Chapter 5

Performance Analysis

Spatial coherence is a well-known means for accelerating volume rendering without degrading image quality. Although speedups of more than an order of magnitude over brute-force algorithms have been reported, the sources of these speedups and the relative merits of competing algorithms are not well-understood. This chapter presents a detailed performance analysis of the shear-warp algorithms and compares them to other coherence-accelerated algorithms.

In the first part of the chapter (Section 5.1) we report on the speed and image quality of the new shear-warp volume rendering algorithms. The results show that the parallel projection algorithm can render 256^3 voxel medical data sets in less than one second on a 150 MHz R4400-based workstation without sacrificing image quality.

In the second part of the chapter (Section 5.2) we compare the shear-warp algorithms to image-order algorithms, both with and without coherence accelerations. We use an octree-accelerated ray caster as a representative example of image-order algorithms. We first show that the observed computational growth rates of a brute-force volume rendering algorithm, the octree-accelerated ray caster, and the shear-warp algorithm are n^3, $n^2 \log_2 n$ and n^2 respectively. We then show that the performance of each algorithm is governed not by its complexity, but by the associated constants. We do so by decomposing execution time into categories that distinguish between essential computation, overhead due to the coherence data structures, and memory overhead. This information allows us to pinpoint the differences between the algorithms.

Finally in the third part of the chapter (Section 5.3) we analyze and measure the performance of the shear-warp algorithms with low-coherence data.

From this analysis we make four observations regarding coherence-accelerated volume rendering algorithms:

- The time spent traversing spatial data structures is at least as great as the time spent resampling and compositing voxels.
- Octree-accelerated image-order algorithms spend more time traversing spatial data structures than object-order algorithms because the former must perform analytic geometry calculations (e.g. intersecting rays with axis-aligned boxes). As a result, the shear-warp algorithm is 4-10x faster than the octree-accelerated ray caster.
- Memory overhead is important in brute-force algorithms but not in coherence-accelerated algorithms.
- Coherence-accelerated volume rendering algorithms can be made to degrade gracefully in the presence of low data coherence.

These observations suggest that scanline-order volume rendering algorithms such as the shear-warp algorithm are likely to dominate over other algorithms, at least for software implementations.

5.1 Performance of the Shear-Warp Algorithms

5.1.1 Speed and Memory Performance

Table 5.1 shows the characteristics of our test data sets, including the sizes and the percentages of non-transparent voxels. The "brain" data set is an MR scan of a human head (Figure 5.1) and the "head" data set is a CT scan of a human head (Figure 5.2). The "brainsmall" and "headsmall" data sets are decimated versions of the larger volumes. The tests use classification functions that produce opaque surfaces and shading functions based on the Phong illumination model implemented with a lookup table. The lookup table maps a precomputed surface normal vector to a color, as described later in Section 7.1. We report rendering times that include the cost of recalculating the entries in the lookup table (which must be performed every time the viewpoint changes).

Table 5.2 lists some additional rendering parameters used in the experiments. All of the results in this chapter were measured on a 150 MHz SGI Indigo2 R4400 workstation without hardware graphics accelerators. This machine has an integer performance rating of 90 SPECint92 and a floating point performance rating of 87 SPECfp92.

The performance results for the three new algorithms are summarized in Table 5.3. The "Fast Classification" timings are for the fast classification algorithm with a parallel projection. The rendering times include all steps required to render from a new viewpoint, including shading, compositing and warping, but the preprocessing step to compute the spatial data structure is not included. The "Avg. Time" field in the table is the average time per

Data set	Size (voxels)	Memory before classification (Mb)	Non-transparent voxels (%)	Image
brainsmall	128x128x84	1.3	18	
headsmall	128x128x113	1.8	6	
brain	256x256x167	10	11	Fig. 5.1
head	256x256x225	14	5	Fig. 5.2
head (skin)	256x256x225	14	11	Fig. 5.5
engine	256x256x110	6.9	8	Fig. 5.6
abdomen	257x257x159	10	11	Fig. 5.7

Table 5.1: Characteristics of the test data sets for the results in this chapter.

Figure 5.1: Volume rendering of an MR scan of a human brain using the shear-warp algorithm with a parallel projection (0.86 sec.).

Figure 5.2: Volume rendering of a CT scan of a human head with a parallel projection (0.80 sec.).

Minimum opacity threshold	0.05
Early ray termination threshold	0.95
Min-max octree parameter threshold	4
Smallest min-max octree node size	4^3 voxels
Image size	256^2 pixels

Table 5.2: Rendering parameters used for the performance tests in this chapter. The minimum opacity threshold determines whether or not a voxel is transparent. Both opacity thresholds are specified on a scale of 0.0 (transparent) to 1.0 (fully opaque). The min-max octree parameter threshold is the largest variation allowed for the parameters of the opacity transfer function in an octree leaf node; nodes with larger variations are subdivided until the size reaches the smallest allowed node size. The octree parameters are described in Section 4.3.2.

Data set	Avg. Time	Min/Max Time	Memory
Parallel projection algorithm			
brainsmall	0.22 sec.	0.19–0.26 sec.	4 Mb.
headsmall	0.25	0.23–0.28	2
brain	0.86	0.69–1.06	19
head	0.87	0.77–0.98	13
Perspective projection algorithm			
brainsmall	0.48 sec.	0.42–0.54 sec.	4 Mb.
headsmall	0.57	0.53–0.62	2
brain	2.04	1.67–2.46	19
head	2.35	2.12–2.55	13
Fast classification algorithm			
brainsmall	0.40 sec.	0.35–0.47 sec.	8 Mb.
headsmall	0.54	0.49–0.61	8
brain	1.94	1.51–2.36	46
head	2.19	1.92–2.39	61

Table 5.3: Rendering time and memory usage on a 150 MHz SGI Indigo2 workstation. Times include shading, resampling, projection and warping. The fast classification times include rendering with a parallel projection. The "Memory" field is the total size of the data structures used by each algorithm; the first two algorithms use a compressed run-length encoded volume while the fast classification algorithm uses an uncompressed voxel array. Table 5.1 describes the data sets.

Image Size	64^2	128^2	256^2
Warp Time (msec.)	5.0	17	64
% Time, 64^3 volume	7	20	48
% Time, 128^3 volume		11	31
% Time, 256^3 volume			9

Table 5.4: Execution time for the 2D warp. The time is expressed both in milliseconds and as a percentage of total rendering time for several image and volume sizes. Unless the image is much larger than the volume, the percentage is small.

frame for rendering 360 frames at two-degree-angle increments, and the "Min/Max" times are for the best and worst case angles. The "Memory" field gives the size in megabytes of all data structures. For the first two algorithms the size includes the three run-length encodings of the volume, the image data structures and lookup tables for shading and opacity correction. For the third algorithm the size includes the unencoded voxel array, the octree, the summed-area table, the image data structures, and the lookup tables.

For comparison purposes, we rendered the same data sets with a ray caster that uses early ray termination and a pyramid to exploit object coherence. Because of its lower computational overhead the shear-warp algorithm is 4-10x faster. In Section 5.2 we compare the two algorithms in more detail.

Cost of the 2D Warp Table 5.4 shows the time required to warp images of three different sizes and expresses the time as a percentage of total rendering time for three different volume sizes. These results are for a parallel projection of the MR "brain" volume prefiltered to the three resolutions. The implementation of the 2D warp uses bilinear interpolation. These results show that unless the image resolution is much larger than the volume resolution the performance overhead incurred by the warp is a small percentage of overall rendering time. Images with sizes much larger than the volume are not useful since the rendering algorithms sample the volume at object resolution. The volume should always be prescaled to roughly match the desired image size.

