Chapter 3

The Shear-Warp Factorization

As we have seen in the previous chapter, image-order algorithms such as ray casters have the advantages of efficient, high-quality resampling and a simple implementation of early ray termination. On the other hand, object-order algorithms such as splatters use simpler addressing arithmetic and they achieve greater speedups from spatial data structures because they traverse the volume in storage order.

The new algorithms in this thesis combine all of these advantages. The algorithms rely on a factorization of the viewing transformation that simplifies projection from the volume to the image, allowing us to construct an object-order algorithm with the same advantages as an image-order algorithm. We call the factorization the *shear-warp factorization*.

In this chapter we first define the shear-warp factorization and motivate its use with a simple brute-force volume rendering algorithm. Next we derive separate versions of the factorization for affine viewing transformations and for perspective viewing transformations. The affine factorization has been used before in several volume rendering algorithms but the perspective factorization is new. We then discuss the properties of the factorization which we will use in the next chapter to develop our new volume rendering algorithms. Finally, we close this chapter with a discussion of previous uses of the affine shear-warp factorization.

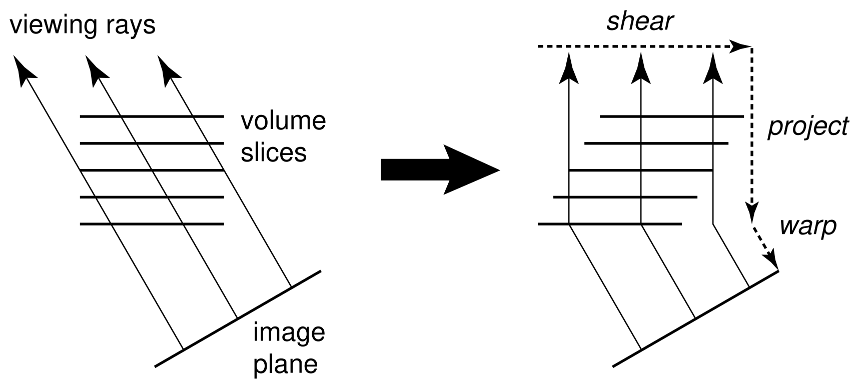

Figure 3.1: To transform a volume into sheared object space for a parallel projection we translate each slice. In sheared object space we can project the voxel slices into an image simply and efficiently.

3.1 An Overview of the Factorization

The arbitrary nature of the mapping from object space to image space complicates efficient, high-quality filtering and projection in object-order volume rendering algorithms and requires extra addressing arithmetic in image-order algorithms. This problem can be solved by transforming the volume to an intermediate coordinate system. We will choose a coordinate system with a simple mapping from the object coordinate system and which allows efficient projection to a 2D image.

We call the intermediate coordinate system "sheared object space" and define it as follows:

> Definition 1: By construction, in sheared object space all viewing rays are parallel to the third coordinate axis.

Figure 3.1 illustrates the transformation from object space to sheared object space for a parallel projection. The horizontal lines in the figure represent slices of the volume data viewed in cross-section. After transformation the volume has been sheared parallel to the set of slices that is most perpendicular to the viewing direction and the viewing rays are perpendicular to the slices. Since the shear is parallel to the slices, we can perform the transformation by simply translating each slice. For a perspective transformation the definition implies that each slice must be scaled as well as translated as shown schematically in Figure 3.2.

