SEPARABLE IMAGE WARPING WITH SPATIAL LOOKUP TABLES

George Wolberg
Terrance E. Boult

Department of Computer Science
Columbia University
New York, NY 10027
[wolberg | tboult]@cs.columbia.edu

April 1989
Technical Report CUCS-398-89

ABSTRACT

Image warping refers to the 2-D resampling of a source image onto a target image. In the general case, this requires costly 2-D filtering operations. Simplifications are possible when the warp can be expressed as a cascade of orthogonal 1-D transformations. In these cases, separable transformations have been introduced to realize large performance gains. The central ideas in this area were formulated in the 2-pass algorithm by Catmull and Smith. Although that method applies over an important class of transformations, there are intrinsic problems which limit its usefulness.

The goal of this work is to extend the 2-pass approach to handle arbitrary spatial mapping functions. We address the difficulties intrinsic to 2-pass scanline algorithms: bottlenecking, foldovers, and the lack of closed-form inverse solutions. These problems are shown to be resolved in a general, efficient, separable technique, with graceful degradation for transformations of increasing complexity.

This work was supported in part by NSF grant CDR-84-21402, NSF grant IRI8800370, and DARPA grant N00039-84-C-1065.

1. INTRODUCTION

Image warping is a geometric transformation that redefines the spatial relationship between points in an image. This area has received considerable attention due to its practical importance in remote sensing, medical imaging, computer vision, and computer graphics. Typical applications include distortion compensation of imaging sensors, decalibration for image registration, geometrical normalization for image analysis and display, map projection, and texture mapping for image synthesis.

Image warping has benefited dramatically from developments in separable geometric transformation algorithms. Also known as scanline algorithms, these methods reduce a 2-D resampling problem into a sequence of 1-D scanline resampling operations. This makes them amenable to streamline processing and allows them to be implemented with conventional hardware. Scanline algorithms have been shown to apply over a wide range of transformations, including perspective projection of rectangles, bivariate patches, and superquadrics [8]. Hardware products such as the Ampex ADO and Quantel Mirage, are based on these techniques to produce real-time video effects for the television industry.

The central ideas to scanline algorithms are presented in the seminal paper by Catmull and Smith [1]. They describe a 2-pass technique that decomposes the 2-D resampling problem into two orthogonal 1-D resampling stages. This is the basis for almost all of the other separable work. Nevertheless, their approach suffers from problems known as "bottleneck" and "fold-over" — difficulties which can result in visual artifacts and costly memory requirements.

This paper introduces a novel scanline algorithm which properly addresses these limitations. As a result, we demonstrate that all geometric transformations can be realized with the algorithm derived herein. This extends the benefits of separable transformations to efficiently process arbitrary spatial mappings — geometric transformations that, until now, required costly 2-D resampling operations.

Section 2 of this paper introduces separable geometric transformation algorithms with special emphasis on the Catmull-Smith algorithm. Section 3 describes the novel scanline warping algorithm. Examples are presented in section 4. Finally, section 5 discusses conclusions and future work.

2. BACKGROUND

Separable geometric transformation algorithms, also known as *scanline algorithms*, spatially transform 2-D images by decomposing the mapping into a sequence of orthogonal 1-D transformations. This implies that the mapping function is separable, i.e., each dimension can be resampled independently of the other. For instance, 2-pass scanline algorithms typically apply the first pass to the image rows and the second pass to the columns. Although classic scanline algorithms cannot handle all possible mapping functions, they can be shown to work particularly well for an interesting class of transformations.

The primary motivation for scanline algorithms is efficiency. Traditionally, geometric transformations have been formulated as either forward or inverse mappings operating entirely in 2-D. While either forward or inverse mappings can be used to realize *arbitrary* mapping functions, they are costly. Separable algorithms present an alternate technique that, for a small decrease in accuracy, yield significant computational savings.

2.1. DEFINITIONS

A *spatial transformation* defines a geometric relationship between each point in the input and output images. The general mapping function can be given in two forms: either relating the output coordinate system to that of the input, or vice versa. Respectively, they can be expressed as

$$[x, y] = [X(u,v), Y(u,v)], \tag{2.1a}$$

or

$$[u, v] = [U(x,y), V(x,y)] \tag{2.1b}$$

where $[u,v]$ refers to the input image coordinates corresponding to output pixel $[x,y]$, and X, Y, U, and V are mapping functions that uniquely specify the spatial transformation. Since X and Y map input onto output, they are referred to as the forward maps. Similarly, the U and V functions are known as the inverse mapping since they map the output onto the input.

2.1.1. Forward Mapping

The *forward mapping* consists of interpolating each input pixel into the output image at positions determined by the X and Y mapping functions. Each input pixel is passed through the spatial transformation where it is assigned new output coordinate values. Since this is a mapping from pairs of integers to pairs of reals, filtering is required to generate the output image.

The real-valued output positions assigned by X and Y present complications. For discrete images, pixels are taken to be finite elements defined to lie on a (discrete) integer lattice. Implementing the spatial transformation as a point-to-point mapping can give rise to two types of problems: holes and overlaps. Holes, or patches of undefined pixels, occur when mapping contiguous input samples to sparse positions on the output grid. In contrast, overlaps occur when

consecutive input samples collapse into one output pixel.

The shortcomings of point-to-point mappings are avoided using a four-corner mapping paradigm. This considers input pixels as square patches assumed to be transformed into quadrilaterals in the output image. Because the projection of an input pixel is free to lie anywhere in the output image, the projections often straddle several output pixels, or lie embedded in one. These two instances are illustrated in Fig. 2.1. An *accumulator array* is required to properly integrate all input pixel fragments that contribute to each output pixel. Partial contributions are handled by scaling the input intensity by the relative area of the output pixel that it covers. Thus, each position in the accumulator array evaluates $\sum_{i=0}^{N} w_i f_i$, where f_i is the input value, w_i is the weight reflecting its coverage of the output pixel, and N is the total number of deposits into the cell. Note that N is free to vary among pixels and is determined only by the mapping function and the output discretization.

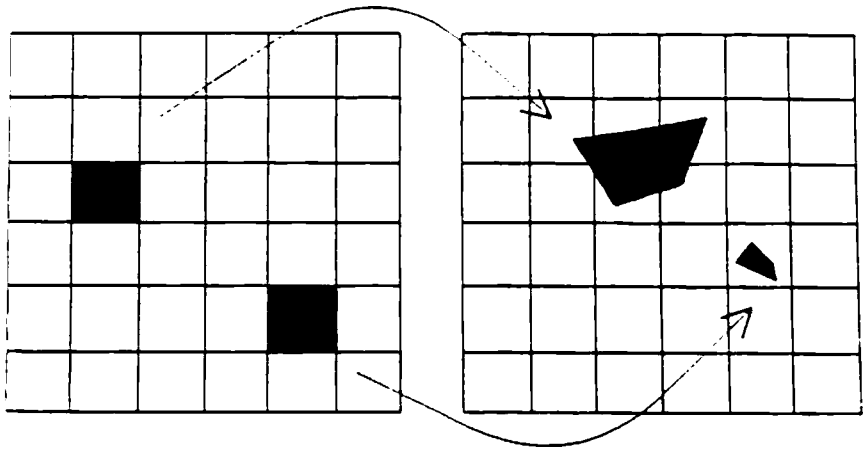


Figure 2.1: Accumulator array.

There are two main problems in the forward mapping process. First, costly intersection tests are needed to derive the area coverage. Second, additional filtering is necessary to ensure that a single input value is correctly handled when undergoing magnification. For a complete review see [5, 11].

2.1.2. Inverse Mapping

The *inverse mapping* operates in screen order, projecting each output coordinate into the input image via U and V. The value of the data sample at that point is copied onto the output pixel. Again, filtering is necessary to combat aliasing artifacts. The principal advantages of this method are that no accumulator array is necessary and output pixels which lie outside a clipping window need not be evaluated. This method is useful when the screen is to be written sequentially, U and V are readily available, and the input image can be stored entirely in memory.

Inverse mappings are more commonly used to perform spatial transformations. By operating in scanline order at the output, square output pixels are back-projected onto arbitrary quadrilaterals in the input. The quadrilaterals, also known as *preimages*, may cover many input pixels. Each preimage must be sampled and convolved with a low-pass filter to compute an intensity at the output. In general, the input image is not accessed in scanline order. Efficient approaches to sampling and convolution have received much attention in the recent literature [3, 4, 5, 6, 11].

2.1.3. Separable Mapping

Separable mapping decomposes the forward mapping function into a series of 1-D transforms. This offers several advantages. First, the resampling problem is made simpler since reconstruction, area sampling, and filtering can now be done entirely in 1-D. Second, efficient data access and substantial savings in I/O time can be realized because the input image can be read in row/column order and the output image produced in scanline order. Third, the approach is amenable to stream-processing techniques such as pipelining, and facilitates the design of hardware to operate at real-time video rates.

It is important to elaborate on our use of the term separable. In signal processing literature, a filter T is said to be separable if $T(u,v) = F(u)\,G(v)$. We extend this definition by defining T to be separable if $T(u,v) = F(u) \circ G(v)$. This simply replaces multiplication with the composition operator in combining both 1-D functions. The definition we offer for separablity in this paper is consistent with standard implementation practices. For instance, the 2-D Fourier Transform, separable in the classic sense, is generally implemented by a 2-pass algorithm. The first pass applies a 1-D Fourier Transform to each row, and the second applies a 1-D Fourier Transform along each column of the intermediate result.

Multi-pass scanline algorithms that operate in this sequential row-column manner are referred to as separable in this paper. The underlying theme is that processing is decomposed into a series of 1-D stages that each operate along orthogonal axes. For example, image rotation has been shown to be decomposable into a 2-pass scale/shear succession [1], a 4-pass scale/shear sequence [10], and a 3-pass shear transformation [7, 9].

2.2. CATMULL-SMITH ALGORITHM

The most general presentation of the 2-pass technique appears in the seminal work described by Catmull and Smith in [1]. That paper tackles the problem of mapping a 2-D image onto a 3-D surface and then projecting the result onto the 2-D screen for viewing. The contribution of that work lies in the decomposition of these steps into a sequence of computationally cheaper mapping operations. In particular, it is shown that in some cases a 2-D resampling problem can be replaced with two orthogonal 1-D resampling stages.

2.2.1. First Pass

In the first pass, each horizontal scanline (row) is resampled according to spatial transformation $F(u)$, generating an intermediate image I in scanline order. All pixels in I have the same x-coordinates that they will assume in the final output; only their y-coordinates now remain to be computed. Since each scanline will generally have a different transformation, function $F(u)$ will usually differ from row to row. Consequently, F can be considered to be a function of both u and v. Obviously, $F(u,v)$ is identical to $X(u,v)$. We rewrite $F(u,v)$ as $F_v(u)$ to denote that F is applied to horizontal scanlines, each having constant v. Therefore, the first pass is expressed as

$$[x, v] = [F_v(u), v] = [X(u,v), v]. \tag{2.2}$$

This relation maps each $[u,v]$ point onto I, an intermediate image in the $[x,v]$ plane.

2.2.2. Second Pass

In the second pass, each vertical scanline (column) in I is resampled according to spatial transformation $G(v)$, generating the final image in scanline order. The second pass is more complicated than the first pass because the expression for G is often difficult to derive. This is due to the fact that we must invert $[x,v]$ to get $[u,v]$ so that G can directly access $Y(u,v)$. In doing so, new y-coordinates can be computed for each point in I.

Inverting f requires us to solve the equation $X(u,v) - \tilde{x} = 0$ for u to obtain $u = H_x(v)$ for vertical scanline (column) $\tilde{x}$. Note that $\tilde{x}$ contains all the pixels along the column at x. Function H, known as the *auxiliary function*, represents the u-coordinates of the inverse projection of $\tilde{x}$, the column we wish to resample. Thus, for every column in I, we compute $H_x(v)$ and use it together with the available v-coordinates to index into mapping function Y. This specifies the vertical spatial transformation necessary for resampling the column. The second pass is therefore expressed as

$$[x, y] = [x, G_x(v)] \tag{2.3}$$

where $G_x(v)$ refers to the evaluation of $G(x,v)$ along vertical scanlines with constant x. It is given by

$$G_x(v) = Y(H_x(v), v) \tag{2.4}$$

The relation in Eq. (2.3) maps all points in I from the $[x,v]$ plane onto the $[x,y]$ plane, defining the final image.

2.2.3. 2-Pass Algorithm

In summary, the 2-pass algorithm has three steps. They correspond directly to the evaluation of scanline functions F and G, as well as the auxiliary function H.

1) The horizontal scanline function is defined as $F_v(u) = X(u,v)$. Each row is resampled

according to this spatial transformation, yielding intermediate image I.