Dependence of Rendering Time on Viewing Angle Rendering time depends on viewing angle, primarily because the effectiveness of the coherence optimizations may vary with

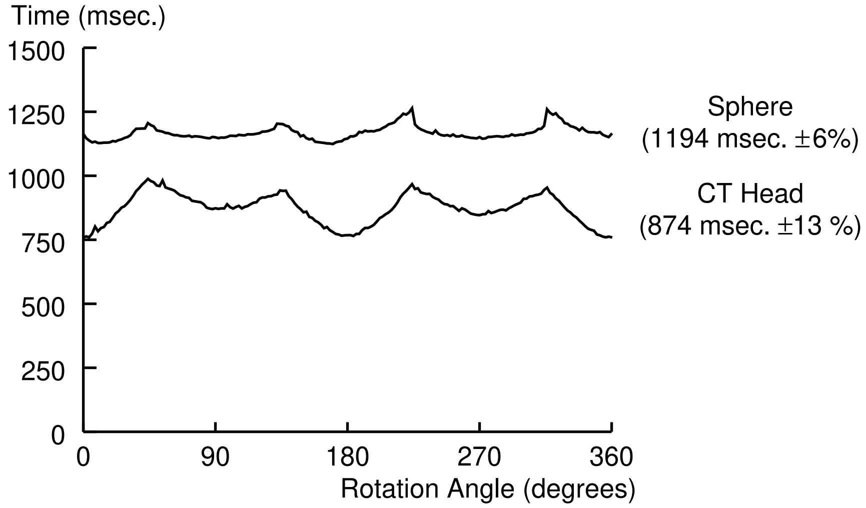


Figure 5.3: Rendering time versus viewing angle for a parallel projection of the CT head data set and a 256^3 voxel data set containing a sphere.

viewpoint. Figure 5.3 shows the rendering time as a function of viewing angle with the parallel projection algorithm for the 256^3 CT "head" data set and for a 256^3 voxel synthetic data set containing a single large sphere. Since the sphere looks identical from all directions the effect of the coherence optimizations does not change with viewpoint, and the rendering time varies by only 6%. For the "head" data set the complexity of the rendered image does change and the variation in rendering time is 13%. There are secondary costs that also vary with viewpoint (for example the size of the intermediate image increases as the rotation angle approaches 45 degrees so the time required to initially clear it increases), but these effects are negligible.

Figure 5.3 also shows that there is no jump in rendering time when the major viewing axis changes. This fact holds true provided the three run-length encoded copies of the volume fit into real memory simultaneously. In the current implementation each copy of the volume contains four bytes per non-transparent voxel and one byte per run. For the 256x256x225 voxel head data set the three run-length encodings total only 9.8 Mbytes. An increase in rendering time would occur only if main memory were not large enough to hold the compressed volume, which would cause paging from the disk.

One might expect cache performance to degrade when the major viewing axis changes, but the results in Figure 5.3 show that there is no impact on overall performance. This result can be explained by two effects. First, as in other volume rendering algorithms [Nieh & Levoy 1992], the ratio of voxel memory references to executed instructions is low. Even ignoring the complicated control logic for traversing the coherence data structures, the inner loop of the rendering algorithm contains about 100 R4400 machine instructions to resample four voxels and composite the result into the image. Only eight of those instructions load voxel data (four opacities and four surface normal vectors for shading). Second, the volume is always traversed in scanline order so the accesses to the volume data have very good spatial locality. Thus these eight load instructions are likely to hit in the cache.

If all eight voxel load instructions resulted in cache misses then the memory overhead could be non-negligible, since miss penalties of 50-100 cycles are becoming common in modern processors. The fast classification algorithm suffers from this problem if the voxel array is not properly transposed, in which case the algorithm does not traverse the volume in scanline order. For example the fast classification algorithm renders the 256^3 sphere data set in 2.7 seconds if it is properly transposed versus 4.0 seconds if it is not.

Cost of Preprocessing and Switching Modes Figure 5.4 gives a breakdown of the time required to render the brain data set with a parallel projection using the fast classification algorithm (left branch) and the fast rendering algorithm (right branch). The "Switch Modes" arrow shows the time required for all three copies of the run-length encoded volume to be computed from the unencoded volume and the min-max octree once the user has settled on an opacity transfer function.

The diagram also shows the time for preprocessing the volume. This step is independent of both the viewpoint and the classification function. The preprocessing includes computing the gradient of the scalar field, the surface normal vectors (an encoded representation of the gradient), and the min-max octree.

Rendering Time for Color Images Compared to gray scale renderings, color renderings take roughly twice as long when using a parallel projection and 1.3x longer when using a perspective projection because of the additional resampling and compositing required for the two extra color channels. The slowdown is worse in the parallel projection algorithm

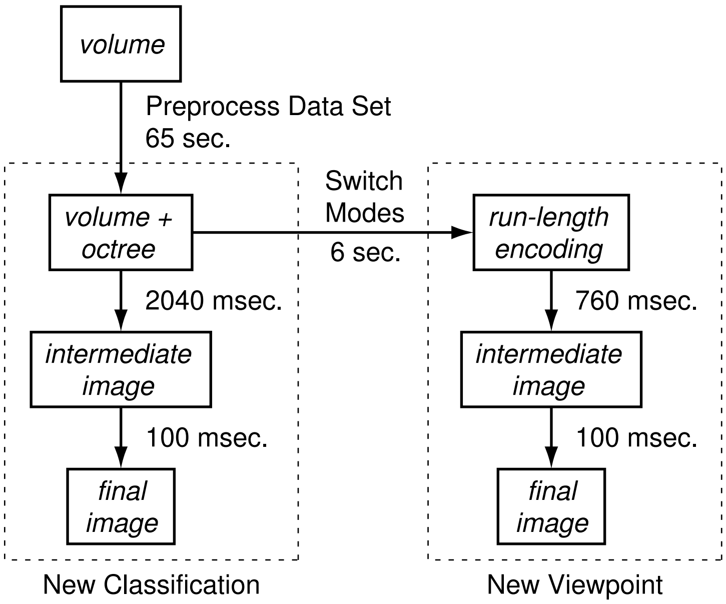


Figure 5.4: Execution time for each stage of rendering the brain data set with a parallel projection. The left side uses the fast classification algorithm and the right side uses the fast rendering algorithm.

because in the perspective projection algorithm a larger percentage of the overall rendering time goes to control overhead and computing resampling weights, and these overheads are not affected by the number of color channels.

Figures 5.5–5.7 show several examples of color volume renderings produced with the parallel projection rendering algorithm. Figure 5.5 is a rendering of the "head" data set classified with semitransparent skin which took 2.8 sec. to render. Compared to Figure 5.2, 630 msec. of the increased time results from using a different classification, 950 msec. of the increase is due to resampling and compositing RGB colors instead of gray scale, and 450 msec. of the increase is due to a more complicated shading function (separate shading lookup tables were used for voxels representing skin and voxels representing bone). Figure 5.6 is a rendering of a 256x256x110 voxel engine block, classified with semi-transparent and opaque surfaces; it took 2.0 sec. to render. Figure 5.7 is a rendering of a 257x257x159 voxel CT scan of a human abdomen, rendered in 2.2 sec. The blood vessels of the subject contain a radio-opaque dye, and the data set was classified to reveal both the dye and bone surfaces. The anatomical structures were segmented manually (a drawing program was used to tag each voxel with an identifier so that the kidneys, blood vessels and bone

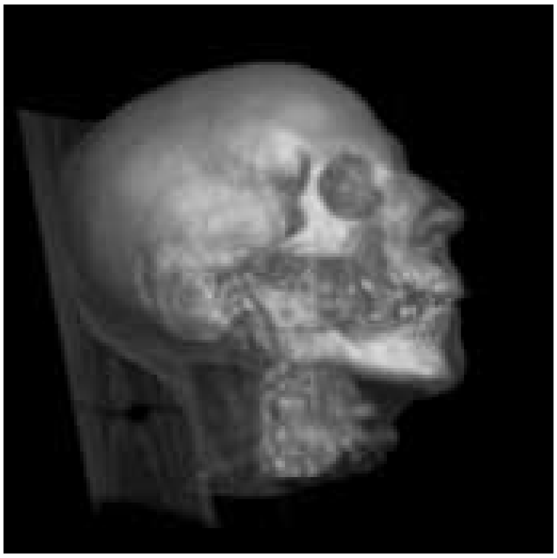


Figure 5.5: Color volume rendering of the CT head data set with a parallel projection, classified with semitransparent skin (2.8 sec.).

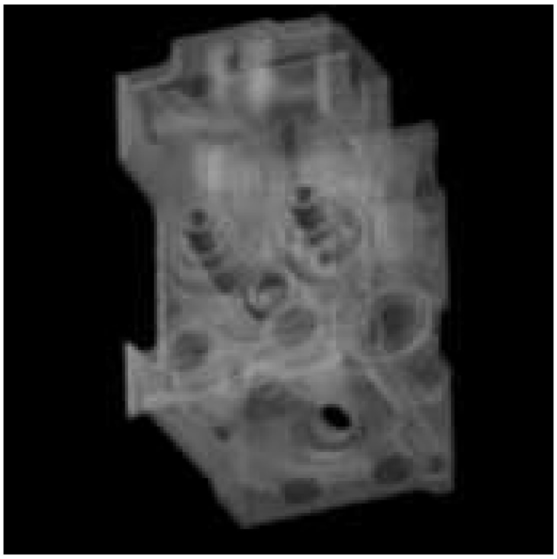


Figure 5.6: Color volume rendering of an engine block with a parallel projection, classified with semitransparent and opaque surfaces (2.0 sec.).