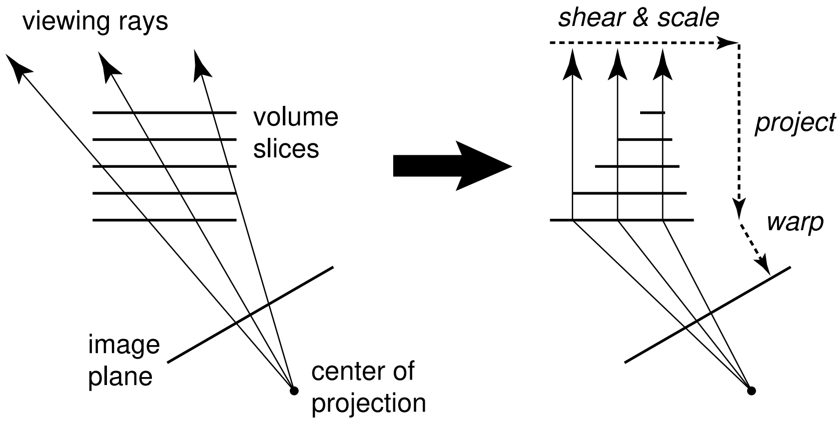

Figure 3.2: To transform a volume into sheared object space for a perspective projection we translate and scale each slice. In sheared object space we can again project the voxel slices into an image simply and efficiently.

A simple object-order volume rendering algorithm based on the transformation to sheared object space operates as follows (see Figure 3.3):

1. Transform the volume data to sheared object space by translating each slice. For perspective transformations, also scale each slice. Of the three possible slicing directions, choose the set of slices that is most perpendicular to the viewing direction. The translation and scaling coefficients are not necessarily integers so each voxel slice must be properly resampled.

2. Composite the resampled slices together in front-to-back order using the "over" operator. This step projects the volume into a distorted 2D intermediate image in sheared object space.

3. Transform the distorted intermediate image to image space by warping it. This second resampling step produces the correct final image.

Klein & Kübler [1985] and Cameron & Undrill [1992] proposed the parallel projection version of this algorithm, while the extension to perspective projections is new.

The reason for first computing a distorted intermediate image is that the properties of the factorization result in a very efficient implementation of the resampling and compositing loop. In particular, scanlines of voxels in each voxel slice are parallel to scanlines of