2) The auxiliary function $H_x(v)$ is derived for each vertical scanline $\tilde{x}$ in I. It is defined as the solution to $\tilde{x} = X(u,v)$ for u, if such a solution can be derived. Sometimes a closed form solution for H is not possible and numerical techniques such as the Newton-Raphson iteration method must be used. As we shall see later, computing H is the principal difficulty with the 2-pass algorithm.

3) Once $H_x(v)$ is determined, the second pass plugs it into the expression for $Y(u,v)$ to evaluate the target y-coordinates of all pixels in column x in image I. The vertical scanline function is defined as $G_x(v) = Y(H_x(v),v)$. Each column in I is resampled according to this spatial transformation, yielding the final image.

2.2.4. An Example: Rotation

The above procedure is demonstrated on the simple case of rotation. The rotation matrix is given as

$$[x, y] = [u, v] \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{2.5}$$

We want to transform every pixel in the original image in scanline order. If we scan a row by varying u and holding v constant, we immediately notice that the transformed points are not being generated in scanline order. This presents difficulties in antialiasing filtering and fails to achieve our goals of scanline input and output.

Alternatively, we may evaluate the scanline by holding v constant in the output as well, and only evaluating the new x values. This is given as

$$[x, v] = [u\cos\theta - v\sin\theta, v] \tag{2.6}$$

This results in a picture that is skewed and scaled along the horizontal scanlines.

The next step is to transform this intermediate result by holding x constant and computing y. However, the equation $y = u\sin\theta + v\cos\theta$ cannot be applied since the variable u is referenced instead of the available x. Therefore, it is first necessary to express u in terms of x. Recall that $x = u\cos\theta - v\sin\theta$, so

$$u = \frac{x + v\sin\theta}{\cos\theta} \tag{2.7}$$

Substituting this into $y = u\sin\theta + v\cos\theta$ yields

$$y = \frac{x\sin\theta + v}{\cos\theta} \tag{2.8}$$

The output picture is now generated by computing the y-coordinates of the pixels in the intermediate image, and resampling in vertical scanline order. This completes the 2-pass rotation.

The stages derived above are directly related to the general procedure described earlier. The three expressions for F, G, and H are explicitly listed below.

1) The first pass is defined by Eq. (2.6). In this case, $F_v(u) = u\cos\theta - v\sin\theta$.
2) The auxiliary function H is given in Eq. (2.7). It is the result of isolating u from the expression for x in mapping function $X(u,v)$. In this case, $H_x(v) = (x + v\sin\theta) / \cos\theta$.
3) The second pass then plugs $H_x(v)$ into the expression for $Y(u,v)$, yielding Eq. (2.8). In this case, $G_x(v) = (x\sin\theta + v) / \cos\theta$.

2.2.5. Bottleneck Problem

After completing the first pass, it is sometimes possible for the intermediate image to collapse into a narrow area. If this area is much less than that of the final image, then there is insufficient data left to accurately generate the final image. This phenomenon, referred to as the *bottleneck problem* in [1], is the result of a many-to-one mapping in the first pass followed by a one-to-many mapping in the second pass. An example is the rotation of an image by 90°. Each row will collapse onto a point, resulting in an intermediate image consisting of a diagonal line. Obviously, no inverse function can resolve the intensities for the second pass.

In both [1] and [8], it is suggested that a solution to this problem lies in considering all the possible orders in which a separable algorithm can be implemented. Four variations, collectively referred to as V, are possible to generate the intermediate image:

1) Transform u first.
2) Transform v first.
3) Transpose the input image and transform u first.
4) Transpose the input image and transform v first.

In each case, the area of the intermediate image can be calculated. They suggest implementing the transformation with the variation that yields the largest intermediate area. For instance, an 87° rotation is best implemented by first rotating the image by 90° via image transposition and then applying a −3° rotation using the 2-pass technique. They purport that the above heuristic has not been known to fail, however no proof of its correctness is given.

We now give three difficulties with their solution to the bottleneck problem. The most critical problem is that the area of the intermediate image is a global measure which may fail to highlight compression variations in local areas. Although the heuristic seems to be satisfactory for the transformations considered in their original paper, it is inadequate for arbitrary mappings — the mappings considered here. For example, consider warping an image into a circular region with each row becoming a radial line, i.e., $(x, y) \rightarrow (r, \theta)$. This demonstrates a simple example in which different areas of the output map are best computed from different variations of V; no single transform from V could correctly process the entire image.

The second problem is that the heuristic does not properly consider aliasing artifacts. In particular, maximizing the intermediate area may require excessive compression in the second pass. For example, in Fig. 8 of [8] variations (1) and (2) were used to map a regular grid onto a sphere. Although variation (2) maximized the area of the intermediate image, it actually caused

more severe aliasing. This non-intuitive result is due to error properties of cascading 1-D filters to approximate a 2-D filter (see section 3.5.)

The third difficulty arises when the closed-form approximation to the intermediate area does not exist. While this does not prove troublesome in simpler domains, the evaluation of the intermediate areas for complex spatial mappings requires as much or more work as computing the first passes for each variation in V.

2.2.6. Foldover Problem

The 2-pass algorithm is particularly well-suited for mapping images onto surfaces with closed-form solutions to auxiliary function H. The process is more complicated for surfaces of higher order, e.g., bilinear, biquadratic, and bicubic patches, which may be self-occluding. This makes F or G become multi-valued at points where the image folds upon itself, a problem known as *foldover*.

Foldover can occur in either of the two passes. A partial solution for *single* folds in G is to compute output pixels in back-to-front order, overwriting the hidden layers. Generating the image in this manner becomes more complicated for surfaces with more than one fold. In the general case, this becomes a hidden surface problem.

The only reasonable solution has been to severely limit the amount of foldover, (e.g., up to three folds for cubic surfaces) and to use additional framebuffers to store the folded layers. This solution is inefficient inasmuch as an entire framebuffer must be utilized even if only a small folded area is involved.

2.2.7. Computing Auxiliary Function H

Closed-form solutions do not exist for H unless the patch has no folds. When folds occur, a solution $u = H_x(0)$ is found for the first horizontal scanline. Since surface patches are assumed to be smooth, a Newton-Raphson iteration method can be used to solve for $H_x(1)$ using the solution from $H_x(0)$ as a starting value. This exploits the spatial coherence of surface elements to solve the inverse problem at hand.

The complexity of solving for H can be reduced at the expense of additional memory. The need to evaluate H can be avoided altogether if we make use of earlier computations. In particular, we have H use the u-coordinates associated with the preimage of a pixel in the intermediate image. Thus, by introducing an auxiliary framebuffer to store these u's while we are in the first pass, H becomes available by trivial lookup table access.

2.3. DISCUSSION

The 2-pass algorithm has been shown to apply to a wide class of transformations of general interest. These mappings include the perspective projection of rectangles, bivariate patches, and superquadrics. Smith has discussed them in detail in [8]. In that paper, he emphasizes the mathematical consequence of decomposing mapping functions X and Y into a sequence of F

followed by G. Smith distinguishes X and Y as the *parallel warp*, and F and G as the *serial warp*, where *warp* refers to resampling. Serial warps offer dramatic computational reductions with only minor degradation. Nevertheless, they are plagued by several complications: the bottleneck problem, foldovers, and computing auxiliary function H.

3. DESCRIPTION OF ALGORITHM

In this section we describe an algorithm that addresses the difficulties that are particular to 2-pass methods. The result is a separable approach that is general, accurate, and efficient, with graceful degradation for transformations of arbitrary complexity.

3.1. OVERVIEW

The goal of this work is to realize an arbitrary warp with a separable algorithm. The proposed technique is an extension of the Catmull-Smith approach where attention has been directed toward solutions to the bottleneck and foldover problems, as well as removing the need for closed-form inverses. Consequently, the advantages of 1-D resampling are more fully exploited.

Conceptually, the algorithm consists of four stages: intensity resampling, coordinate resampling, distortion measurement, and compositing. Figure 3.1 shows the interaction of these components. Note that bold arrows represent the flow of images through a stage, and thin arrows denote those images which act upon the input. The subscripts x and y are appended to images which have been resampled in the horizontal and vertical directions, respectively.

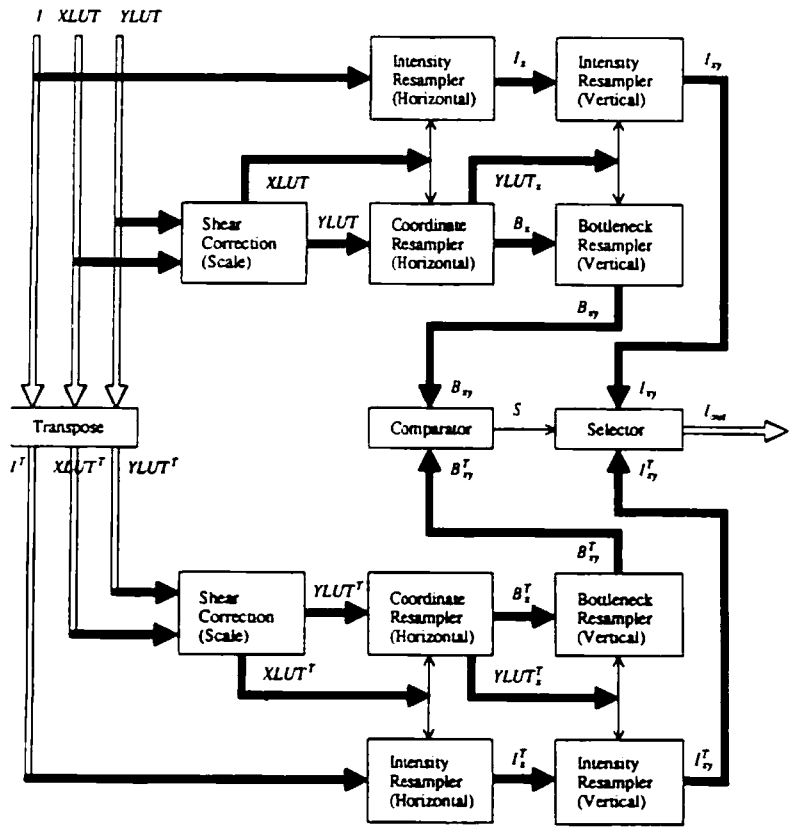


Figure 3.1: Block diagram of the algorithm.

The intensity resampler applies a 2-pass algorithm to the input image. Since the result may suffer bottleneck problems, the identical process is repeated with the transpose of the image. This accounts for the vertical symmetry of Fig. 3.1. Pixels which suffer excessive bottlenecking in the natural processing can be recovered in the transposed processing. In our implementation, we realize transposition as a 90° clockwise rotation so as to avoid the need to reorder pixels left to right.

The coordinate resampler computes spatial information necessary for the intensity resampler. It warps the spatial lookup table $Y(u,v)$ so that the second pass of the intensity resampler can access it without the need for an inverse function.

Local measures of shearing, perspective distortion, and bottlenecking are computed to indicate the amount of information lost at each point. This information, together with the transposed and non-transposed results of the intensity resampler, are passed to the compositor. The final output image is generated by the compositor which samples those pixels from the two resampled images such that information loss is minimized.

3.2. SPATIAL LOOKUP TABLES

Scanline algorithms generally express the coordinate transformation in terms of forward mapping functions X and Y. Sampling X and Y over all input points yields two new real-valued images, *XLUT* and *YLUT*, specifying the point-to-point mapping from each pixel in the input image onto the output images. This paper refers to *XLUT* and *YLUT* as *spatial lookup tables* since they can be viewed as 2-D tables which express a spatial transformation.

In addition to *XLUT* and *YLUT* we also provide a mechanism for the user to specify *ZLUT* which associates a z-coordinate value with each pixel. This allows warping of planar textures onto non-planar surfaces, and is useful in dealing with foldovers. Our goal, however, is not to solve the general 3-D viewing problem. The z-coordinates are assumed to be from a particular point of view which the user determines before supplying *ZLUT* to the system.

The motivation for introducing spatial lookup tables is generality. Our goal is to find a serial warp equivalent to any given parallel warp. Thus, we find it impossible to retain the mathematical elegance of closed-form expressions for the mapping functions F, G, and the auxiliary function, H. Therefore, assuming the forward mapping functions, X and Y, have closed-form expressions seems overly restrictive. Instead, we assume that the parallel warp is defined by the samples that comprise our spatial lookup tables. This provides a general means of specifying arbitrary mapping functions.

For each pixel (u,v) in input image I, spatial lookup tables *XLUT*, *YLUT*, and *ZLUT* are indexed at location (u,v) to determine the corresponding (x,y,z) position of the input point after warping. This new position is orthographically projected onto the output image. Therefore, (x,y) is taken to be the position in the output image. (Of course, a perspective projection may be included as part of the warp). The z-coordinate will only be used to resolve foldovers. This straightforward indexing applies only if the dimensions of I, *XLUT*, *YLUT*, and *ZLUT* are all

identical. If this is not the case, then the smaller images are upsampled (magnified) to match the largest dimensions.