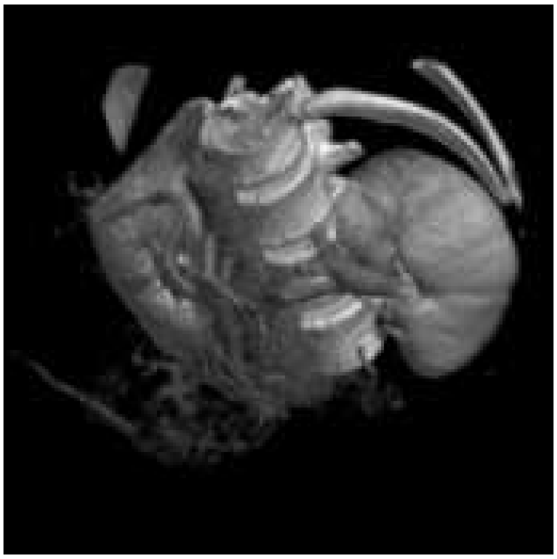


Figure 5.7: Color volume rendering of a CT scan of a human abdomen with a parallel projection (2.2 sec.). The blood vessels contain a radio-opaque dye.

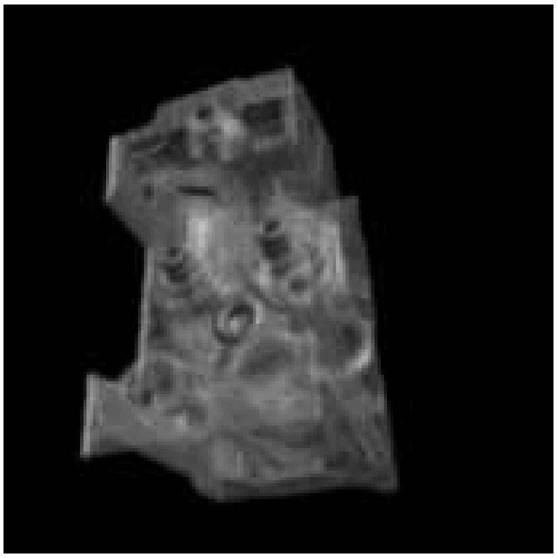


Figure 5.8: Color volume rendering of the engine data set with a perspective projection (2.6 sec.).

Figure 5.9: Image quality comparison of an image rendered by the shear-warp algorithm (left) with an image rendered by a ray caster using trilinear interpolation (right). The data set is a 256^3 voxel CT scan of a head, oriented at 45 degrees relative to the axes of the volume.

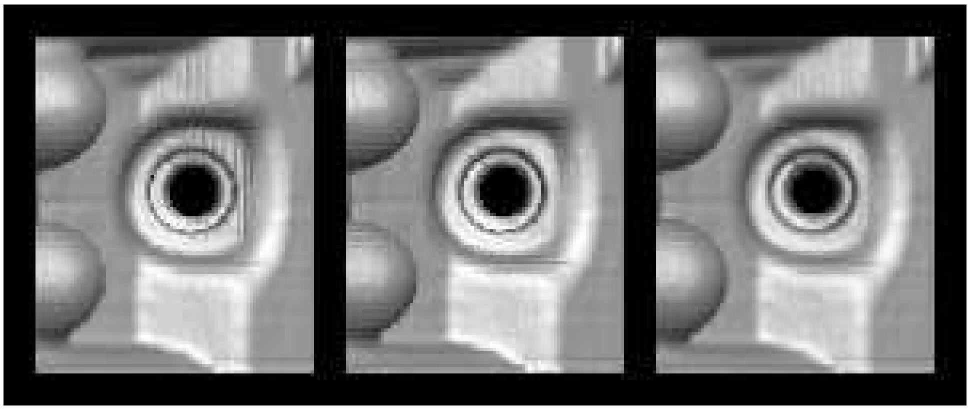
structures could be shaded differently).

Figure 5.8 is a color volume rendering of the engine data set using the perspective projection algorithm. It took 2.6 sec. to compute.

5.1.2 Image Quality

Figure 5.9 shows two volume renderings, one computed with the shear-warp algorithm and one computed with a ray caster using trilinear interpolation. The two images are virtually identical.

Nevertheless, the shear-warp algorithm imposes several limitations on the filter used to resample the volume and these limitations can potentially result in image degradation. The first limitation is that two resampling steps are required: the algorithm resamples the volume slices and then resamples the intermediate image to produce the final image. Multiple resampling steps can potentially cause blurring and loss of detail. However in practice this problem is not severe. Even in the high-detail regions of Figure 5.9 there is no noticeable blurring.



(a) (b) (c)

Figure 5.10: Comparison of image quality with bilinear and trilinear filters for a portion of the engine data set. The images have been enlarged by replicating pixels. (a) Bilinear filter with binary classification. (b) Trilinear filter with binary classification. (c) Bilinear filter with smooth classification.

A second limitation is that the shear-warp algorithm uses a 2D rather than a 3D reconstruction filter to resample the volume data. A high-quality filter can be used to accurately reconstruct each voxel slice, but the algorithm does not interpolate data between voxel slices. Aliasing artifacts can appear if the opacity or color attributes of the volume contain high frequencies in the dimension perpendicular to the slices (although if the frequencies exceed the Nyquist rate then these artifacts are unavoidable).

Figure 5.10 shows a case where a trilinear interpolation filter outperforms a bilinear filter. The left image is a rendering produced with the shear-warp algorithm of a portion of a data set that has been classified with extremely sharp ramps to produce high frequencies in the volume's opacity. The viewing angle is set to 45 degrees relative to the axes of the data set—the worst case—and aliasing is apparent. For comparison, the middle image is a rendering produced with a ray caster using trilinear interpolation and otherwise identical rendering parameters; here there is virtually no aliasing. However, with a smoother opacity transfer function these filtering artifacts can be reduced. The right image is a rendering using the shear-warp algorithm and a less-extreme opacity transfer function. Here the aliasing is barely noticeable because the high frequencies in the scalar field have effectively been low-pass filtered by the transfer function.

In practice, if the original volume is band-limited and the opacity transfer function does not result in a binary classification (so that high frequencies are not introduced) then the bilinear filter produces good results. Synthetic data sets that have not been properly antialiased do not always satisfy these conditions and may result in renderings with artifacts. Real-world sampled data generally does satisfy these conditions as long as binary classification is not used, and as shown in the figures the resulting images are of comparable quality to those produced with trilinear interpolation. Binary classification could be supported by band-limiting the volume after classification, i.e. blurring the opacities.

Finally, a third limitation of the shear-warp rendering algorithms is that the resolution of the input volume fixes the sampling rate during rendering, regardless of the image resolution. If the image resolution is much greater than the volume resolution then the image will be either blurry or blocky, depending upon the filter used for the 2D warp. The quality of the rendering could be improved by supersampling each slice of voxel data as it is transformed into sheared object space. However the increased algorithmic complexity caused by the supersampling would result in reduced performance, and it is impossible to supersample in the dimension perpendicular to the voxel slices without significant changes to the algorithm. A better alternative is to prescale the volume by resampling it to a higher-resolution grid before rendering. A high-quality filter can be used to prescale the volume since the cost would be incurred only once instead of during each rendering. The main disadvantage of prescaling is that the upsampled volume requires more memory.

The volume should also be prescaled if the voxel spacing in the original volume is not isotropic (i.e. the spacing between samples is not the same in all three dimensions). Otherwise the sampling rate in image space will be non-uniform, potentially resulting in either aliasing artifacts or an excessively high voxel sampling rate in some parts of the image. Ray casters do not have this problem because the sample locations along a viewing ray can be distributed uniformly, independently of the volume resolution.

The renderings in this chapter show that the three filtering limitations do not have a significant effect on the quality of still images. However, subtle aliasing artifacts in a still image can become more apparent in movie sequences: pixels scintillate if their brightness changes from frame to frame. We have used the shear-warp algorithm to generate animations from the data sets shown in Figures 5.5–5.7 and we found no objectionable artifacts.

The shear-warp rendering algorithms produce smooth rotation sequences without noticeable aliasing. Furthermore, the image sequences are continuous even when the principal viewing axis changes.