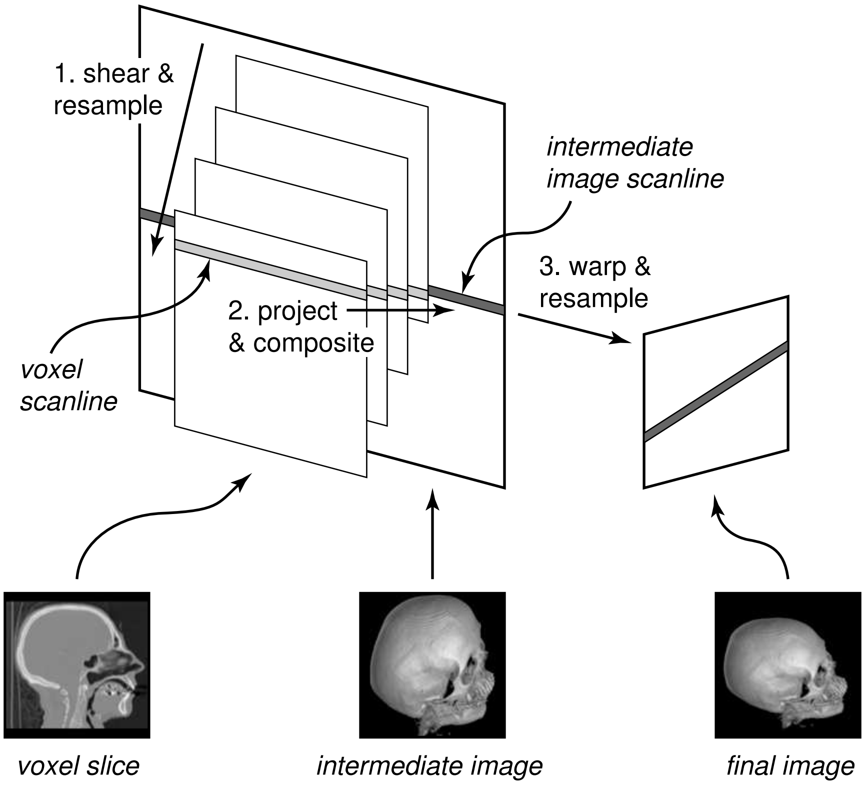

Figure 3.3: The brute-force shear-warp algorithm includes three conceptual steps: shear and resample the volume slices, project resampled voxel scanlines onto intermediate image scanlines, and warp the intermediate image into the final image. The shearing step shears the volume parallel to the volume slices in both the horizontal and vertical directions (although only a vertical shear is shown here). The horizontal shear accounts for rotation about the vertical axis and the vertical shear accounts for rotation about the horizontal axis. Rotation about the third axis is incorporated into the final warp. This algorithm works for arbitrary viewing transformations.

pixels in the intermediate image. Thus, the transformation from the object coordinate system to the intermediate image coordinate system (sheared object space) is very simple. We will elaborate on the properties of the factorization in Section 3.4. Transforming the 2D intermediate image into the final image requires a general warp but this operation is relatively inexpensive since the 2D intermediate image is much smaller than the 3D volume.

The brute-force shear-warp algorithm falls into a new category of the loop-order taxonomy introduced in the previous chapter:

for $z_o = 1$ **to** VolumeDepth
  for $y_i = 1$ **to** ImageHeight
    for $x_i = 1$ **to** ImageWidth
      foreach y_o in ResamplingFilter(x_i, y_i)
        foreach x_o in ResamplingFilter(x_i, y_i)
          add contribution of Voxel[x_o, y_o, z_o] to ImagePixel[x_i, y_i]

The outermost loop iterates over slices of the volume. The next two loops iterate over pixels in the image. Finally, the inner two loops iterate over the voxels in the slice that contribute to a particular image pixel. There are only two loops over the resampling filter kernel because the algorithm uses a 2D resampling filter. In this algorithm the outer loop traverses the volume in object order, but the inner loops are similar to an image-order algorithm.

This loop ordering combined with the shear-warp factorization enables an algorithm that traverses the volume in object order while using nearly the same resampling filter implementation as an image-order algorithm. The only difference in the filter is that most image-order algorithms use a 3D resampling filter. The filter differences have little impact on image quality as we will demonstrate in Section 5.1.2.

3.2 The Affine Factorization

The shear-warp factorization for affine viewing transformations follows directly from Definition 1. Although the factorization has been used before, the derivation presented here has not been previously described.

Let M_{view} be a 4x4 affine viewing transformation matrix that transforms points (represented as column vectors) from object space to image space. We want to factor M_{view} into two matrices:

$$M_{\text{view}} = M_{\text{warp2D}} \cdot M_{\text{shear3D}}$$

M_{shear3D} must be a shear matrix that transforms object space to sheared object space. By definition this requires shearing the coordinate system until the viewing direction becomes perpendicular to the slices of the volume.

In image space the viewing direction vector is $\vec{v}_i = (0, 0, 1)$. Let $\vec{v}_o$ be the viewing

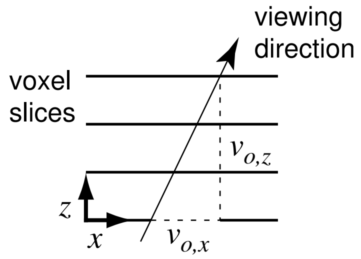

Figure 3.4: Determining the affine shear coefficients: This diagram shows a cross-section of the volume and the viewing direction projected onto the x-z plane (for the case where the principal viewing direction is the $+z$ axis). To transform the volume to sheared object space we must shear the volume in the x direction by $-v_{o,x}/v_{o,z}$. This shear causes the viewing direction to become perpendicular to the slices of the volume.

direction vector transformed to object space. It obeys the linear system of equations:

$$\vec{v}_i = M_{\text{view,3x3}} \cdot \vec{v}_o$$

where $M_{\text{view,3x3}}$ is the upper-left 3x3 submatrix of M_{view}. We can solve this system analytically for $\vec{v}_o$ using Cramer's Rule, yielding:

$$\vec{v}_o = \begin{bmatrix} m_{12}m_{23} - m_{22}m_{13} \\ m_{21}m_{13} - m_{11}m_{23} \\ m_{11}m_{22} - m_{21}m_{12} \end{bmatrix}$$

where the m_{ij} are elements of M_{view}.

