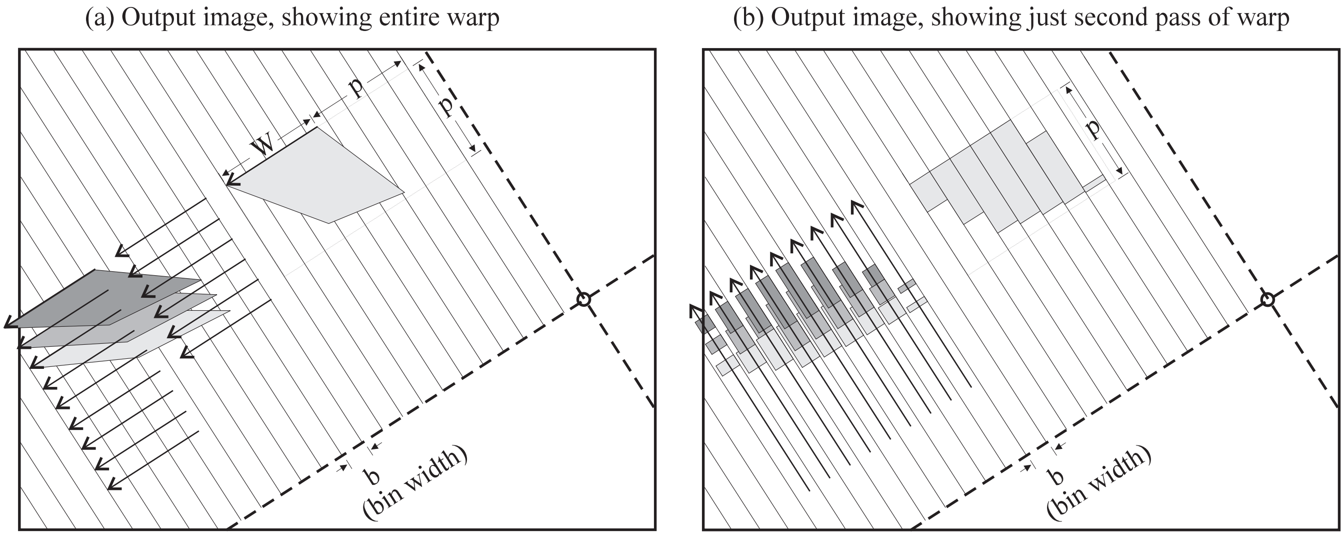

Figure 13: *Second pass of the two-pass warp. (a) shows the entire warp in output-image space. (b) shows just the second pass of the warp in output-image space. In the second pass, the bins from the first pass are traversed one at a time, and the pixels from the bins are written into the output image.*

filled. Second, we neglect memory-bandwidth and cache inefficiencies caused by the fact that output-image cache blocks are normally greater than one pixel in size. Part of this second assumption is that for all algorithms except version A of the one-pass by-columns algorithm, output-image pixels are never loaded into the cache from main memory—they need only be written, since we know that they were previously untouched.

The main point of Table 2 is not to give precise numerical results, but to show that at reasonable resolutions, the $O()$ differences in cache sizes between the different algorithms are in fact important. We also want to convey a sense for the order-of-magnitude of required cache sizes and memory bandwidths.

7 Discussion and Conclusions

Before trying to design hardware to implement an algorithm, it is important to understand the algorithm as completely as possible. Our goal in conducting this work has been to better understand the memory access properties of the 3D warp, and to explore some possible implementation-oriented algorithms. More analysis would be required before designing hardware for any of these strategies, but we have provided a framework for conducting this analysis.

Although we have focused on occlusion-compatible orders because they avoid Z-buffering and allow inexpensive anti-aliasing, many of the ideas developed in this paper apply to non-occlusion-compatible orders as well. The analysis of memory access patterns is also important for parallelizing the 3D warp and for developing inverse-mapped implementations of the 3D warp.

We have conducted this analysis in the context of the planar-to-planar 3D warp, but most of the ideas could be easily extended to the cylindrical-to-planar 3D warp. Our analysis has avoided extensive discussion of the reconstruction problem, but it could be easily adjusted to account for different reconstruction strategies, since the underlying warp equation remains the same.

We have concentrated on a worst-case analysis, so that given some *a priori* bounds, hardware can be designed that will guarantee a desired performance. Because most scenes have much greater local coherence than that implied by the overall bounds, the worst case is not the typical case. If guaranteed performance is not important, then analysis of typical scenes for a given application could be used to determine cache sizes smaller than those specified here.

The guarantees that our analysis provides do have an additional benefit. They enable the use of a simple software-managed cache, rather than a hardware-managed cache. Our analysis provides the information needed to load the software-managed cache with exactly the right cache blocks at the appropriate time.

We consider the major contributions of this paper to be the following:

- Development of a framework for analyzing cache requirements of 3D warps based on bounds on disparity and translation.
- Actual analysis of several 3D warping algorithms.
- Development of a new 2-pass 3D warp algorithm.
- Discussion of errors that can occur when theoretically occlusion-compatible warping algorithms are applied to actual discrete images, and presentation of a new 8-sheet division of the reference image that almost entirely eliminates these errors.

8 Acknowledgements

Leonard McMillan has contributed greatly to our understanding of image-based rendering, laying the groundwork that made this work possible. Bill began this work as a final project for Turner Whitted's graphics architecture class, and we thank Turner for providing such a convenient opportunity to explore the topic. Fred Brooks, Henry Fuchs, Steve Molnar, and the entire UNC image-based rendering group have furnished helpful advice and encouragement.

Finally, we would like to thank our generous sponsors. This research was supported by fellowships from the Microsoft Corporation and the Link Foundation, and by DARPA, under contract #DABT 63-93-C-0048.

References

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

[2] Ed Catmull and Alvy Ray Smith. 3-D transformations of images in scanline order. *Computer Graphics (Proceedings of SIGGRAPH 80)*, 14(3):279–285, July 1980.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Leonard McMillan. Computing visibility without depth. Technical Report UNC-CH TR95-047, University of North Carolina at Chapel Hill, Dept. of Computer Science, 1995. Available at http://www.cs.unc.edu/Research/tech-reports.html.

[11] 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.

[12] 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.

[13] 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.

[14] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California, 1992.