5.2 Comparison of Coherence-Accelerated Volume Rendering Algorithms

The shear-warp algorithms achieve substantially improved performance compared to existing volume rendering algorithms. The primary reason for the performance gains is that the shear-warp algorithm traverses the coherence data structures and the volume data in scanline order, resulting in low computational overhead. In contrast, most existing high-quality coherence-accelerated volume rendering algorithms are based on ray casting. In the remainder of this chapter we use an octree-accelerated ray caster as a representative example of existing methods and compare it to the shear-warp algorithms. The comparison illustrates the performance differences between scanline-order algorithms and ray casters.

Our experiments use the optimized ray casting algorithm described by Levoy [1990a] augmented with the template optimization described by Yagel & Kaufman [1992]. The algorithm uses a full octree (a pyramid) to encode a classified volume. Each node in the octree includes a flag indicating whether the node or any of its children contain at least one non-transparent voxel. During ray casting the algorithm uses the information in the octree to determine the largest step that can be taken along a viewing ray before reaching the next non-transparent voxel. When the algorithm finds a non-transparent voxel it enters a standard ray casting loop that steps along the ray at fixed intervals, sampling the volume and compositing the samples together, until a transparent voxel is reached. Then the algorithm again uses the octree to find the next non-transparent voxel. The template optimization is used to reduce the costs of tracing rays and computing resampling weights as described in Section 3.5. The algorithm also uses the early ray termination optimization.

5.2.1 Asymptotic Complexity

Brute-force volume rendering algorithms operate by resampling and compositing all of the voxels in a volume. Two examples are the shear-warp algorithm without run-length encoded data structures and the ray casting algorithm without an octree or early ray termination. In these algorithms the work per voxel is constant, so assuming that the volume contains n^3 voxels (n voxels per side) and the image contains n^2 pixels, and that the ray caster casts one ray per pixel with n samples along the ray, the asymptotic complexity of the brute-force algorithms is $O(n^3)$ operations.

Coherence optimizations reduce the number of voxels processed. For some classes of volumes containing significant coherence, e.g. scenes consisting of voxelized surfaces, these optimizations also reduce the asymptotic complexity. However, accesses to the coherence data structures also have a cost. This section analyzes the complexity of an octree-accelerated ray caster, the shear-warp algorithm with run-length encoded data structures, and the shear-warp algorithm with a min-max octree data structure.

Complexity of Octree-Accelerated Ray Casting The octree data structure in the coherence-accelerated ray caster allows the algorithm to avoid shading, resampling and compositing the transparent voxels in the volume. In the worst case all voxels in the volume have a small but non-negligible opacity such that they all contribute to the image. In that case neither the octree nor early ray termination helps, so the worst-case asymptotic complexity of the rendering algorithm is still $O(n^3)$ operations.

However, typical classified volumes have structure that can be exploited with the octree. In practice a common case is a scene consisting of surfaces, some of which may be transparent or fuzzy. For instance, doctors and engineers who use CT and MR scanners to examine physical objects are generally interested in the boundaries between different tissues or materials, so they often choose a classification function that exposes these surfaces. The octree can efficiently encode the large regions of transparent space lying between individual surfaces.

Suppose the classified scene consists of a collection of opaque surfaces. In this case, the number of visible voxels equals the projected surface area of the visible surfaces (at most n^2 voxels) multiplied by the average thickness t of the surfaces. The thickness of a voxelized surface depends only on the support of the band-limiting filter used when creating the volume, not on the resolution of the volume (since the true surface is infinitesimally thin). In the case of acquired data the characteristics of the measurement equipment determine the support of the band-limiting filter.

If the scene contains semi-transparent surfaces then the maximum number of visible voxels increases by a factor d which is the depth complexity of the scene (the average number of surfaces intersected by a viewing ray before the maximum ray opacity threshold is reached). The depth complexity is also independent of the resolution of the volume. Thus in the case of a classified volume representing either opaque or transparent surfaces, the maximum number of visible voxels is dtn^2 (where $d = 1$ for strictly opaque surfaces). The ray casting algorithm with an octree processes only the visible voxels, so the asymptotic complexity of resampling and compositing the volume is $O(dtn^2)$ operations.

The octree-accelerated ray caster also incurs costs due to the optimizations. The algorithm implements early ray termination by checking the accumulated ray opacity every time a voxel is composited, so the number of checks is dtn^2. Descending the octree to find the intersection of a ray with a surface incurs a cost of $O(\log_2 n)$ operations. If the number of surfaces is much smaller than the number of voxels, $d \ll n$, then the cost to find all surfaces intersecting a ray is $O(d \log_2 n)$ (although if $d \approx n$ then the cost of descending the tree can be amortized over many surfaces and the total work is $O(n)$). Finally, there is an additional cost due to grazing rays which pass near a surface. For these rays the rendering algorithm must still descend the octree in the vicinity of a surface even though the ray does not intersect the surface. It is difficult to analyze this cost rigorously since it depends on how smooth or convoluted the surfaces are (which is related to depth complexity). In practice our measurements have shown that the cost of grazing rays roughly doubles the total cost of casting rays through the octree. We therefore conclude that the overall asymptotic complexity of the optimized ray casting algorithm is:

$$O(dn^2 \log_2 n + dtn^2)$$

This result is a significant improvement over the $O(n^3)$ brute force algorithm.

Complexity of the Shear-Warp Algorithm The shear-warp rendering algorithms also process only the non-transparent, non-occluded voxels. Just as for the optimized ray caster, in the case of a scene containing surfaces the cost of resampling and compositing the volume is $O(dtn^2)$ operations. The shear-warp algorithm has four additional costs. It must warp the 2D image, costing $O(n^2)$ operations, and there are costs incurred by the coherence data structures: traversing voxel runs, and performing the UNION-FIND operations associated with the image pixel runs.

The cost of traversing voxel runs is $c_v n^2$ operations, where n^2 is the number of voxel scanlines in the volume and c_v is the average number of runs per scanline. Unlike a ray caster the shear-warp rendering algorithm always traverses every run in the volume, although it does not process individual voxels in the transparent runs. If the volume consists of surfaces then the number of runs per scanline is proportional to the number of surfaces (which is also an upper bound on the depth complexity).

The cost of traversing and maintaining the opaque pixel runs depends on the number of times a new run is created (n_c) and the number FIND operations (n_f) as described in Section 4.1.3. Consider an intermediate image scanline containing n pixels. On average, the number of voxels composited into the image scanline over the course of rendering the entire image equals the average number of surfaces per scanline times the average thickness of each surface, which equals $\frac{1}{2}c_v t$. This number is an upper bound for the number of opaque pixel runs created (n_c). An upper bound for the number of FIND operations (n_f) is the maximum number of opaque pixel runs times the number of times the image scanline is traversed, which equals $\frac{1}{2}c_v tn$. The cost of the UNION-FIND operations for a single intermediate image scanline is $O(n_f \log_{1+n_f/n_c} n_c)$ [Cormen et al. 1990]. Thus the total cost for all n scanlines in the image is $O(c_v tn^2 \log_{1+n_f/n_c} c_v t)$.

The result of this analysis is the following worst-case asymptotic complexity for the shear-warp algorithm, under the assumption that the scene consists of opaque or semi-transparent surfaces:

$$O([dt + c_v t \log_{1+n_f/n_c} c_v t]n^2)$$

In practice we observe execution time growth rates consistent with this analysis as we saw in Table 5.3: an eight-fold increase in the number of voxels leads to only a four-fold increase in time for the compositing stage of the algorithm and just under a four-fold increase in overall rendering time.

To place the relative importance of each term into context, for the CT head data set (Figure 5.2) the rough breakdown of computational costs in our implementation is: 40% for resampling and compositing voxels, 30% for traversing voxel runs, 10% for the 2D warp, and 5% for the UNION-FIND operations. These are percentages of computational costs only, ignoring memory overhead. We will analyze the measured results later in this chapter. For this data set, $n_f/n_c = 2.5$, $c_v = 5.8$, $t = 2.4$, and $d \approx 1$ since most surfaces are opaque.

The n^2 growth rate of the shear-warp algorithm is better than the $n^2 \log_2 n$ growth rate of the octree-accelerated ray caster. However, an improvement by a factor of $\log_2 n$ does not result in a large performance gain. The more important improvement is that the cost polynomial for the shear-warp algorithm also has lower constant factors. Experimental results later in this chapter validate this claim.