3.3. INTENSITY RESAMPLING

This section discusses how spatial lookup tables are used to resample intensity images. The 1-D intensity resampling algorithm is well-suited for hardware implementation and compatible with spatial lookup tables. It is the basis of the 2-pass intensity resampling stage depicted in the first row of Fig. 3.1.

3.3.1. 1-D Intensity Resampler

The central benefit of separable algorithms is the reduction in complexity allowed by 1-D resampling algorithms. This provides efficient solutions for the image reconstruction and antialiasing components of resampling. Fant presents a detailed description of such a 1-D algorithm that is well-suited for hardware implementation and compatible with spatial lookup tables [4]. It is the principal 1-D resampling method used in separable transformations, including that of the work presented here.

The process treats the input and output as streams of pixels that are consumed and generated at rates determined by the spatial mapping. The input is assumed to be mapped onto the output along a single direction, i.e., with no folds. As each input pixel arrives, it is weighted by its partial contribution to the current output pixel and integrated into a single-element accumulator. For input pixels that spread out over many output pixels, image reconstruction is currently implemented with linear interpolation. In terms of the input and output streams, one of three conditions is possible:

1) The current input pixel is entirely consumed without completing an output pixel.
2) The input is entirely consumed while completing the output pixel.
3) The output pixel will be completed without entirely consuming the current input pixel. In this case, a new input value is interpolated from the neighboring input pixels at the position where the input was no longer consumed. It is used as the next element in the input stream.

If conditions (2) or (3) apply, the output computation is complete and the accumulator value is stored into the output array. The accumulator is then reset to zero in order to receive new input contributions for the next output pixel. Since the input is unidirectional, a one-element accumulator is sufficient. The process continues to cycle until the entire input stream is consumed.

3.3.2. Example

Consider the 1-D arrays shown in Fig. 3.2. The first row is taken from *XLUT*, the second from *YLUT*, and the third from input intensity image *I*. The next two arrays show *YLUT* and *I* resampled according to *XLUT*.

XLUT	.6	2.3	3.2	3.3	3.9
YLUT	100	106	115	120	
I	100	106	92	90	
$YLUT_x$	100	101	105	113	
I_x	40	101	106	82	

Figure 3.2: Resampling example.

The computation of the resampled intensity values is given below. For clarity, the following notation is used: interpolated input values are written within square brackets ([]), weights denoting contributions to output pixels are written within an extra level of parentheses, and input intensity values are printed in boldface.

$$I_x[0] = (\mathbf{100})((.4)) = 40$$

$$I_x[1] = \left[(\mathbf{100})\left[1-\frac{.4}{1.7}\right] + (\mathbf{106})\left[\frac{.4}{1.7}\right]\right]((1)) = 101$$

$$I_x[2] = \left[(\mathbf{100})\left[1-\frac{1.4}{1.7}\right] + (\mathbf{106})\left[\frac{1.4}{1.7}\right]\right]((.3)) + (\mathbf{106})((.7)) = 106$$

$$I_x[3] = \left[(\mathbf{106})\left[1-\frac{.7}{.9}\right] + (\mathbf{92})\left[\frac{.7}{.9}\right]\right]((.2)) + (\mathbf{92})((.1)) + (\mathbf{90})((.6)) = 82$$

The algorithm demonstrates both image reconstruction and antialiasing. When we are not positioned at pixel boundaries in the input stream, linear interpolation is used to reconstruct the discrete input. When more than one input element contributes to an output pixel, the weighted results are integrated in an accumulator to achieve antialiasing. These two cases are both represented in the above equations, as denoted by the expressions between square brackets and double parentheses, respectively.

3.3.3. 2-Pass Intensity Resampling

The 1-D intensity resampler is applied to the image in two passes, each along orthogonal directions. The first pass resamples horizontal scanlines, warping pixels along a row in the intermediate image. Its purpose is to deposit them into the proper columns for vertical resampling. At that point, the second pass is applied to all columns in the intermediate image, generating the

output image.

In Fig. 3.1, input image I is shown warped according to *XLUT* to generate intermediate image I_x. In order to apply the second pass, *YLUT* is warped alongside I, yielding $YLUT_x$. This resampled spatial lookup table is applied to I_x in the second pass as a collection of 1-D vertical warps. The result is output image I_{xy}.

The intensity resampling stage must handle multiple output values to be defined in case of foldovers. This is an important implementation detail which has impact on the memory requirements of the algorithm. We defer discussion of this aspect of the intensity resampler until section 3.6, where foldovers are discussed in more detail.

3.4. COORDINATE RESAMPLING

$YLUT_x$ is computed in the coordinate resampling stage depicted in the second row of the block diagram in Fig. 3.1. The ability to resample *YLUT* for use in the second pass has important consequences: it circumvents the need for a closed-form inverse of the first pass. As briefly pointed out in [1], that inverse provides exactly the same information that was available as the first pass was computed, i.e., the u-coordinate associated with a pixel in the intermediate image. Thus, instead of computing the inverse to index into *YLUT*, we simply warp *YLUT* into $YLUT_x$ allowing direct access in the second pass.

3.4.1. Coordinate Resampler

The coordinate resampler is similar to the intensity resampler. It differs only in the notable absence of antialiasing filtering — the output coordinate values in $YLUT_x$ are computed by *point* sampling *YLUT*. Interpolation is used to compute values when no input data is supplied at the resampling locations. However, unlike the intensity resampler, the coordinate resampler does not weigh the result with its area coverage nor does the resampler average it with the coordinate values of other contributions to that pixel. This serves to secure the accuracy of edge coordinates, even when the edge occupies only a partial output pixel.

3.4.2. Example

The following example is offered to demonstrate the coordinate resampling algorithm. Consider the arrays shown before in Fig. 3.2. $YLUT_x$ in the example is the output of the coordinate resampling as computed below. Notice that the output consists of point samples taken at pixel boundaries in the output stream. They are not influenced by any other entries deposited into their respective output pixels. The computations are given below.

$$YLUT_x[0] = (100)\left[1-\frac{0}{1.7}\right] + (106)\left[\frac{0}{1.7}\right] = 101$$

$$YLUT_x[1] = (100)\left[1-\frac{.4}{1.7}\right] + (106)\left[\frac{.4}{1.7}\right] = 101$$

$$YLUT_x[2] = (100)\left[1-\frac{1.4}{1.7}\right] + (106)\left[\frac{1.4}{1.7}\right] = 105$$

$$YLUT_x[3] = (106)\left[1-\frac{.7}{.9}\right] + (115)\left[\frac{.7}{.9}\right] = 113$$

As mentioned before, the user can define *ZLUT* which associates a *z*-coordinate with each pixel. While it does not appear in the system block diagram of Fig. 3.1, we also apply this resampling to *ZLUT* in exactly the same manner as it was applied to *YLUT*.

3.5. DISTORTIONS AND ERRORS

In forward mapping, input pixels are taken to be squares that map onto arbitrary quadrilaterals in the output image. Although separable mappings greatly simplify resampling by treating pixels as points along scanlines, the measurement of distortion must necessarily revert to 2-D to consider the deviation of each input pixel as it projects onto the output.

As is standard, we treat the mapping of a square onto a general quadrilateral as a combination of translation, scaling, shearing, rotation, and perspective transformations. Inasmuch as separable kernels exist for realizing translations and scale changes, these transformations do not suffer degradation in scanline algorithms and are not considered further. Shear, perspective and rotations, however, offer significant challenges to the 2-pass approach. In particular, excessive shear and perspective contribute to aliasing problems while rotations account for the bottleneck problem.

We first examine the errors introduced by separable filtering. We then address the three sources of geometric distortion for 2-pass scanline algorithms: shear, perspective, and rotation.

3.5.1. Filtering Errors

One of the sources of error for scanline algorithms comes from the use of cascaded orthogonal 1-D filtering. Let us ignore rotation for a moment, and assume we process the image left-to-right and top-to-bottom. Then one can easily show that scanline algorithms will, in the first pass, filter a pixel based only on the horizontal coverage of its top segment. In the second pass, they will filter based only on the vertical coverage of the left-hand segment of the input pixel. As a result, a warped pixel generating a triangular section in an output pixel is always approximated by a rectangle (Fig. 3.3). Note this can be either an overestimate or underestimate, and the error depends on the direction of processing. This problem is not unique to our approach. It

is shared by all scanline algorithms known to us.

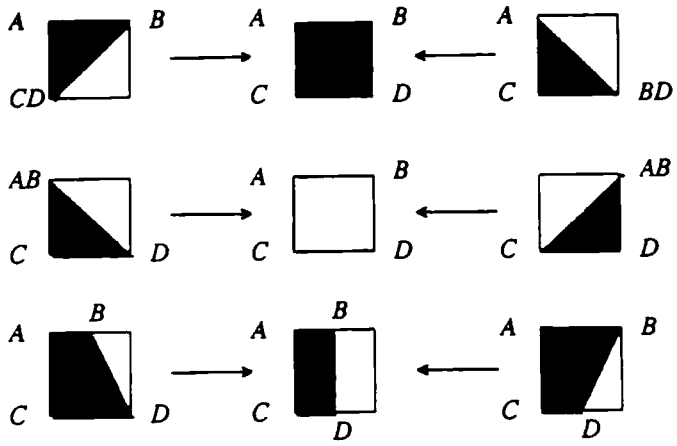


Figure 3.3: Examples of filtering errors.

3.5.2. Shear

Figure 3.4 depicts a set of spatial lookup tables which demonstrate horizontal shear. For simplicity, the example includes no scaling or rotation. The figure also shows the result obtained after applying the tables to an image of constant intensity (100). The horizontal shear is apparent in the form of jagged edges between adjacent rows.

0	1	2	3
2	3	4	5
4	5	6	7

XLUT

0	0	0	0
1	1	1	1
2	2	2	2

YLUT

100	100	100				
		100	100	100		
				100	100	100

Aliased Output Image

Figure 3.4: Horizontal shear: Spatial LUTs and output image.

Scanline algorithms are particularly sensitive to this form of distortion because proper filtering is applied only *along* scanlines — filtering issues *across* scanlines are not considered. Consequently, horizontal (vertical) shear is a manifestation of aliasing along the vertical (horizontal) direction, i.e., between horizontal (vertical) scanlines. The prefiltering stage described below must be introduced to suppress these artifacts *before* the regular 2-pass algorithm is applied.

This problem is a symptom of undersampled spatial lookup tables, and the only real solution lies in increasing the resolution of the tables by sampling the continuous mapping functions more densely. If the continuous mapping functions are no longer available to us, then new values are computed from the sparse samples by interpolation. (In this paper, linear interpolation is assumed to be adequate.)

We now consider the effect of increasing the spatial resolution of *XLUT* and *YLUT*. The resulting image in Fig. 3.5 is shown to be antialiased, and clearly superior to its counterpart in Fig. 3.4. The values of 37 and 87 reflect the partial coverage of the input slivers at the output. Note that with additional upsampling, these values converge to 25 and 75, respectively. Adjacent rows are now constrained to lie within 1/2 pixel of each other.

The error constraint can be specified by the user and the spatial resolution for the lookup tables can be determined automatically. This offers us a convenient mechanism in which to control error tolerance and address the space/accuracy tradeoff. For the examples herein, both horizontal and vertical shear are restricted to one pixel.

37	87	100	87	37				
		37	87	100	87	37		
				37	87	100	87	37

Figure 3.5: Corrected output image.

By now the reader may be wondering if the shear problems might be alleviated, as was suggested in [1], by considering a different order of processing. While the problem may be slightly ameliorated by changing processing direction, the real problem lies in undersampling the lookup tables. They are specifying an output configuration (with many long thin slivers) which, because of filtering errors, cannot be accurately realized by separable processing in any order.

3.5.3. Perspective

Like shear, perspective distortions may also cause problems by warping a rectangle into a triangular patch which results in significant filtering errors. In fact, if one only considers the warp determined by any three corners of an input pixel, one cannot distinguish shear from perspective projection. The latter requires knowledge of all four corners. The problem generated by perspective warping can also be solved by the same mechanism as for shears: resample the spatial lookup tables to ensure that no long thin slivers are generated. However, unlike shear, perspective also effects the bottleneck problem because, for some orders of processing, the first pass may be contractive while the second pass is expansive. This perspective bottlenecking is handled by the same mechanism as for rotations, as described below.

3.5.4. Rotation

In addition to jagginess due to shear and perspective, distortions are also introduced by rotation. Rotational components in the spatial transformation are the *major* source of bottleneck problems. Although all rotation angles contribute to this problem, we consider those beyond 45° to be inadequately resampled by a 2-pass algorithm. This threshold is chosen because 0° and 90° rotations can be performed exactly. If other exact image rotations were available, then the worst case error could be reduced to half the maximum separation of the angles. Local areas whose rotational components exceed 45° are recovered from the transposed results, where they obviously undergo a rotation less than 45°.