We define the *principal viewing axis* to be the object-space axis that is most parallel to the viewing direction. For simplicity we assume in this derivation that the principal viewing axis is the $+z$ axis of the object coordinate system as shown in Figure 3.4 (a derivation without this restriction appears in Appendix A). In the x-z plane the slope of $\vec{v}_o$ is the ratio of the x and z components of the vector: $v_{o,x}/v_{o,z}$. The shear necessary in the x direction to make the viewing direction perpendicular to the constant-z slices equals the negative of this slope (because if the slope is zero no shear is necessary). A similar argument holds for the shear in the Y direction. Thus the shear coefficients are:

$$s_x = -\frac{v_{o,x}}{v_{o,z}} = \frac{m_{22}m_{13} - m_{12}m_{23}}{m_{11}m_{22} - m_{21}m_{12}}$$

$$s_y = -\frac{v_{o,y}}{v_{o,z}} = \frac{m_{11}m_{23} - m_{21}m_{13}}{m_{11}m_{22} - m_{21}m_{12}}$$

We can now write the two factors of the view transformation matrix. The first factor shears the volume:

$$M_{3D} = \begin{bmatrix} 1 & 0 & s_x & 0 \\ 0 & 1 & s_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Once this transformation has been applied to the volume the transformed voxel slices can be composited together along the $+z$ axis. This operation forms an intermediate image.

The second factor of the viewing matrix describes how to warp the intermediate image into the final image:

$$M_{\text{warp2D}} = M_{\text{view}} \cdot M_{\text{shear3D}}^{-1} = M_{\text{view}} \cdot \begin{bmatrix} 1 & 0 & -s_x & 0 \\ 0 & 1 & -s_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This matrix is a general affine warp.

Appendix A contains a more detailed derivation of the affine factorization including many implementation details.

3.3 The Perspective Factorization

The shear-warp factorization for perspective viewing transformations also follows from Definition 1. This factorization is new.

Let M_{view} be a 4x4 perspective viewing transformation matrix that transforms points

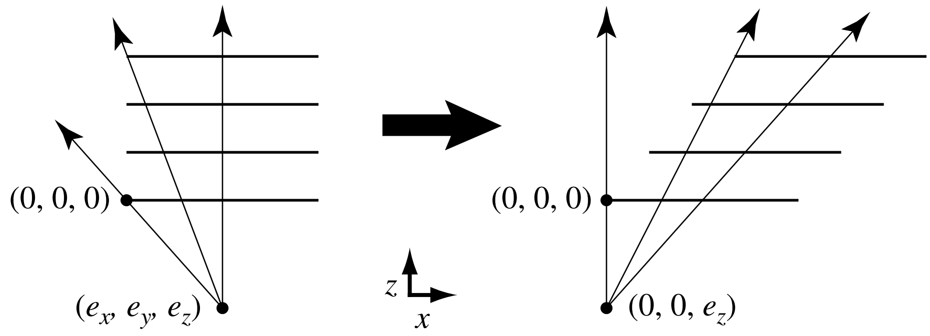

Figure 3.5: The shear transformation in the perspective shear-warp factorization. After the shear the viewing ray from the eyepoint (e_x, e_y, e_z) through the origin of the volume is perpendicular to the slices of the volume.

from object space to image space. We want to factor M_{view} into two matrices:

$$M_{\text{view}} = M_{\text{warp2D}} \cdot M_{\text{shear3D}}$$

The "shear" factor (M_{shear}) must contain both a shear transformation and a transformation that scales each volume slice in such a way that the viewing direction becomes perpendicular to the slices.