Complexity of the Fast Classification Algorithm The fast classification algorithm has the same costs as the basic shear-warp rendering algorithm analyzed in the previous section, plus additional costs to compute the summed-area table and traverse the min-max octree. Computing the summed-area table requires one operation for every combination of arguments to the opacity transfer function. If the function has p parameters each of which can take on m values then the computational complexity for computing the table is m^p operations. Typically the number of parameters p equals one or two, and the value of each parameter is stored in an eight-bit byte ($m = 256$).

The cost for one traversal of the min-max octree (to compute the run lengths for one voxel scanline) is $O(c_v t \log_2 n)$ operations assuming that the number of non-transparent voxels runs per scanline (c_v) is much less than n, just as we argued for the octree-accelerated ray caster. Since there are n^2 voxel scanlines the cost of the octree traversal for all scanlines is $O(c_v t n^2 \log_2 n)$. Adding up all of the terms, the asymptotic complexity of the fast classification algorithm is:

$$O([dt + c_v t \log_{1+n_f/n_c} c_v t]n^2 + c_v t n^2 \log_2 n + m^p)$$

In our implementation the cost of classifying, resampling and compositing the non-transparent voxels dominates (70% of the computation required to compute Figure 5.2). The octree traversal and summed-area-table initialization account for 10% of the computational costs.

5.2.2 Experimental Methodology

Operation counts and asymptotic complexities do not give a complete picture of an algorithm's performance. In the case of optimized volume rendering algorithms the constant factors associated with accessing coherence data structures and executing control logic are the dominant influences on performance. The only way to determine these costs is to measure the execution times of real implementations. The disadvantage of comparing algorithms by measuring execution times is that the relative performance depends on the specific implementations and on how carefully the implementations have been tuned. However, we are still able to draw some general conclusions from the results.

Implementation of the Algorithms The experiments in this section compare the ray caster and the shear-warp rendering algorithm for parallel projections. The results include timings for both brute-force versions and optimized versions of the algorithms. The optimizations fall into two classes: optimizations that reduce the number of voxels processed, and optimizations that reduce the cost per voxel actually processed. The former category includes early ray termination and coherence optimizations based on spatial data structures. The optimizations in the latter category have been chosen so that the ray caster and the shear-warp algorithm have nearly identical costs per voxel. These optimizations include:

filter kernel Different filter kernels have different costs. We use the ray caster to compare the costs of trilinear interpolation and bilinear interpolation. When bilinear interpolation is enabled the ray caster restricts each sample point to lie in the plane of a voxel slice, just like the shear-warp algorithm.

template-based ray casting Yagel's template algorithm [Yagel & Kaufman 1992, Yagel & Ciula 1994] (discussed in Section 3.5) uses the shear-warp factorization to reduce the cost of computing sample locations and filter weights in the ray casting algorithm.

With bilinear interpolation and the template optimization in the ray caster the cost of resampling is nearly identical to the cost of resampling in the shear-warp algorithm.

resampled opacity threshold When this optimization is enabled, if the opacity of a resampled voxel falls below a user-defined threshold then the renderer does not composite the voxel into the image. Both algorithms use this optimization to reduce the number of unnecessary compositing operations.

The performance results later in this section show that the cost-per-voxel optimizations are important when spatial data structures are not used, but are not nearly as important in the coherence-accelerated algorithms.

To allow for a fair comparison both implementations have been tuned at the source code level for the architecture on which the tests were run (an SGI Indigo2 R4400-based workstation). Except where otherwise noted, we show results for the shear-warp algorithm with run-length encoded data structures. The fast classification algorithm and the octree-accelerated ray casting algorithm are not directly comparable since the latter is not classification-independent.

Timing Tests Our experiments measure both the wall clock execution time of each algorithm on a set of benchmark data sets and a breakdown of the execution times into categories. The categories have been chosen to highlight the primary sources of overhead in each algorithm. The categories include:

resampling and compositing Time spent resampling voxels and compositing them into the image (or intermediate image). This work is essential to compute the image; all other categories represent overhead.

looping Time spent on control overhead (updating loop counters, advancing pointers, etc.) and traversing coherence data structures while searching for the next voxel to process.

early ray termination Time spent updating data structures required for early ray termination in the shear-warp algorithm. Early-ray termination has negligible cost in a ray caster.

fixed costs Miscellaneous fixed costs (costs that are independent of the size of the volume) required to initialize data structures and precompute lookup tables. For the shear-warp algorithm and the template-based ray caster this category includes the cost of warping the intermediate image into the final image.

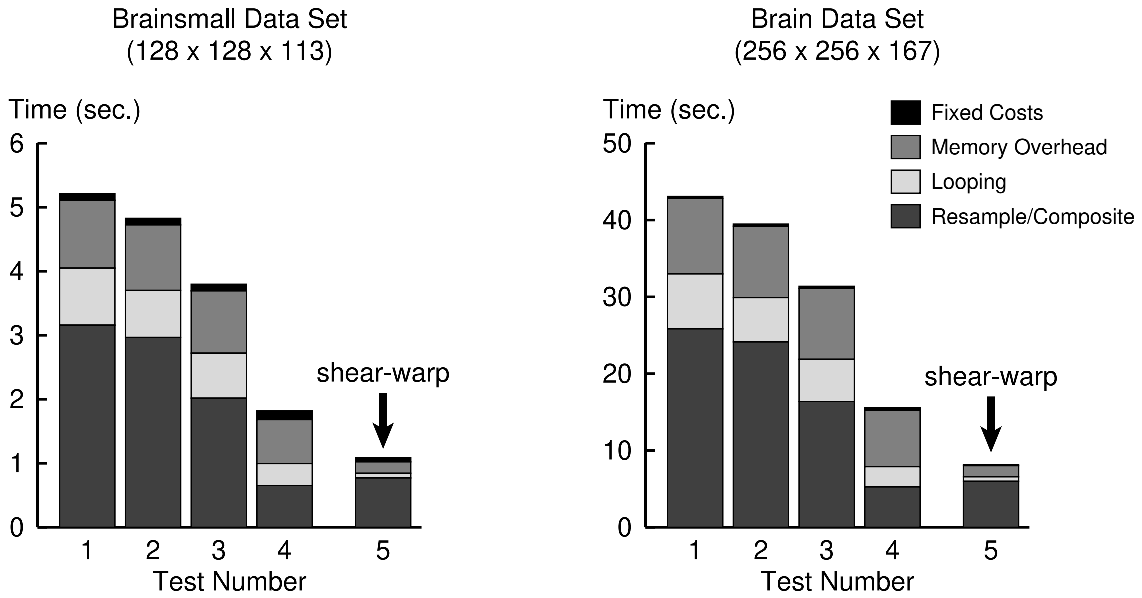
memory overhead Stall time associated with memory references. The other categories include computation time and processor stalls due to hardware interlocks (such as instruction dependencies) but not memory stall time.

Most of these cost categories involve operations that are implemented with short code fragments. Interval timers cannot be used to time such short code fragments since the overhead of querying the timer would perturb the results. Instead, the method we use to measure execution times is based on basic-block counting. First, we instrument the volume rendering program's binary executable to count the number of times each basic block is executed (using the Pixie program [Smith 1991]). We then run the instrumented program and use a post-processor to tabulate the number of times each instruction was executed along with the number of cycles the instruction took to execute (including pipeline stalls). Using information in the program's symbol table each instruction can be attributed to an individual source file line, and thus the instruction cycle counts can be grouped into the categories in the list above. Finally, given the speed of the processor, the cycle counts can be converted into execution times.

The basic-block timing method accounts for all costs in the program except for memory stalls. To measure memory overhead we run the uninstrumented program with the same parameters as the instrumented program on an otherwise-idle machine and measure the total execution time of the program with an interval timer. The difference between the total time and the time measured by basic block counting gives the time spent on memory overhead. We have automated the entire process of collecting cycle counts, attributing cycle counts to categories, and computing memory overhead so that data collection is easy and reliable.