3.5.5. Distortion Measures

Consider scanning two scanlines jointly, labeling an adjacent pair of pixels in the first row as A, B, and the associated pair in the second row as C, and D. Let (x_A,y_A), (x_B,y_B), (x_C,y_C), and (x_D,y_D) be their respective output coordinates as specified by the spatial lookup tables. These points define an output quadrilateral onto which the square input pixel is mapped. From these four points it is possible to determine the horizontal and vertical scale factors necessary to combat aliasing due to shear and perspective distortions, and also determine if extensive bottlenecking is present. For convenience, we define

$$\Delta x_{ij} = |x_i - x_j|; \quad \Delta y_{ij} = |y_i - y_j|; \quad s_{ij} = \Delta y_{ij} / \Delta x_{ij}.$$

Pseudo-code for the procedure is given below.

```
bottleneck = 0                          /* Initially no bottleneck */
IF( Δy_AB ≤ Δx_AB ) {                   /* AB remains horizontal */
      vfctr = max ( Δx_AC, Δx_BD )      /* measure horizontal shear */
} ELSE IF( s_AB ≤ s_AC ) {              /* AC remains vertical */
      hfctr = max ( Δy_AB, Δy_CD )      /* measure vertical shear */
} ELSE bottleneck = 1                   /* bottleneck occurs */
```

If AB has not rotated from the horizontal by more than 45°, then its error due to bottlenecking is considered acceptable, and we say that it remains "horizontal." Only the vertical aliasing distortions due to horizontal shearing and/or perspective need to be considered in this case. The vertical scale factor, *vfctr*, for *XLUT* and *YLUT* is given by $vfctr = \text{MAX}(\Delta x_{AC}, \Delta x_{BD})$. Briefly, this measures the maximum deviation in the horizontal direction for a unit step in the vertical direction. To ensure an alignment error of at most ε, the image must be rescaled vertically by a factor of $vfctr/\varepsilon$. Examples of quadrilaterals that satisfy this case are illustrated in Fig. 3.6.

If AB is rotated by more than 45°, then we say that it has become "vertical" and two possibilities exist: vertical shearing/perspective or rotation. In order to consider vertical shear/perspective, the magnitude of the slope of AC is measured in relation to that of AB. If $s_{AB} \le s_{AC}$, then AC is considered to remain vertical and the pixel is tested for vertical

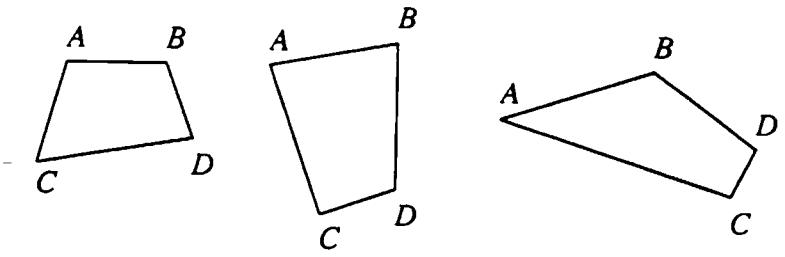


Figure 3.6: Warps where *AB* remains horizontal.

shear/perspective. (In order to enhance computational efficiency and to avoid divide-by-zero errors, the test condition is actually posed in terms of multiplication only.) If the test condition is satisfied, the horizontal scale factor, *hfctr*, for the spatial lookup tables is expressed as $hfctr = \mathrm{MAX}(\Delta y_{AB}, \Delta y_{CD})$. Briefly stated, this measures the the maximum deviation in the vertical direction for a unit step in the horizontal direction. Again, alignment error can be limited to ε by rescaling the image horizontally by a factor of $hfctr/\varepsilon$. Examples of this condition are shown in Fig. 3.7.

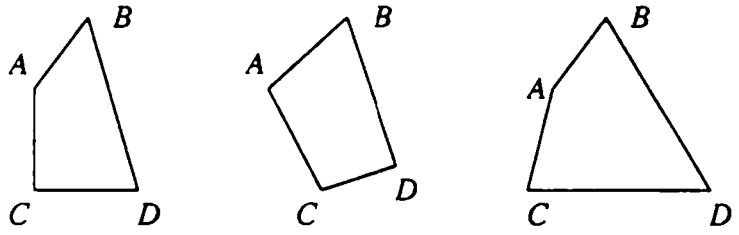


Figure 3.7: *AB* has rotated while *AC* remains vertical. Vertical shear.

If, however, angle *BAC* is also found to be rotated, (i.e., neither of the above tests are satisfied) then the entire quadrilateral *ABCD* is considered to be bottlenecked because it has rotated and/or undergone a perspective distortion. The *bottleneck* flag is set to one to denote the presence of the bottleneck problem at this pixel and, as described below, its contributions will be taken from the transposed result. This case is depicted in Fig. 3.8.

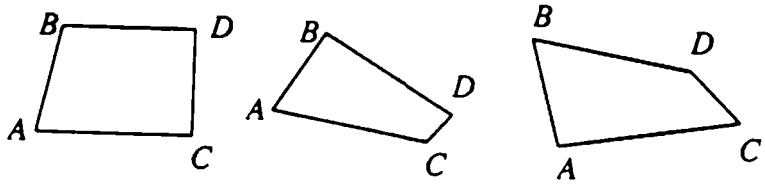


Figure 3.8: Both *AB* and *AC* have rotated. Bottleneck problem.

The code fragment listed above is applied to each input pixel. Currently, the maximum values of $hfctr/\varepsilon$ and $vfctr/\varepsilon$ are used to scale the spatial lookup tables before they enter the 2-pass resampling stage. In this manner, the output of this stage is guaranteed to be free of aliasing due to undersampled spatial lookup tables. In the future, we will examine using this as a local measure to adaptively resample the tables.

3.5.6. Bottleneck Distortion

The bottleneck problem was described earlier as a many-to-one mapping followed by a one-to-many mapping. The extent to which the bottleneck problem becomes manifest is intimately related to the order in which the orthogonal 1-D transformations are applied. The four possible orders in which a 2-D separable transformation can be implemented are listed in section 2.2.3. Of the four alternatives, we shall only consider variations (1) and (3). Although variations (2) and (4) may have impact on the extent of aliasing in the output image (see Fig. 8 of [8]), their roles may be obviated by upsampling the spatial lookup tables before they enter the 2-pass resampling stage.

A solution to the bottleneck problem thereby requires us to consider the effects which occur as an image is separably resampled with and without a preliminary image transposition stage. Unlike the Catmull-Smith algorithm which selects only one variation for the entire image, we are operating in a more general domain which may require either of the two variations over arbitrary regions of the image. This leads us to develop a local measure of bottleneck distortion which is used to determine which variation is most suitable at each output pixel. Thus alongside each resampled intensity image, another image of identical dimensions is computed to maintain estimates of the local bottleneck distortion.

A 2-pass method is introduced to compute bottleneck distortion estimates at each point. As above, the *bottleneck* flag is determined for each input pixel. If no bottlenecking is present, then the area coverage of that input pixel is integrated into bottleneck image B_x. If, however, the *bottleneck* flag is set to one, then that pixel makes no contribution to B_x. The bottleneck image thus reflects the fraction of each pixel in the intermediate image *not* subject to bottleneck distortion in the first pass. The computations are simple, and serve a secondary function in that the entries correspond exactly to the weights needed for antialiasing in the intensity resampling stage. Thus we are getting a local distortion measure at virtually no additional cost.

The second pass resamples intermediate image B_x in the same manner as the intensity resampler, thus spreading the distortion estimates to their correct location in the final image. The result is a double-precision bottleneck-distortion image B_{xy}, with values inversely proportional to the bottleneck artifacts. The distortion computation process is repeated for the transpose of the image and spatial lookup tables, generating image B_{xy}^T.

Since the range of values in the bottleneck image are known to lie between 0 and 1, it is possible to quantize the range into N intervals for storage in a lower precision image with $\log_2 N$ bits per pixel. This space/accuracy tradeoff will be assessed in future work. We point out that the measure of area is not exact. It is subject to exactly the same errors as intensity filtering.

3.6. FOLDOVER PROBLEM

Up to this point, we have been discussing our warping algorithm as though both passes resulted in only a single value for each point. Unfortunately, this is often not the case — a warped scanline can fold back upon itself.

In [1] it was proposed that multiple framebuffers be used to store each level of the fold. While this solution may be viable for low-order warps, as considered in [1] and [8], it may prove to be too costly for arbitrary warps where the number of potential folds may be large. Furthermore, it is often the case that the folded area may represent a small fraction of the output image. Thus, using one frame buffer per fold would be prohibitively expensive, and we seek a solution which degrades more gracefully.

If we are to allow an image to fold upon itself, we must have some means of determining which of the folds are to be displayed. The simplest mechanism, and probably the most useful, is to assume that the user will supply not only *XLUT* and *YLUT*, but also *ZLUT* to specify the output z-coordinates for each input pixel. In the first pass *ZLUT* will be processed in exactly the same way as *YLUT*, so the second pass of the intensity resampler can have access to the z-coordinates.

Given *ZLUT* we are now faced with the problem of keeping track of the information from the folding. A naive solution might be to use a z-buffer in computing the intermediate and final images. Unfortunately, while z-buffering will work for the output of the second pass, it cannot work for the first pass because some mappings fold upon themselves in the first pass only to have some of the "hidden" part exposed by the second pass of the warp. Thus, we must find an efficient means of incorporating all the data, including the foldovers, in the intermediate image.

3.6.1. Representing Foldovers

Our solution is to maintain multiple columns for each column in the intermediate image. The extra columns, or layers, of space are allocated to hold information from foldovers on an as-needed basis. The advantage of this approach is that if a small area of the image undergoes folding, only a small amount of extra information is required. When the warp has folds, the intermediate image has a multi-layered structure, like that in Fig. 3.9.

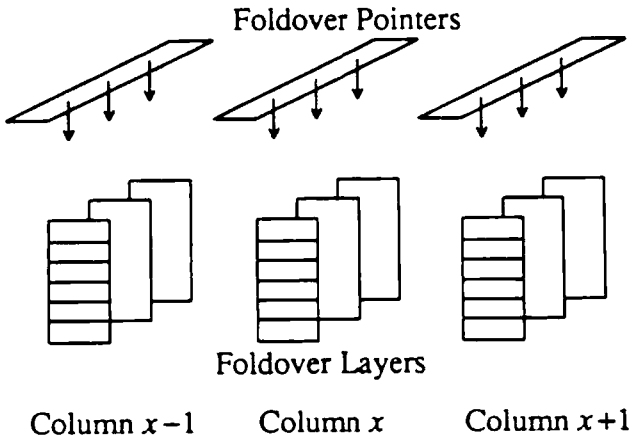


Figure 3.9: Data structure for folded warps.

While this representation is superior to multiple frame buffers, it may still be inefficient unless we allow each layer in the intermediate image to store data from many different folds (assuming that some of them have terminated and new ones were created). Thus, we reuse each

foldover layer whenever possible

In addition to the actual data stored in extra layers, we also maintain a number of extra pieces of information (described below), such as various pointers to the layers, and auxiliary information about the last entry in each layer.

3.6.2. Tracking Foldovers

It is not sufficient to simply store all the necessary information in some structure for later processing. Given that folds do occur, there is the problem of how to filter the intermediate image. Since filtering requires all the information from one foldover layer to be accessed coherently, it is necessary to track each layer across many rows of the image. For efficiency, we desire to do this tracking using a purely local match from one row to the next. The real difficulty in the matching is when fold layers are created, terminated, or bifurcated. We note that any ''matching'' must be a heuristic, since without strong assumptions about the warps, there is *no* procedure to match folds from one row to another. (The approach in [1] assumes that the Newton-Raphson algorithm can follow the zeros of the auxiliary function H correctly, which is true only for simple auxiliary functions with limited bifurcations.)

Our heuristic solution to the matching problem uses three types of information: direction of travel when processing the layer (left or right in the row), ordering of folds within a column, and the original u-coordinate associated with each pixel in the intermediate image.

First, we constrain layers to match only those layers where the points are processed in the same order. For instance, matching between two leftward layers is allowed, but matching between leftward and rightward layers is not allowed.

Secondly, we assume the layers within a single column are partially ordered. Within each column, every folded pixel in the current row is assigned a unique number based on the order in which it was added to the foldover lists. The partial order would allow matching pixels 12345 with 1?23??4 (where the symbol ? indicates a match with a null element), but would not allow matching of 12345 with 1?43??2.

Finally, we use the u-coordinate associated with each pixel to define a distance measure between points which satisfies the above constraints. The match is done using a divide-and-conquer technique. Briefly, we first find the best match among all points, i.e., minimum distance. We then subdivide the remaining potential matches to the left and to the right of the best match, thus yielding two smaller subsets on which we reapply the algorithm. For hardware implementation, dynamic programming may be more suitable. This is a common solution for related string matching problems.