We will choose the shear component of M_{shear} such that after it is applied to the volume the ray from the eye (the center of projection) through the origin of standard object space becomes perpendicular to the slices of the volume (Figure 3.5). For simplicity we assume that the principal viewing direction is the $+z$ axis of the object coordinate system.

To compute the shear we must first determine the location of the eye in object space ($\vec{e}_o$) given just the viewing transformation matrix.[1] In image space the homogeneous coordinates of the eye are $\vec{e}_i = (0, 0, -1, 0)$. The viewing transformation matrix must transform $\vec{e}_o$ into $\vec{e}_i$:

$$\vec{e}_i = M_{\text{view}} \cdot \vec{e}_o$$

We can solve this system analytically using Cramer's Rule, yielding an expression for each component of the eyepoint expressed as a determinant:

[1] In the affine derivation we started by finding the viewing direction instead of the viewpoint. Alternatively, we could have used the viewpoint by representing it with a point at infinite distance from the volume.

$$e_{o,x} = -\begin{vmatrix} m_{12} & m_{13} & m_{14} \\ m_{22} & m_{23} & m_{24} \\ m_{42} & m_{43} & m_{44} \end{vmatrix} \qquad e_{o,y} = \begin{vmatrix} m_{11} & m_{13} & m_{14} \\ m_{21} & m_{23} & m_{24} \\ m_{41} & m_{43} & m_{44} \end{vmatrix}$$

$$e_{o,z} = -\begin{vmatrix} m_{11} & m_{12} & m_{14} \\ m_{21} & m_{22} & m_{24} \\ m_{41} & m_{42} & m_{44} \end{vmatrix} \qquad e_{o,w} = \begin{vmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{41} & m_{42} & m_{43} \end{vmatrix}$$

These equations give the homogeneous coordinates of the eye point in object space. For convenience, define the homogenized coordinates as follows:

$$\begin{aligned} e_x &= e_{o,x}/e_{o,w} \\ e_y &= e_{o,y}/e_{o,w} \\ e_z &= e_{o,z}/e_{o,w} \end{aligned}$$

By inspection (from Figure 3.5) the shear transformation is:

$$M_{\text{sh}} = \begin{bmatrix} 1 & 0 & -\frac{e_x}{e_z} & 0 \\ 0 & 1 & -\frac{e_y}{e_z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The second component of M_{shear} is the scale component which scales the slices of the volume so that the sheared viewing rays become parallel. Thus the required transformation is a pure perspective transformation (Figure 3.6). Without loss of generality we arbitrarily choose to scale the voxel slice closest to the viewer (the $z = 0$ slice) by a factor of one. To find the required transformation, consider a particular viewing ray passing through point $\vec{p}$ in the $z = 0$ slice and point $\vec{q}$ in some other slice (Figure 3.6, left side). The perspective transformation must scale the voxel slice containing $\vec{q}$ by p_x/q_x in the x direction and p_y/q_y in the y direction (because if $p_x = q_x$ and $p_y = q_y$ then no scale is necessary). Using the