5.2.3 Comparison of Speedups from Algorithmic Optimizations

Figure 5.11 gives timing breakdowns for the ray caster and the shear-warp algorithm without coherence optimizations or early ray termination. Since the coherence optimizations have



Test Number	1	2	3	4	5
Renderer	ray caster	ray caster	ray caster	ray caster	shear-warp
Opacity threshold	none	0.05	0.05	0.05	0.05
Filter	trilinear	trilinear	bilinear	bilinear	bilinear
Template	no	no	no	yes	N/A
Time (sec.) brainsmall	5.2	4.8	3.8	1.8	1.1
Time (sec.) brain	43	39	31	16	8

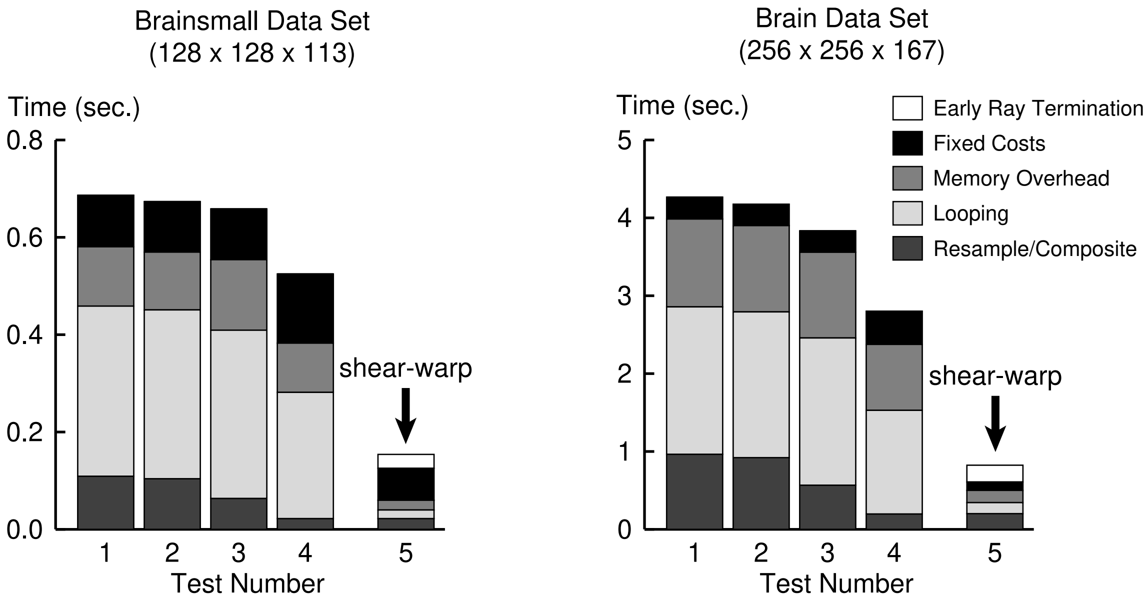
Figure 5.11: Timing breakdowns for the ray casting algorithm and the shear-warp algorithm without coherence optimizations.

been disabled the results are not view-dependent or data-dependent, but timings are shown for two different data set sizes. Each bar is broken down into components corresponding to the cost categories. For the ray caster the figure shows a sequence of measurements starting on the left with the most basic brute-force algorithm and progressively adding the cost-per-voxel optimizations. For the shear-warp algorithm the figure shows a single measurement (Test 5) with all of the cost-per-voxel optimizations.

These results show that when coherence optimizations are not used the dominant computational cost comes from resampling and compositing voxels. As a result the cost-per-voxel optimizations significantly reduce execution time. Using a bilinear interpolation filter rather than a trilinear filter results in a speedup of 1.3x, and the template optimization results in an additional speedup of 2.0x (slightly better than the results reported in Yagel & Ciula [1994]).

With these two optimizations the computation time for the ray caster (Test 4) is within 15% of the computation time for the shear-warp algorithm (Test 5), so the cost per voxel in the two algorithms is nearly identical. This fact will make it easier to isolate the effects of the coherence data structures in the fully-optimized algorithms. However the volume traversal order is different for the two algorithms. The ray caster does not traverse the volume in storage order so it has higher memory overhead than the shear-warp algorithm, resulting in the difference in overall execution time. Memory overhead will be discussed later in this chapter. The looping overhead is also slightly higher in the ray caster, although compared to the total rendering time the difference is not significant enough to merit investigation.

Figure 5.12 gives timing breakdowns for the ray caster and the shear-warp algorithm with both coherence optimizations and early ray termination. These timings were measured for a single representative viewpoint (Figures 5.1 and 5.2). The figure shows the same progression of cost-per-voxel optimizations as in Figure 5.11. These results show that resampling and compositing voxels is no longer the dominant cost. Looping (including traversal of the spatial data structures) is more significant, and in the case of the ray caster it accounts for the overwhelming majority of the computational costs. As a result the cost-per-voxel optimizations make very little difference. On the other hand, comparing the results for the brute-force algorithms and the coherence-accelerated algorithms shows that the coherence data structures provide speedups of roughly an order of magnitude as previously reported in Levoy [1990a].



Test Number	1	2	3	4	5
Renderer	ray caster	ray caster	ray caster	ray caster	shear-warp
Opacity threshold	none	0.05	0.05	0.05	0.05
Filter	trilinear	trilinear	bilinear	bilinear	bilinear
Template	no	no	no	yes	N/A
Time (sec.) brainsmall	0.69	0.67	0.66	0.52	0.15
Time (sec.) brain	4.3	4.2	3.8	2.8	0.82

Figure 5.12: Timing breakdowns for the ray casting algorithm with an octree and early ray termination, and for the shear-warp algorithm with run-length encoding.

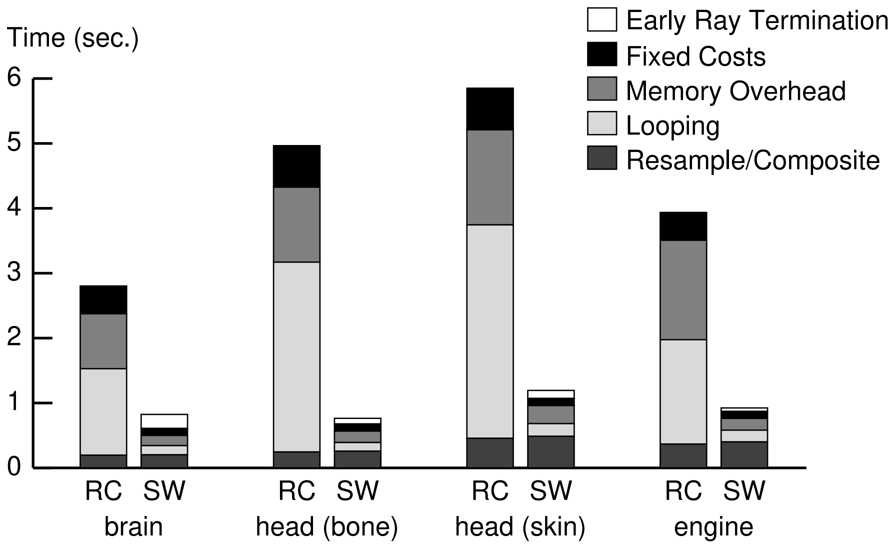


Figure 5.13: Comparison of timing breakdowns for several data sets. The left bar of each pair shows results for the coherence-accelerated ray casting algorithm with the template optimization and bilinear interpolation, and the right bar shows results for the coherence-accelerated shear-warp algorithm. All renderings are gray scale.

Figure 5.12 also shows that the cost of the early ray termination implementations is small. For the ray caster the cost is so small it is difficult to measure reliably and does not show on the scale of the figure. For the shear-warp algorithm the cost is slightly higher. We will discuss this cost in the next section.

Since the coherence optimizations are data-dependent these results depend on the specific data set. Figure 5.13 shows results for the larger MR brain data set and three other data sets: the 256x256x225 CT head data set classified to reveal bone (Figure 5.2), the same data set classified to reveal semitransparent skin and bone (a gray scale version of Figure 5.5), and the 256x256x110 CT engine data set (a gray scale version of Figure 5.6). This figure shows results only for the fully-optimized versions of the ray casting algorithm and the shear-warp algorithm. Again, the cost of processing voxels accounts for a small fraction of rendering time.

Finally, Table 5.5 shows the speedups achieved by the fully-optimized shear-warp algorithm relative to two versions of the octree-accelerated ray caster, one with trilinear interpolation and the other with bilinear interpolation and the template optimization. These results

Data set	Shear-warp	Ray caster w/ octree and trilirp		Ray caster w/ octree, bilirp and template	
	Time (msec.)	Time (msec.)	Shear-warp speedup	Time (msec.)	Shear-warp speedup
brainsmall	154	622	4.0	524	3.4
headsmall	192	1201	6.3	990	5.2
brain	824	4124	5.0	2803	3.4
head (bone)	764	9459	12.4	4962	6.5
head (skin)	1194	9100	7.6	5849	4.9
engine	924	7617	8.2	3935	4.3

Table 5.5: Speedup of the shear-warp algorithm relative to an octree-accelerated ray caster. The table includes results for two versions of the ray caster: a standard ray caster with trilinear interpolation, and a ray caster with bilinear interpolation and the template optimization.

are for the same data sets used in Figure 5.13. Relative to the standard ray caster the shear-warp algorithm achieves speedups of 4-12x, and relative to the template-based ray caster the speedups are 3-6x.