Consider a column which previously had foldover layers labeled 123456, with orientation *RLRLRL*, and original u-coordinates of 10,17,25,30,80,95. If two of these layers now disappeared leaving four layers, say *abcd*, with orientation *RLRL* and original u-coordinates of 16,20,78,101, then we would do the matching finding *abcd* matching 1256 respectively.

3.6.3. Storing Information from Foldovers

Once the matches are determined, we must rearrange the data so that the intensity resampler can access it in a spatially coherent manner. To facilitate this, each column in the intermediate image has a block of pointers that specify the order of the foldover layers. When the matching algorithm results in a shift in order, a different set of pointers is defined, and the valid range of the previous set is recorded. The advantage of this explicit reordering of pointers is that it allows for efficient access to the folds while processing.

We describe the process from the point of view of a single column in the intermediate image, and note that all columns are processed identically. The first encountered entry for a row goes into the base layer. For each new entry into this column, the fill pointer is advanced (using the block of pointers), and the entry is added at the bottom of the next fold layer. After we compute the "best" match we move incorrectly stored data, reorder the layers and define a new block of pointers.

Let us continue the example from the end of the last section, where 123456 was matched to 1256. After the matching, we would then move the data, incorrectly stored in columns 3 and 4 into the appropriate location in 5 and 6. Finally we would reorder the columns and adjust the pointer blocks to reflect the new order 125634. The columns previously labeled 34 would be marked as terminated, and considered spares to be used in later rows if a new fold layer begins.

3.6.4. Intensity Resampling with Foldovers

A final aspect of the foldover problem is how it affects the 2-D intensity resampling process. The discussion above demonstrates that all the intensity values for a given column are collected in such a way that each fold layer is a separate contiguous array of spatially coherent values. Thus, the contribution of each pixel in a fold layer is obtained by standard 1-D filtering of that array.

From the coordinate resampler, we obtain $ZLUT_{xy}$, and thus, merging the foldovers is equivalent to determining which filtered pixels are visible. Given the above information, we implement a simple z-buffer algorithm, which integrates the points in front-to-back order with partial coverage calculations for antialiasing. When the accumulated area coverage exceeds 1, the integration terminates. Note that this z-buffer requires *only* a 1-D accumulator, which can be reused for each column. The result is a single intensity image combining the information from all visible folds.

3.7. COMPOSITOR

The compositor generates the final output image by selecting the most suitable pixels from I_{xy} and I^T_{xy} as determined by the bottleneck images B_{xy} and B^T_{xy}. A block diagram of the compositor is shown in center row of Fig. 3.1.

Bottleneck images B_{xy} and B^T_{xy} are passed through a comparator to generate bitmap image S. Also known as a *vector mask*, S is initialized according to the following rule.

$$S[x,y] = (B_{xy}[x,y] \leq B^{T}_{xy}[x,y])$$

Images S, I_{xy}, and I^{T}_{xy} are sent to the selector where I_{out} is assembled. For each position in I_{out}, the vector mask S is indexed to determine whether the pixel value should be sampled from I_{xy} or I^{T}_{xy}.

4. EXAMPLES

This section illustrates some examples of the algorithm. Figure 4.1 shows two images that will be used as source images for numerous examples. We refer to these images as the checkerboard and as Madonna †.

Figure 4.2 shows the final result of warping the checkerboard and Madonna into a 360° circle. This transformation takes each row of the source image and maps it into a radial line. This corresponds directly to a mapping from the Cartesian coordinate system to the polar coordinate system, i.e., $(x, y) \rightarrow (r, \theta)$.

Figure 4.3 illustrates the output of the intensity resampler for the non-transposed and transposed processing. I_{xy} appears in Fig. 4.3a, and I^T_{xy} is shown in Fig. 4.3b. Figure 4.3c shows S, the vector mask image. S selects points from I_{xy} (white) and I^T_{xy} (black) to generate the final output image I_{out}. Gray points in S denote equal bottleneck computations from both sources. Ties are arbitrarily resolved in favor of I^T_{xy}. Finally, in Fig. 4.3d, the two spatial lookup tables *XLUT* and *YLUT* that defined the circular warp, are displayed as intensity images, with y increasing top-to-bottom, and x increasing left-to-right. Bright intensity values in the images of *XLUT* and *YLUT* denote high coordinate values. Note that if the input were to remain undistorted *XLUT* and *YLUT* would be ramps. The deviation from the ramp configuration depicts the amount of deformation which the input image undergoes.

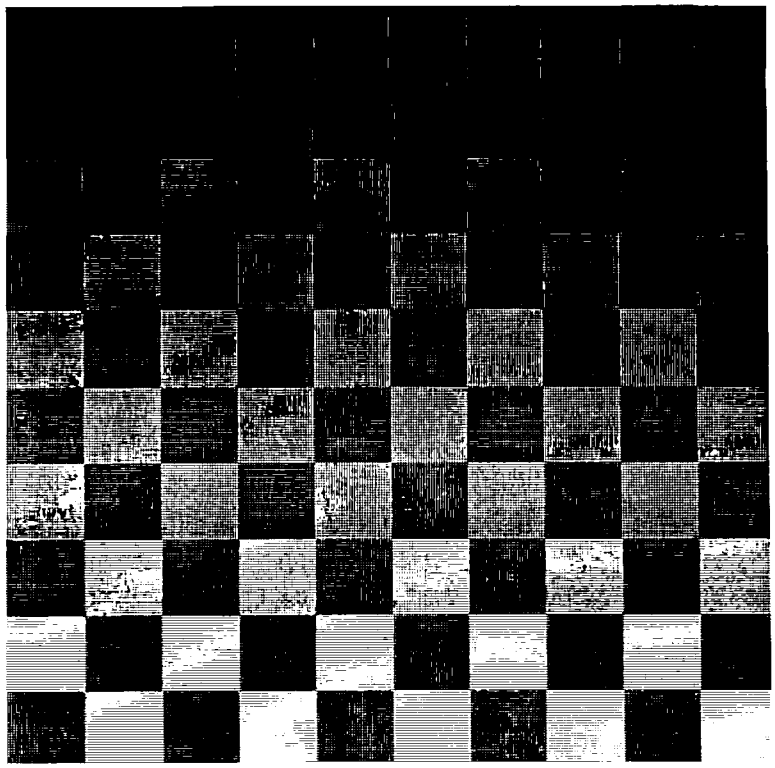
Figure 4.4 demonstrates the effect of undersampling the spatial lookup tables. The checkerboard is again warped into a circle. However, *XLUT* and *YLUT* were supplied at lower resolution. The jagginess in the results are now more pronounced.

Figure 4.5a illustrates an example of foldovers. Figure 4.5b shows *XLUT* and *YLUT*. A foldover occurs because *XLUT* is not monotonically increasing from left to right. In Figs. 4.6a and 4.6b, the foldover regions are shown magnified (with pixel replication) to highlight the results of two different methods of rendering the final image. In Fig. 4.6a, we simply selected the closest pixels. Note that dim pixels appear at the edge of the fold as it crosses the image. This subtlety is more apparent along the fold upon the cheek. The intensity drop is due to the antialiasing filtering that correctly weighted the pixels with their area coverage along the edge. This can be resolved by integrating partially visible pixels in front-to-back order. As soon as the sum of area coverage exceeds 1, no more integration is necessary. Fortunately, the bottleneck image can be used to directly supply the area coverage data. The improved result appears in Fig. 4.6b.

Figure 4.7 shows the result of bending horizontal rows. As we scan across the rows in left-to-right order, the row becomes increasingly vertical. This is another example in which the traditional 2-pass method would clearly fail since a wide range of rotation angles are represented. A vortex warp is shown in Fig. 4.8.

---

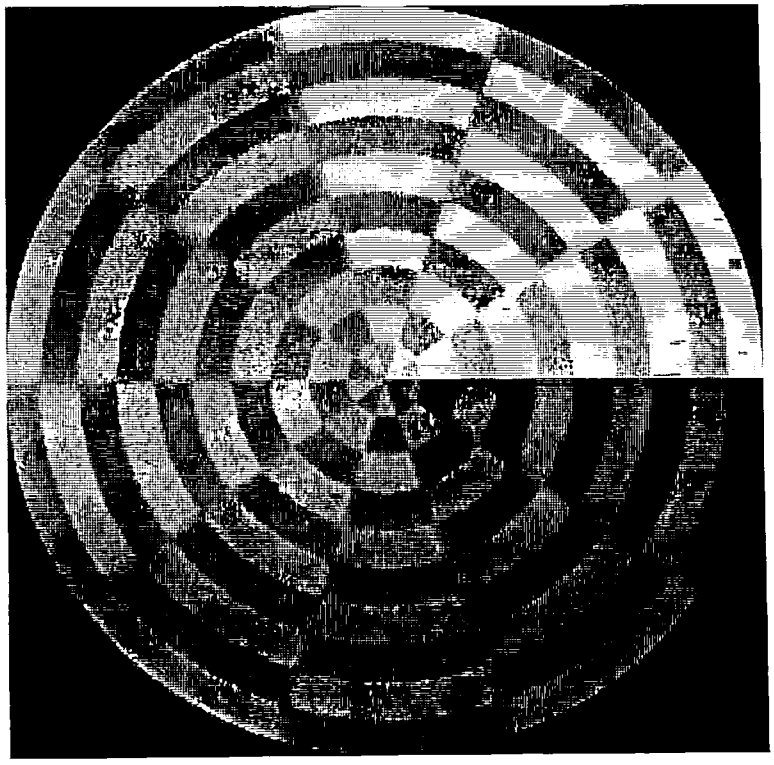
† Reprinted with permission from Warner Bros. Records.



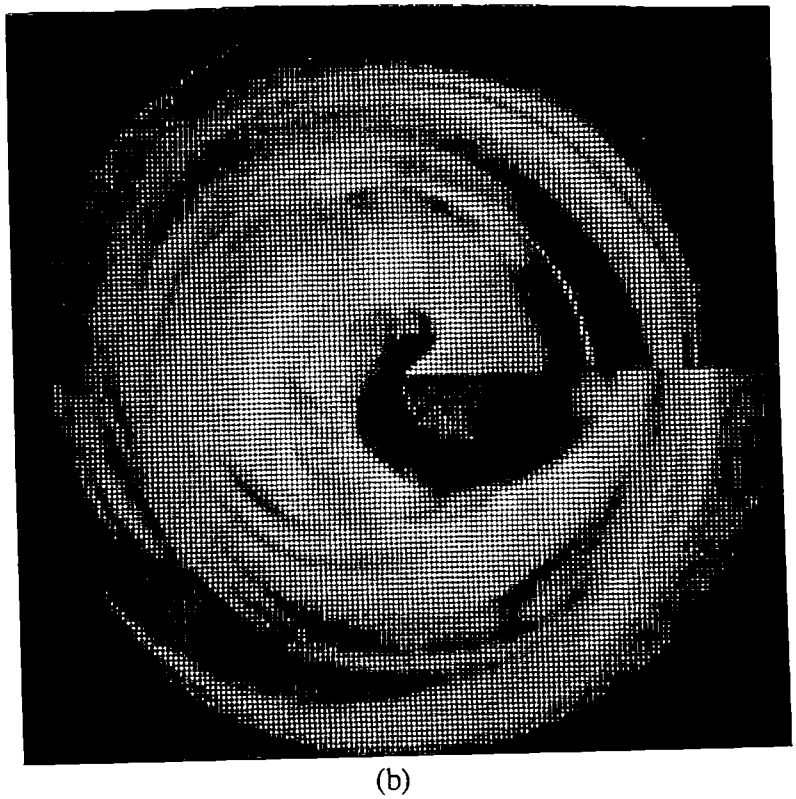
(a)

(b)

Figure 4.1: Input images: (a) Checkerboard and (b) Madonna.