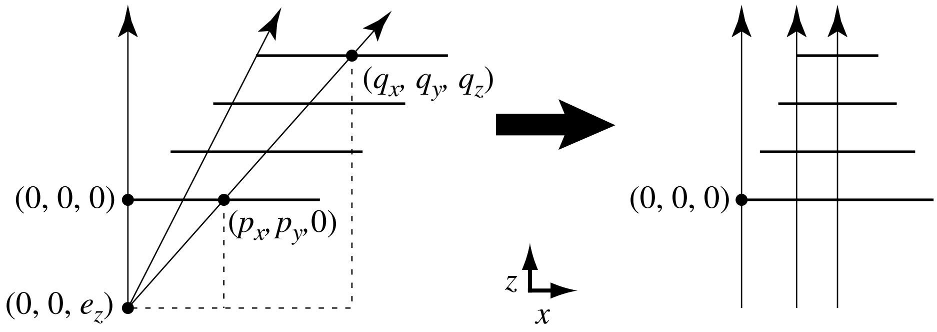

Figure 3.6: The scale transformation in the perspective shear-warp factorization. Each slice is scaled uniformly, but a different scale factor is applied to each slice. After the scale all of the viewing rays are perpendicular to the slices of the volume. This type of transformation is a pure perspective transformation.

similar triangles indicated by the dotted lines,

$$\frac{p_x}{q_x} = \frac{-e_z}{q_z - e_z} = \frac{1}{1 - q_z/e_z}$$

With a different set of similar triangles we can show that p_y/q_y equals the same expression. Thus the required perspective transformation is:

$$M_{\mathrm{s}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{e_z} & 1 \end{bmatrix}$$

The shear factor of the perspective shear-warp factorization is the product of the pure shear and the perspective scaling transformations. After substituting the homogeneous coordinates of the eye location we obtain:

$$M_{\mathrm{shear}} = M_{\mathrm{s}} \cdot M_{\mathrm{sh}} = \begin{bmatrix} 1 & 0 & -\frac{e_{o,x}}{e_{o,z}} & 0 \\ 0 & 1 & -\frac{e_{o,y}}{e_{o,z}} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{e_{o,w}}{e_{o,z}} & 1 \end{bmatrix}$$

This matrix specifies that to transform a particular slice of voxel data in the $z = z_0$ plane from object space to sheared object space the slice must be translated by $(-z_0 e_{o,x}/e_{o,z}, -z_0 e_{o,y}/e_{o,z})$ and then scaled uniformly by $1/(1 - z_0 e_{o,w}/e_{o,z})$. This matrix reduces to the affine shear matrix if the viewing transformation is affine.

The warp transformation equals the view transformation matrix multiplied by the inverse of the shear transformation matrix:

$$M_{\text{warp}} = M_{\text{view}} \cdot M_{\text{shear}}^{-1} = M_{\text{view}} \cdot \begin{bmatrix} 1 & 0 & \frac{e_{o,x}}{e_{o,z}} & 0 \\ 0 & 1 & \frac{e_{o,y}}{e_{o,z}} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{e_{o,w}}{e_{o,z}} & 1 \end{bmatrix}$$

This transformation is a general perspective warp.

We have now completed the derivation of the perspective shear-warp factorization. Refer to Appendix A for a more detailed derivation, including a discussion of how to find the principal viewing axis. In some cases it is impossible to choose a single axis, for example if the perspective projection specifies a wide field of view, since there may be no single slicing direction that is compatible with all viewing rays. In this case the volume must be subdivided, which is discussed with the implementation of the perspective rendering algorithm in Section 4.2.5.

3.4 Properties of the Factorization

The projection from the volume to the intermediate image has several geometric properties that simplify the rendering algorithm:

Property 1: Scanlines of pixels in the intermediate image are parallel to scanlines of voxels in the volume array.

Property 2: All voxels in a given voxel slice are scaled by the same factor.

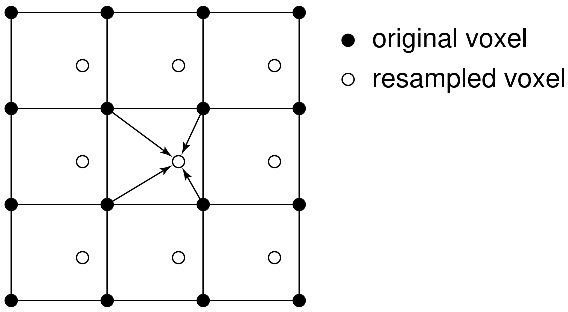

Figure 3.7: In the shear-warp algorithm for parallel projections each slice of the volume is simply translated, so as shown here every voxel in the slice has the same resampling weights.