The results in this section have shown that cost-per-voxel optimizations are not important in coherence-accelerated volume rendering algorithms. The results have also shown that the dominant cost in the brute-force algorithms is resampling and compositing voxels, but in the coherence-accelerated algorithms the overhead of looping and accessing the data structures is more important. The next section compares this overhead for the shear-warp algorithm and the ray caster in more detail.

5.2.4 Costs of Coherence Accelerations

Table 5.6 contains timing information in tabular form (but otherwise identical to Figure 5.13) for two of the data sets and shows the percentage of total execution time for each category of overhead.

The most important observation from this data is that the ray caster spends far more time looping than does the shear-warp algorithm. For the ray caster 70% of the time in the "looping" category comes from traversing the octree, and the remaining 30% comes from overheads in the loop that iterates over a span of non-transparent voxels within an octree leaf

Data Set	brain		head (skin)	
Algorithm	Ray Caster	Shear-Warp	Ray Caster	Shear Warp
Total Time (msec.)	2802	824	5850	1194
Early Ray Termination	0 (0%)	216 (26%)	0 (0%)	124 (10%)
Fixed Costs	427 (15%)	108 (13%)	641 (10%)	110 (9%)
Memory Overhead	847 (30%)	157 (19%)	1465 (25%)	277 (23%)
Looping	1332 (47%)	142 (17%)	3288 (56%)	196 (16%)
Resample/Composite	197 (7%)	202 (24%)	457 (7%)	487 (40%)

Table 5.6: Detailed timing breakdowns for coherence-accelerated rendering algorithms (same data as in Figure 5.13). These results are for a ray caster with the template optimization and bilinear interpolation.

node. The source of the overhead in the octree traversal comes from the complexity of computing the intersection of an arbitrarily-oriented ray with the axis-aligned sides of the octree nodes.

Figure 5.14 is a pseudo-code description of the algorithm for tracing a ray through an octree. The algorithm begins by finding the first intersection of the ray with the top-level node of the octree. The loop on line 5 descends to the lowest non-transparent octree node containing the current sample point. The notation “Octree[level, x, y, z]” refers to the octree node at the specified level that contains the specified voxel coordinates. The call to *NextIntersection* calculates the intersection of the ray with the far side of the octree node. The algorithm then renders samples along the ray up to the intersection point. The loop on line 11 ascends back up the octree until the current octree node and the next node intersected by the ray both have the same parent (so that the next call to *RenderSegment* can take the longest possible step). Finally, the algorithm moves to the next octree node along the ray and repeats until the ray exits the volume (or until the accumulated ray opacity reaches the early ray termination threshold, although this test is not shown).

This algorithm contains a substantial amount of control logic (loops and conditionals), and the *NextIntersection* routine must compute the intersection of the ray with each of the three possible cube faces it might hit. Floating point arithmetic must be used to compute the intersection points in order to maintain a consistent sample spacing and to correctly compute resampling weights (unless the rendering algorithm uses a nearest-neighbor filter). These costs are inherent in an octree-accelerated ray caster. In contrast, the shear-warp algorithm

```
procedure TraceRay()
    level = top_level;
    (x, y, z) = FirstIntersection();
    while ((x, y, z) ∈ Volume)
        while (level < bot_level and Octree[level, x, y, z] ≠ transparent)
            level = level + 1;
        end
        (dx, dy, dz) = NextIntersection(level, x, y, z);
        if (Octree[level, x, y, z] ≠ empty)
            RenderSegment(x, y, z, dx, dy, dz);
        while (level > top_level and
                  Parent(level, x, y, z) ≠ Parent(level, x+dx, y+dy, z+dy))
            level = level – 1;
        end
        (x, y, z) = (x, y, z) + (dx, dy, dz);
    end
end
```

Figure 5.14: Pseudo-code for tracing a ray through an octree.

simply loads a run length and increments a pointer to advance to the next non-transparent voxel.

The ray caster has a second extra cost: the cost of tracing rays that graze but do not intersect surfaces in the volume. A ray that passes close to a non-transparent object causes the ray caster to descend to the leaf nodes of the octree, only to determine that the ray does not intersect the object. We have performed measurements indicating that the cost of grazing rays amounts to half of the total cost of looping in the ray caster. The shear-warp algorithm with a run-length encoded volume never wastes time sampling transparent voxels so it does not incur any additional cost for grazing rays.

Most other coherence-accelerated volume rendering algorithms also trace rays through an octree [Meagher 1982, Wilhelms & Van Gelder 1992, Danskin & Hanrahan 1992, Stander & Hart 1994] or a similar data structure such as a k-d tree (constructed using a median-cut algorithm to partition the volume with axis-aligned planes) [Subramanian & Fussell 1990]. These algorithms all have overheads due to the tree traversal. The fast classification algorithm proposed in Chapter 4 does not suffer from these overheads even though it uses an octree because the algorithm always traverses the data structure along axis-aligned rays, so the traversal algorithm is more efficient.

Zuiderveld proposes an alternative ray casting algorithm that uses distance transforms instead of a hierarchical data structure [Zuiderveld et al. 1992, Zuiderveld 1995]. During a preprocessing step the algorithm computes the distance from each voxel to the nearest non-transparent voxel. The rendering algorithm uses the distance stored in a voxel at a particular sample point to determine the longest step size before the next interesting voxel. Zuiderveld shows that the computational overhead required to access the distance array and increment the sample location is less than the overhead of traversing an octree.

However, the algorithm has two disadvantages. First, the algorithm requires enough memory to store the entire volume and the distance array. Second, the distance transform only produces a conservative approximation of the distance to the next sample point. A ray caster using the distance transform steps by shorter and shorter distances as it approaches a non-transparent voxel, and at each step it must sample the volume to determine if the adjacent voxels are transparent (similar to the grazing-ray costs in the octree-accelerated ray caster). The shear-warp algorithm is more memory-efficient and spends less time searching for non-transparent voxels.

Returning to the table of timing breakdowns (Table 5.6), we see that the cost of early ray termination in the shear-warp algorithm is small but non-negligible. About 60-80% of the time in the "early ray termination" category is due to traversing the run-length encoded voxel scanlines when skipping occluded voxels. The individual voxels need not be visited, but the algorithm must step through the runs in sequence to reach the next non-occluded voxel. In a ray caster the octree nodes associated with occluded voxels are never visited, so there is no corresponding overhead. The remaining 20-40% of the time is spent traversing and coalescing the opaque image pixel links (Section 4.1.3).

The fixed costs for the ray caster are somewhat higher than for the shear-warp algorithm. Since this category accounts for only 10-15% of overall execution time the differences are not significant, but most of the increase is due to initializing the ray template and performing clipping calculations for each ray cast into the volume.

The final cost category is memory overhead, which we consider in the next section.

5.2.5 Memory Overhead

To demonstrate the impact of memory overhead we performed an experiment in which we rotated the CT head volume over a 90 degree range of angles and rendered it with both the ray caster and the shear-warp algorithm. We performed the experiment with both the brute-force and coherence-accelerated algorithms. We chose the viewpoint such that at the initial angle the viewing rays coincided with the direction of the voxel scanlines in the data structures used by the ray caster, and at the final angle the viewing rays were perpendicular to the voxel scanlines.

When the viewing rays line up with the voxel scanlines the ray caster benefits from good spatial locality, meaning that when the algorithm references a voxel the voxels immediately adjacent in memory will soon be referenced. When the viewing rays are perpendicular to the voxel scanlines there is no spatial locality. However, in the experiment we cast the rays in a sequence that maximizes temporal locality, meaning that when the algorithm references a voxel the same voxel may be referenced again soon. The algorithm does this by casting adjacent rays in sequence so that each ray reuses some of the same voxels as the previous ray during resampling.

Spatial and temporal locality improve memory performance because the cost of fetching a cache line from memory can be amortized over several voxel accesses, whereas an algorithm that references memory randomly causes a cache miss for every reference [Hennessy & Patterson 1990]. The shear-warp algorithm always benefits from good spatial locality because it traverses the volume in scanline order regardless of the viewing direction.

Figure 5.15 shows the results of the experiment for the brute force algorithms, and Figure 5.16 shows the results for the coherence accelerated algorithms. The timings were performed on an SGI Indigo2 workstation with a 16 Kbyte direct-mapped primary cache with 16 byte lines and a 1 Mbyte two-way associative secondary cache with 128 byte lines. The long cache lines in the secondary cache result in a significant benefit from spatial locality.