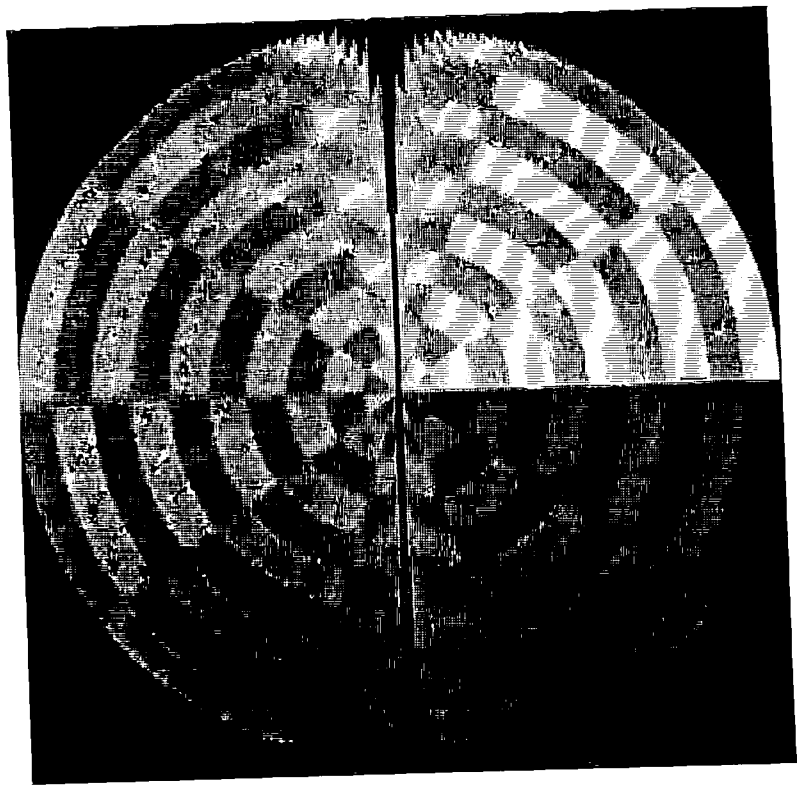


(a)

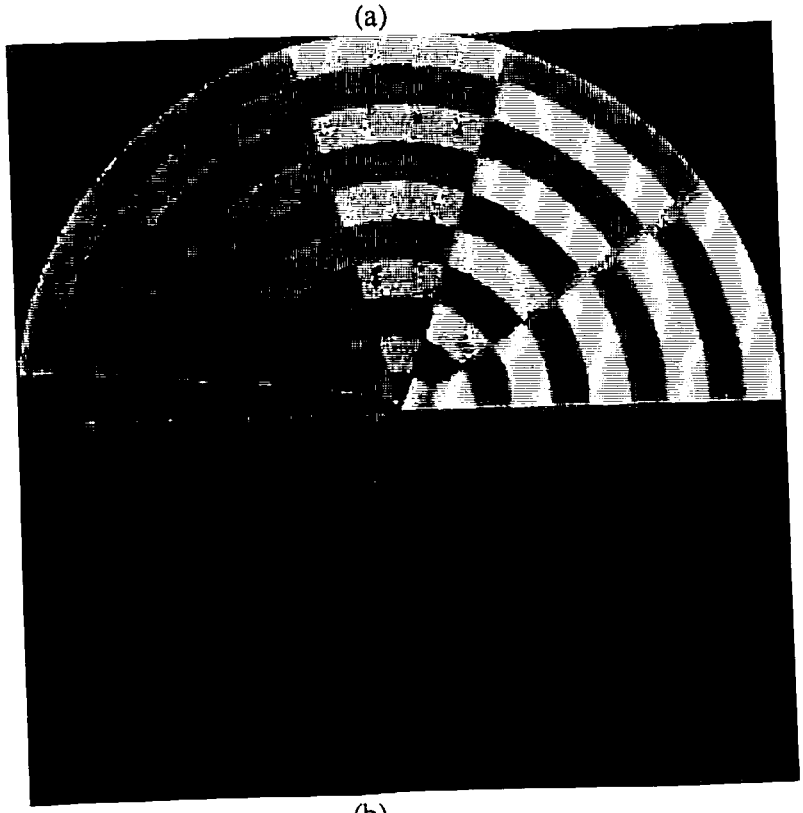


(b)

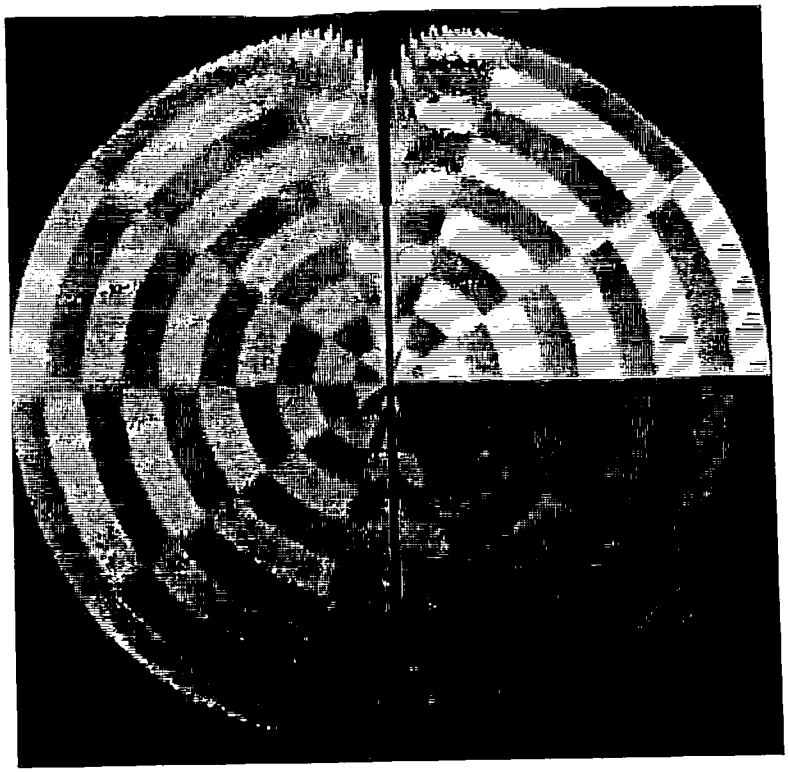
Figure 4.2: 360° warps on (a) Checkerboard and (b) Madonna.



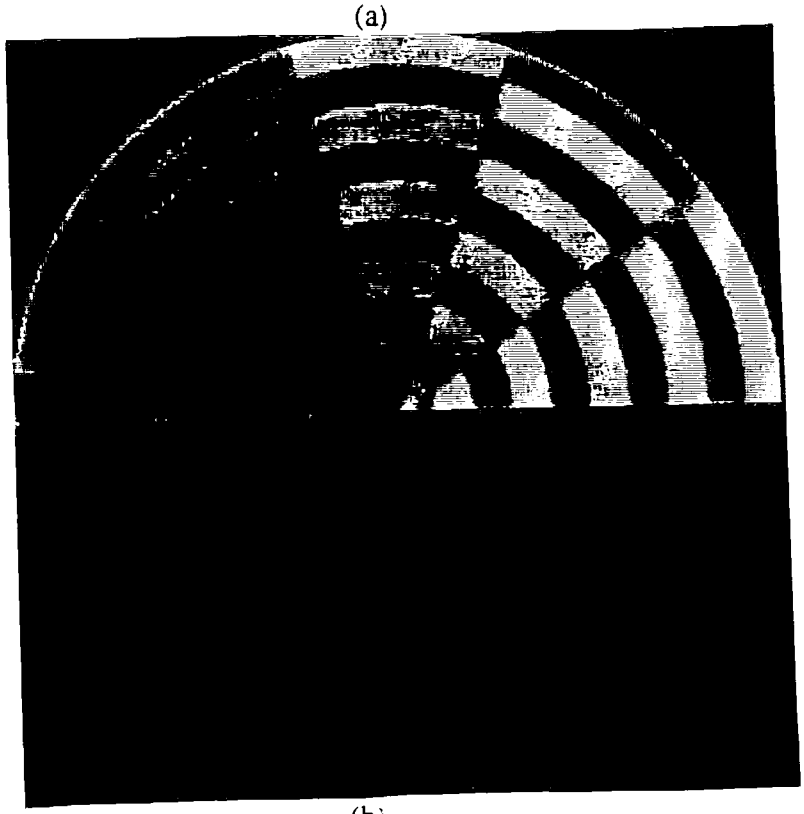
(a)



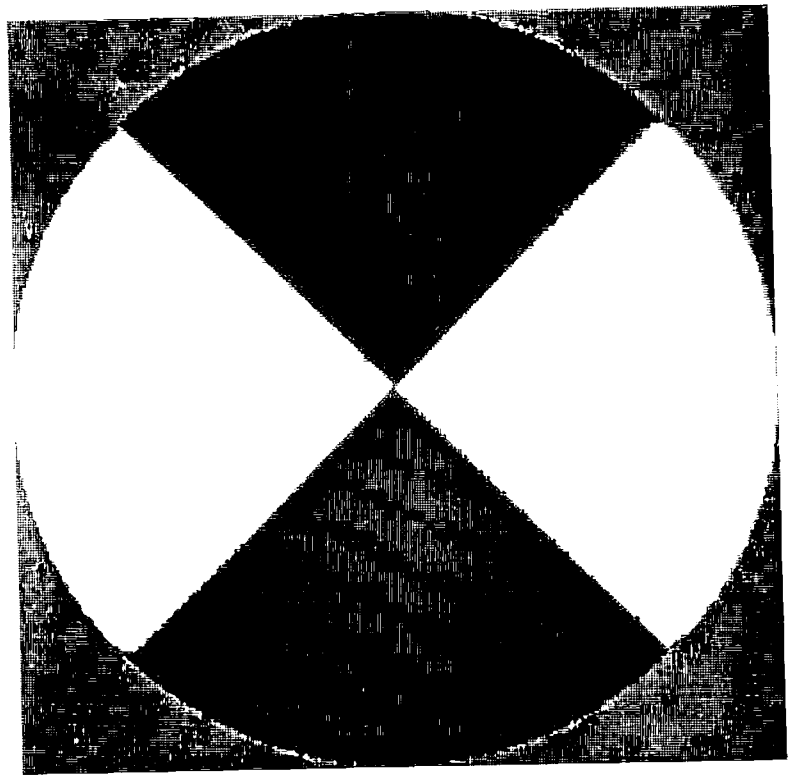
(b)



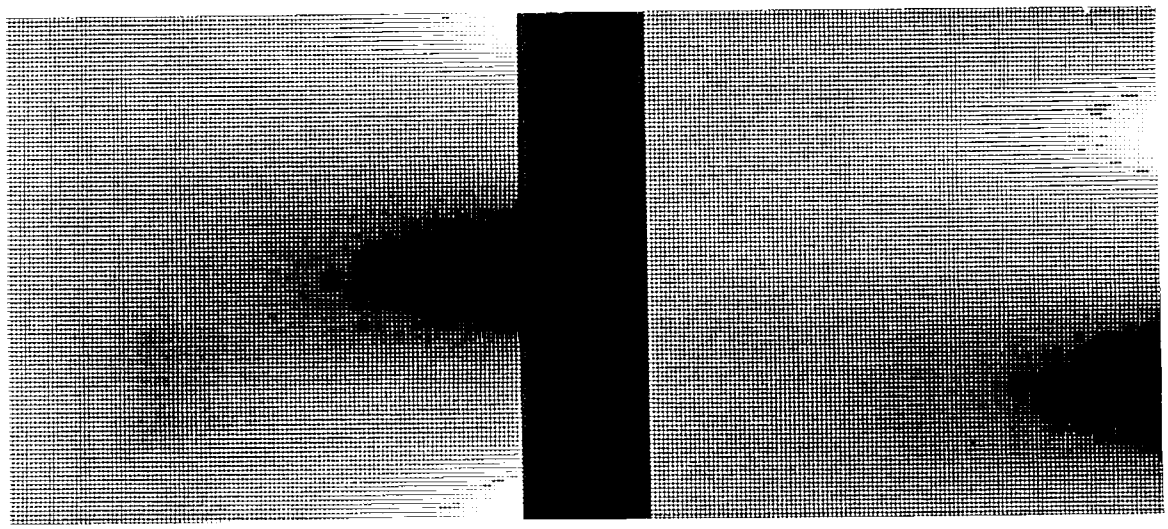
(a)



(b)

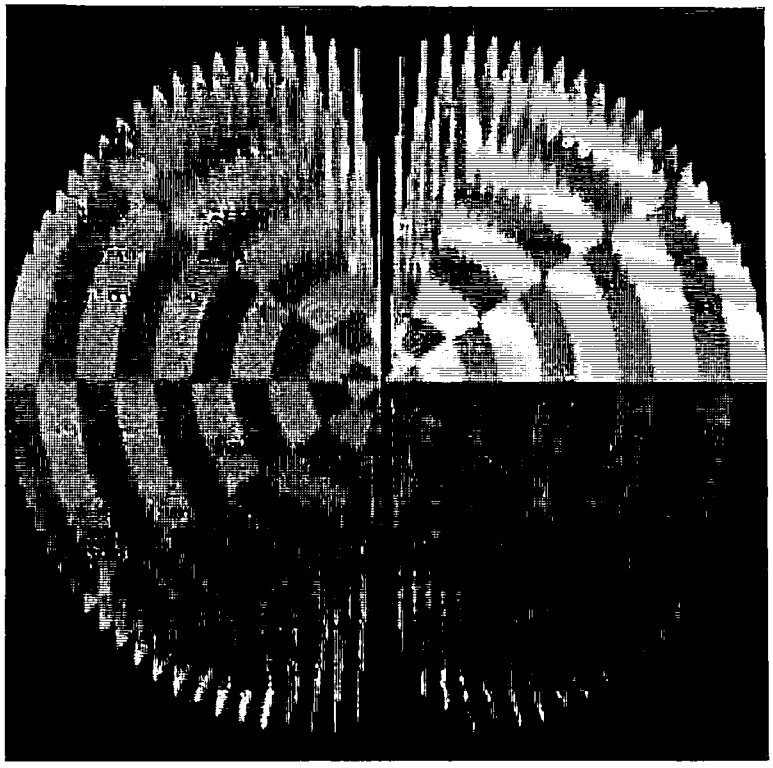


(c)