Property 3 (parallel projections only): Every voxel slice has the same scale factor when projected into the intermediate image, and this factor can be chosen arbitrarily. In particular, we can choose a unity scale factor so that for a given voxel scanline there is a one-to-one mapping between voxels and intermediate-image pixels.

Property 1 follows from the fact that the shear matrix contains no rotation component. Property 2 follows from the fact that the shear matrix scales each voxel slice uniformly.

A useful implication of these properties for parallel projections is that every voxel in a given slice of the volume has the same resampling weights (Figure 3.7). Each slice of voxels is simply translated, so one set of resampling weights can be precomputed and then reused for every voxel. This fact eliminates the problems associated with efficient resampling in object-order volume rendering algorithms.

In the next chapter we will use these properties to develop efficient volume rendering algorithms.

3.5 Existing Shear-Warp Algorithms

Earlier uses of the shear-warp factorization for parallel projections include three applications: reducing the cost of projection in object-order volume rendering algorithms, reducing the cost of tracing rays in ray casting algorithms, and regularizing communication patterns in parallel volume rendering algorithms for massively-parallel SIMD arrays.

Klein & Kübler [1985] and Ylä-Jääski, Klein & Kübler [1991] describe an object-order rendering algorithm for volume data using the shear-warp factorization. This early work predates modern volume rendering, so the algorithm makes two simplifications that were common at the time: it thresholds each voxel to a binary value, thereby reducing the compositing operation used in volume rendering to the simpler problem of rendering the front-most non-transparent voxel, and it uses point sampling to resample the volume, thereby eliminating the cost of a resampling filter. Apart from these two differences, the algorithm is identical to the volume rendering algorithm for parallel projections presented in Section 3.1.

Ylä-Jääski, Klein & Kübler [1991] and Yagel & Kaufman [1992] independently proposed methods that use the shear-warp factorization to accelerate ray casting algorithms for parallel projections. Both algorithms take advantage of Property 3 of the factorization to reduce the amount of addressing arithmetic required for ray casting. A basic ray caster determines the step from one sample point to the next along a ray by adding a delta vector to the current sample location, converting the new floating-point sample location to an integer array index, and then computing resampling weights and looking up voxel values in the volume array. Alternatively, a discrete line drawing algorithm such as a Bresenham algorithm can be used to eliminate some of the floating point arithmetic. In either case, the addressing arithmetic calculations are a significant overhead in a ray caster (unlike in an object-order algorithm).

However, ray casters that use the shear-warp factorization simplify the computation by casting rays in sheared-object space instead of in image space. Suppose we cast one ray from the center of each intermediate image pixel (Figure 3.8). Then for a given slice of the volume each ray pierces the slice in such a way that the resampling weights are the same for every ray (recall Figure 3.7). Furthermore, if the array indexes (or memory offsets) of the voxels pierced by one ray are stored in a "template," that same template can be reused for every other ray by simply adding an integer offset to each value in the template. The precomputed template can be used to eliminate most of the computation associated with finding sample points along a ray, computing voxel addresses, and computing resampling weights. As in the object-order algorithm, the template-based ray casting algorithm computes a distorted intermediate image that must be warped into the correct final image. Using templates, Yagel reports speedups of roughly 2-3x compared to a brute force algorithm when