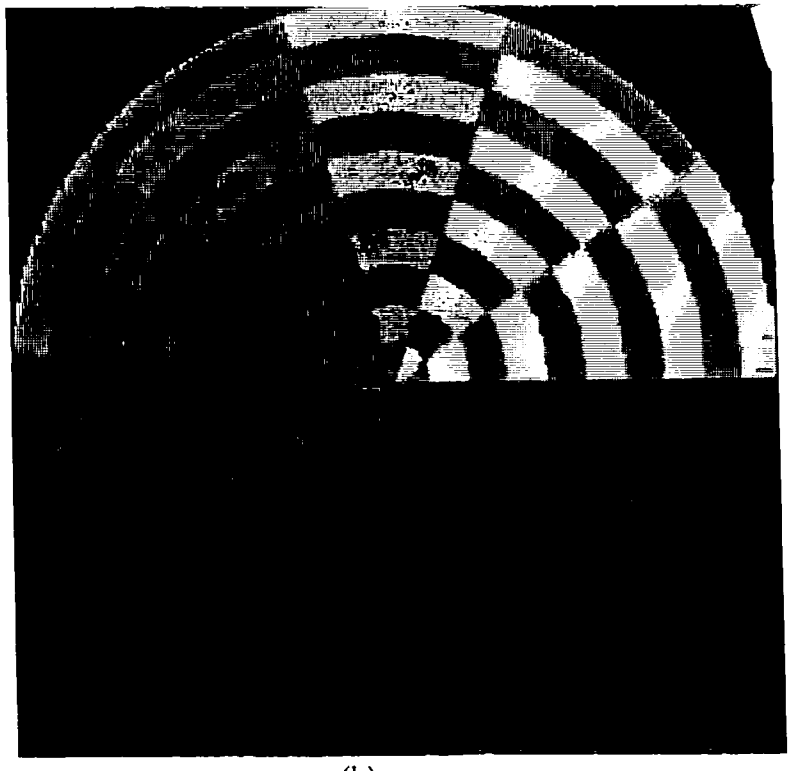


(d)

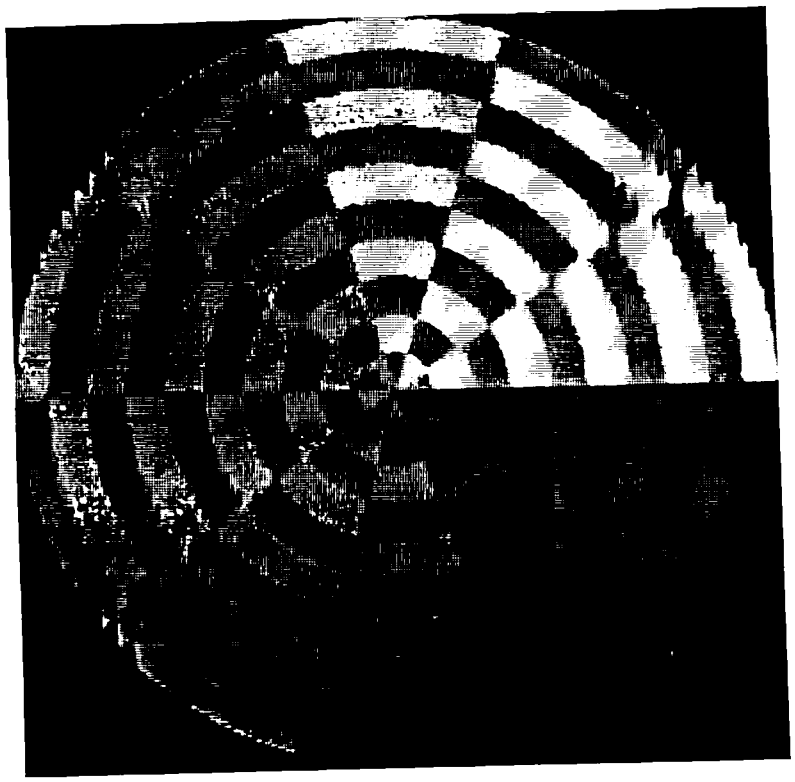
Figure 4.3: (a) I_{xy}; (b) I_{xy}^{T}; (c) S; (d) *XLUT* and *YLUT*.



(a)



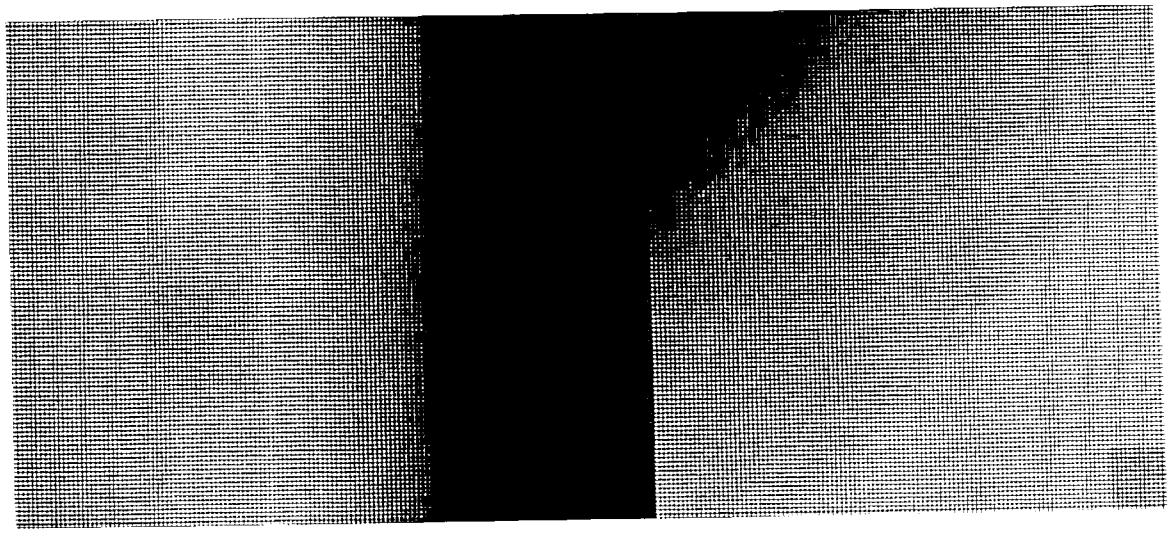
(b)



(c)

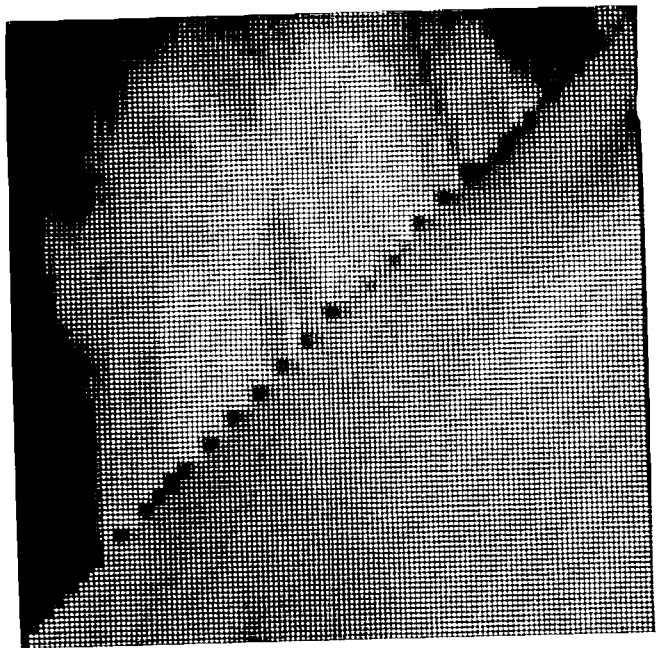
Figure 4.4: Undersampled spatial lookup tables. (a) I_{xy}; (b) I_{xy}^{T}; (c) Output.

(a)

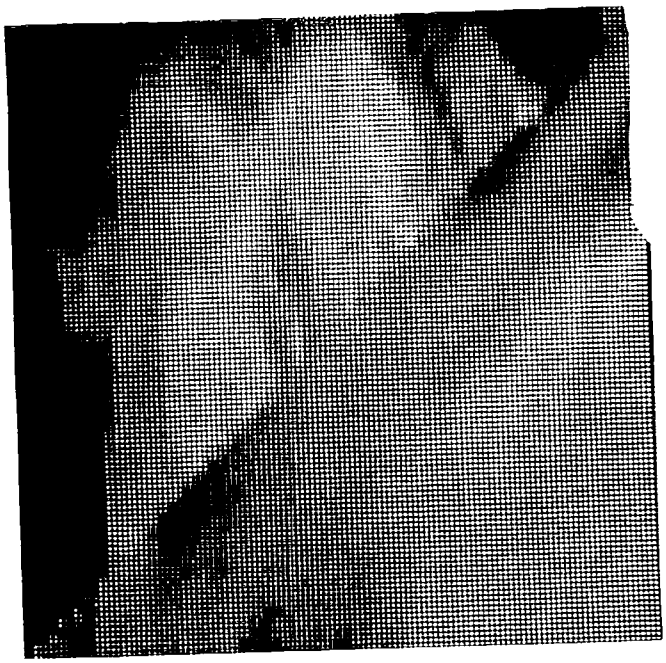


(b)

Figure 4.5: (a) Foldover; (b) *XLUT* and *YLUT*.

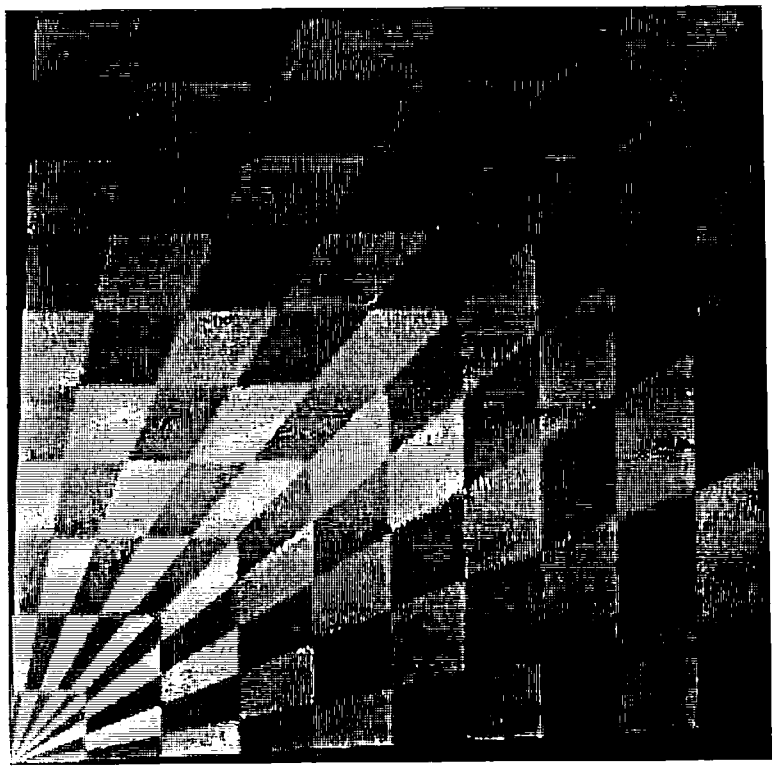


(a)



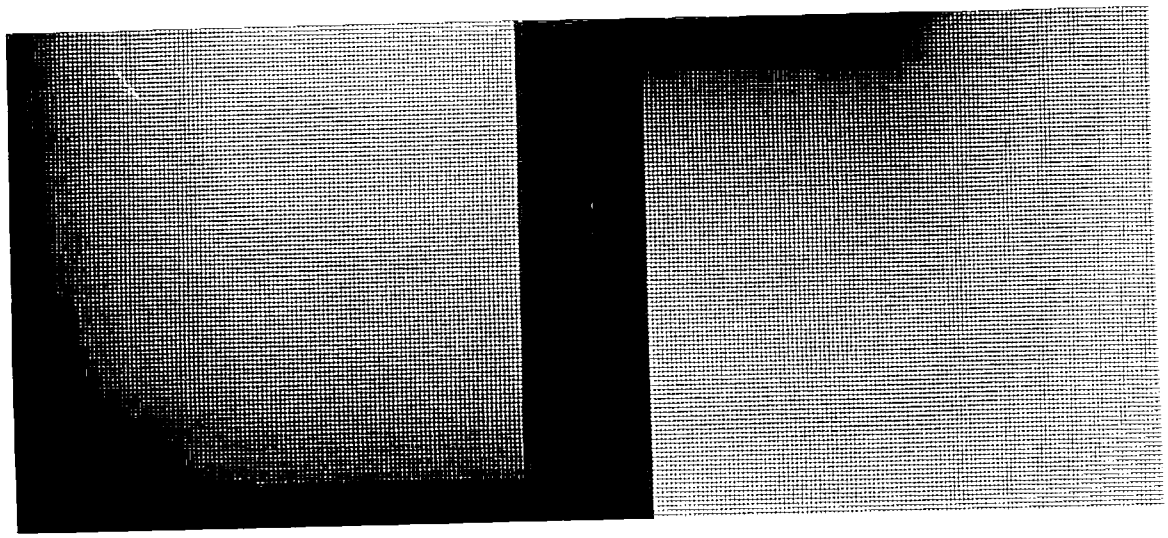
(b)

Figure 4.6: Magnified foldover. (a) No filtering. (b) Filtered result.