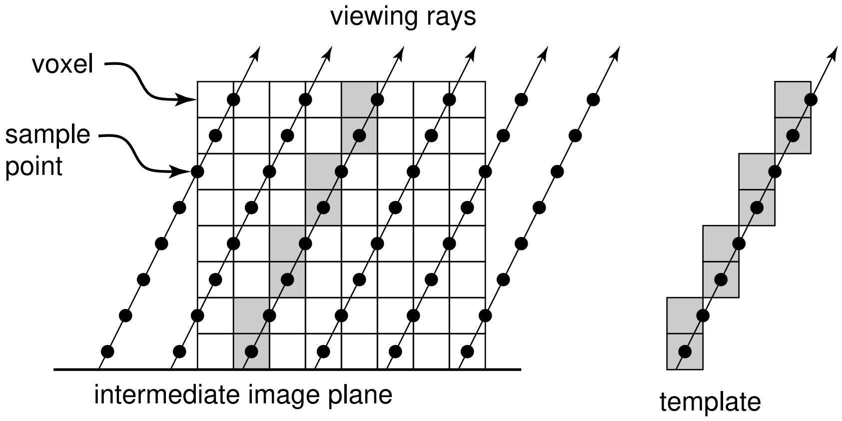

Figure 3.8: Ray casters based on the shear-warp factorization use a precomputed "template" containing resampling weights and memory offsets for the sample points in the volume (after Yagel & Kaufman [1992]).

discrete line drawing and nearest-neighbor sampling are used [Yagel & Kaufman 1992], and 1.3-1.4x when continuous line drawing and trilinear interpolation are used [Yagel & Ciula 1994].

Cameron & Undrill [1992] use the shear-warp factorization in a parallel volume rendering algorithm for a SIMD massively parallel multiprocessor. They also use the object-order rendering algorithm described in Section 3.1. They parallelize the algorithm by assigning each pixel in the intermediate image to a different processor. The volume is loaded into the processor memories in such a way that each voxel in a slice of the volume is assigned to a different processor. The voxels in each slice are shifted via a 2D mesh network to implement the shear and then composited into the intermediate image in parallel. The fact that intermediate image scanlines and voxel scanlines are aligned (Property 1) guarantees that the shift can be performed using the regular communication of a 2D mesh network, and the one-to-one mapping between pixels and voxels (Property 3) ensures that the voxels required to compute a given intermediate image pixel are always available in the local memory or the immediately-adjacent memories in the 2D mesh. The shear-warp factorization therefore enables the algorithm to work on a 2D SIMD array because of the regular communication patterns and the identical processing required for each voxel.

Schröder & Stoll [1992] describe a parallel volume rendering algorithm for a SIMD massively-parallel multiprocessor that independently uses the same rendering algorithm as

Yagel. The algorithm assigns one column of voxels (aligned with the principal viewing axis) to each processor. As the algorithm projects rays from the intermediate image through the volume the partially-computed ray colors and opacities are communicated to the processor containing the next required voxel. This algorithm computes the ray template on-the-fly during rendering. Because the template is the same for every ray the processing for each ray is identical and the algorithm can communicate the partially-accumulated ray values using nearest-neighbor shifts on a 2D mesh network. Just as in Cameron and Undrill's algorithm, the shear-warp factorization makes the communication patterns very regular, although in Schröder and Stoll's algorithm the ray values are communicated instead of the volume data.

To summarize, Klein, Kübler and Ylä-Jääski's object-order algorithm uses the shear-warp factorization to simplify projecting voxels into an image. Their ray casting algorithm and Yagel's template method both use the factorization to reduce the cost of tracing a ray. Cameron and Undrill's algorithm and Schröder and Stoll's algorithm both use the factorization to construct volume renderers with regular communication patterns appropriate for brute-force SIMD parallel implementations.

We extend this work by introducing a factorization for perspective projections and by exploiting the properties of the factorization in new ways to improve the performance of coherence optimizations. These topics are the focus of the next chapter.

3.6 Chapter Summary

The shear-warp factorization allows us to incorporate the advantages of ray casting algorithms into an object-order algorithm. In this chapter we presented an existing version of the factorization for parallel projections and a new version for perspective projections. In the next chapter we will combine the factorization with the optimizations from the previous chapter to develop three new volume rendering algorithms. We will exploit the properties of the factorization to maximize the benefit of spatial data structures and early ray termination.