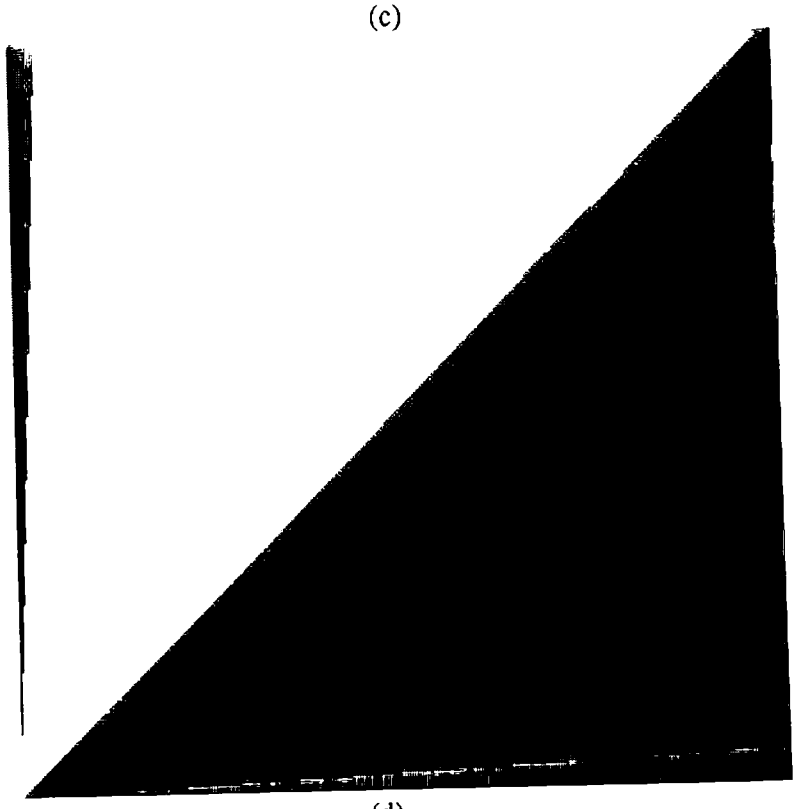


(a)

(b)

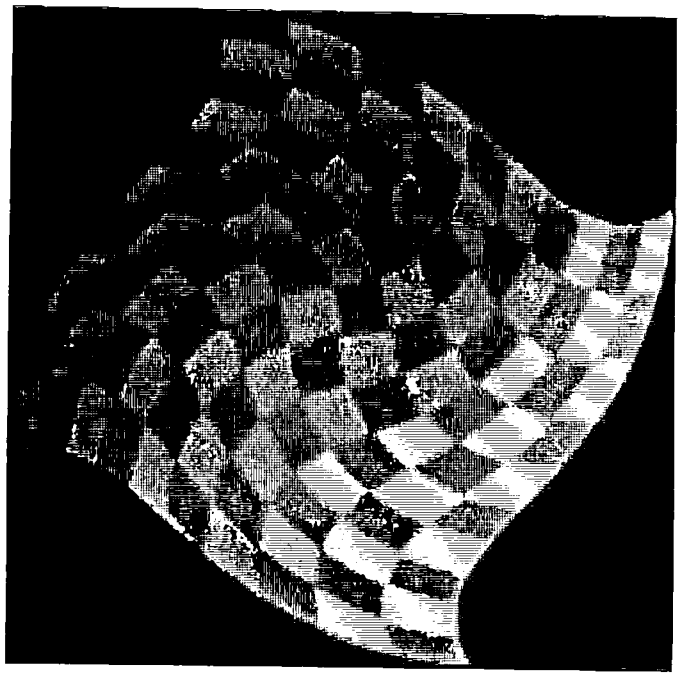


(c)



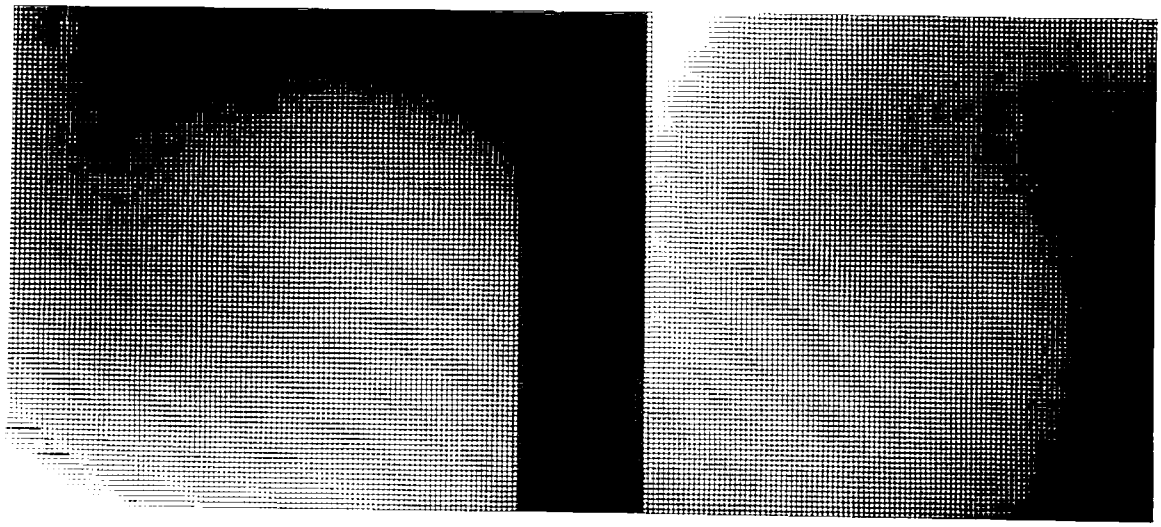
(d)

Figure 4.7: Corner warp. (a) Checkerboard; (b) Madonna; (c) *XLUT* and *YLUT*; (d) *S*.

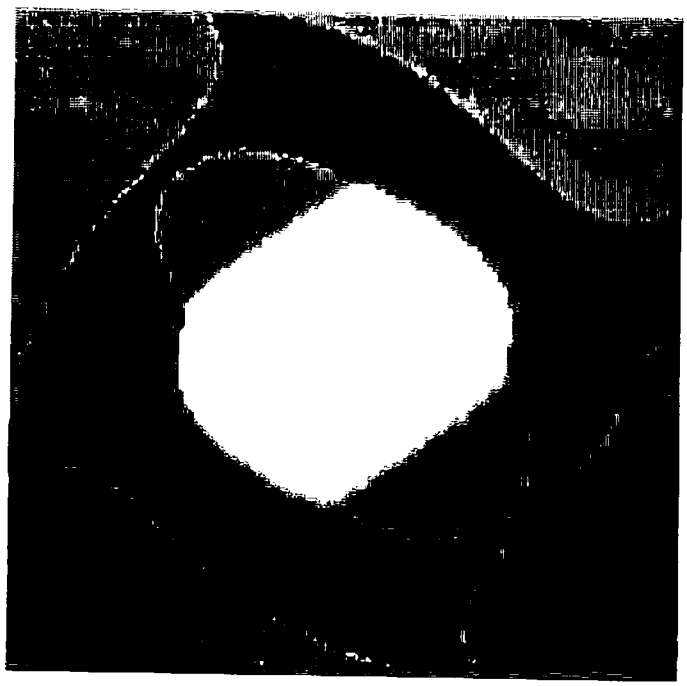


(a)

(b)



(c)



(d)

Figure 4.8: Vortex warp. (a) Checkerboard; (b) Madonna; (c) *XLUT* and *YLUT*; (d) *S*.

5. CONCLUSIONS AND FUTURE WORK

This paper describes a separable algorithm for realizing arbitrary warps which are specified by spatial lookup tables. It greatly extends the class of warps which can take advantage of separable techniques. As a result, it is possible to design real-time hardware built around off-the-shelf 1-D resamplers.

Our approach is based on the solution of the three main difficulties of the Catmull-Smith approach: bottlenecking, foldovers, and the need for a closed-form inverse. In addition, we addressed some of the errors caused by filtering, especially those caused by insufficient resolution in the sampling of the mapping function. Through careful attention to efficiency and graceful degradation, we developed a method that is no more costly than the Catmull-Smith algorithm when bottlenecking and foldovers are not present. However when these problems do surface, they are resolved at a cost proportional to their manifestation. Since the underlying data structures continue to facilitate pipelining, this method offers a promising hardware solution to the implementation of arbitrary spatial mapping functions.

The limitations of the work can be separated into two classes: inherent limitations, and those to be addressed by future work. The primary inherent limitation is that which is imposed by the separable nature of the filtering — the combination of two 1-D filtering operations is not as flexible as true 2-D filtering. A secondary inherent limitation is that, as a forward mapping, the entire input image and spatial map must be used even if much of it will be clipped out of the final image. Future work should allow the clipping to be done as part of the first pass coordinate resampling.

One of the limitations to be overcome by future work is that of incorporating local adaptive sampling in response to the local measure of shear. The difficulties to be overcome include data representation (the shape of the intermediate image might be far from rectangular), and proper reconstruction filtering for the spatial lookup tables.

A second avenue for future research is a detailed study of the error properties of the approach, and how they are affected by choice of reconstruction filter, processing order, and sampling density. We know that certain perspective distortions can be ameliorated by processing in different orders, and we hope that a local measure incorporating both bottleneck distortion and perspective distortion can be developed.

Further work remains to be done in extending this method to 3-D for warping volumetric data in medical imaging, e.g., CAT scans. Other applications include scientific visualization, whereby parameters may be mapped onto different warping functions. Visual music would then be possible by using filtered sound to perturb the spatial lookup tables. This would prove to be a valuable extension to existing visualization tools. Since warping is a process common to many disciplines, there are likely to be many applications which can exploit the results of this work.

6. REFERENCES

[1] Catmull, E. and A.R. Smith, "3-D Transformations of Images in Scanline Order," *Computer Graphics*, (SIGGRAPH '80 Proceedings), vol. 14, no. 3, pp. 279-285, July 1980.

[2] Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM Trans. on Graphics*, vol. 5, no. 1, pp. 51-72, January 1986.

[3] Dippe, M.A.Z and E.H. Wold, "Antialiasing Through Stochastic Sampling," *Computer Graphics*, (SIGGRAPH '85 Proceedings), vol. 19, no. 3, pp. 69-78, July 1985.

[4] Fant, K.M., "A Nonaliasing, Real-Time Spatial Transform Technique," *IEEE Computer Graphics and Applications*, vol. 6, no. 1, pp. 71-80, January 1986.

[5] Heckbert, P., "Survey of Texture Mapping," *IEEE Computer Graphics and Applications*, vol. 6, no. 11, pp. 56-67, November 1986.

[6] Mitchell, D., "Generating Antialiased Images at Low Sampling Densities," *Computer Graphics*, (SIGGRAPH '87 Proceedings), vol. 21, no. 4, pp. 65-72, July 1987.

[7] Paeth, A.W., "A Fast Algorithm for General Raster Rotation," *Graphics Interface* '86, pp. 77-81, May 1986.

[8] Smith, A.R., "Planar 2-Pass Texture Mapping and Warping," *Computer Graphics*, (SIGGRAPH '87 Proceedings), vol. 21, no. 4, pp. 263-272, July 1987.

[9] Tanaka, A., M. Kameyama, S. Kazama, and O. Watanabe, "A Rotation Method for Raster Image Using Skew Transformation," *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 272-277, June 1986.

[10] Weiman, C.F.R., "Continuous Anti-Aliased Rotation and Zoom of Raster Images," *Computer Graphics*, (SIGGRAPH '80 Proceedings), vol. 14, no. 3, pp. 286-293, July 1980.

[11] Wolberg, G., "Geometric Transformation Techniques for Digital Images: A Survey," Columbia University Computer Science Tech. Report CUCS-390-88, December 1988. To appear as a monograph by IEEE Computer Society Press.