The brute-force shear-warp algorithm has constant memory overhead of only 15% for the entire range of angles. For the brute force ray caster the memory overhead increases by a factor of 2.5x as the viewing angle changes from 0 to 90 degrees, causing a 1.25x increase in rendering time. Because of the long secondary cache line the benefit of spatial locality

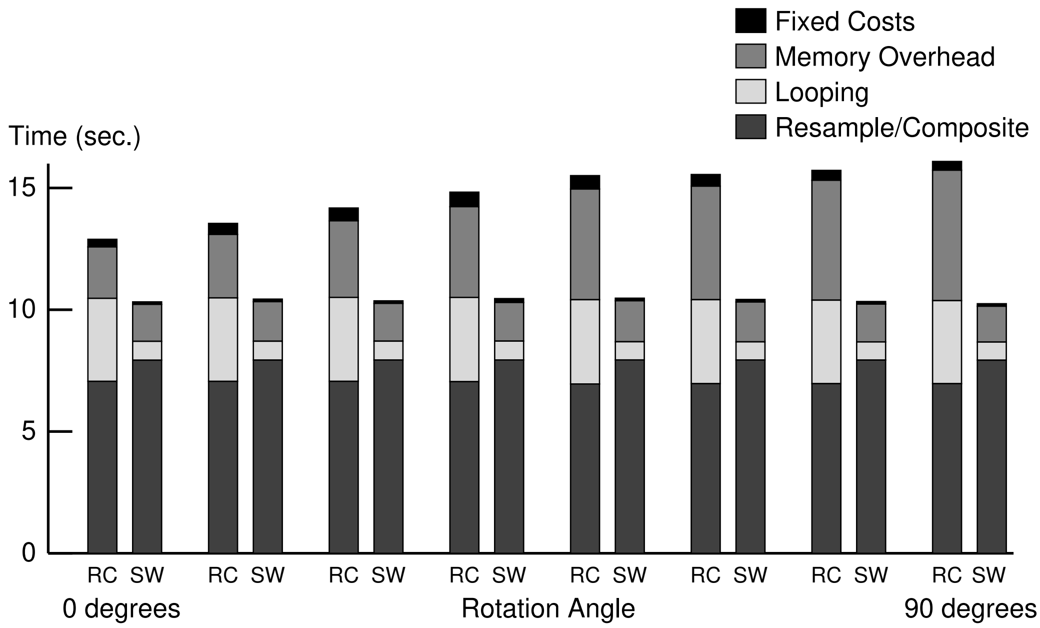


Figure 5.15: Comparison of memory overhead in the ray caster and the shear-warp algorithm without coherence optimizations. The ray caster uses the template optimization and a bilinear interpolation filter. The data set is the 256 x 256 x 225 voxel CT head. As the viewpoint changes over a 90 degree range of angles the memory overhead increases significantly for the ray caster but remains small for the shear-warp algorithm.

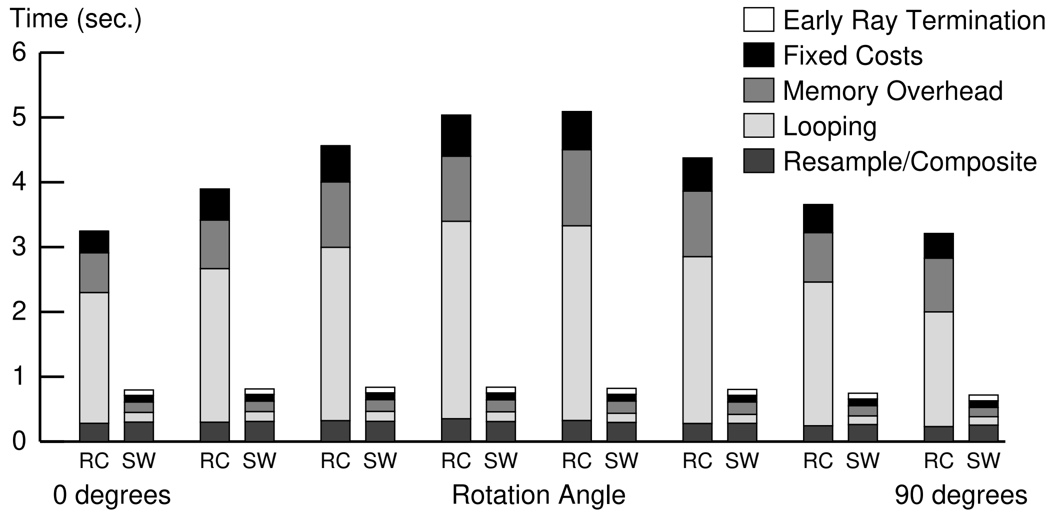


Figure 5.16: Comparison of memory overhead in the coherence-accelerated ray caster and the shear-warp algorithm. Memory overhead does not vary significantly as the viewpoint changes, although the overhead incurred by the octree does change.

is greater than the benefit of temporal locality. However, temporal locality still provides a significant benefit: when the rays are cast in a different order to eliminate the temporal locality the memory overhead increases by 10x relative to the case with optimal spatial locality, and the overall rendering time increases by 2.6x. The current trend in processor design is towards longer cache lines to cope with the increasing gap between processor speeds and memory latencies, so algorithms that benefit from spatial locality are important.

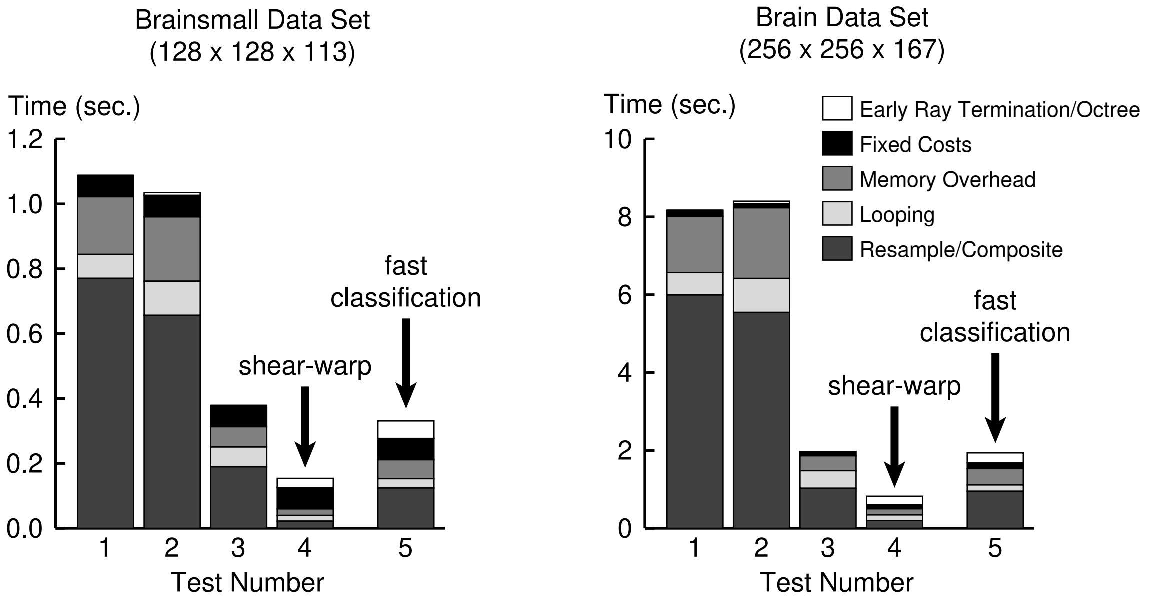
Figure 5.16 shows that for the coherence-accelerated algorithms memory overhead is not so significant: 20% of execution time for both algorithms. Most of the time in these algorithms is spent traversing data structures rather than processing voxels, so loading voxels from memory is a virtually insignificant overhead. The variations in memory overhead are small and are caused by variations in the accesses to the spatial data structures rather than the voxel array. Most likely the good memory performance in the ray caster is due to temporal locality when accessing the octree since many adjacent rays traverse the same octree nodes.

We conclude that when coherence optimizations are disabled or do not work effectively (due to low-coherence data sets) the shear-warp algorithm has lower memory overhead than the ray caster. In practice, the coherence optimizations are effective and both algorithms have excellent memory behavior.

5.2.6 Analysis of the Shear-Warp Coherence Data Structures

Figure 5.17 demonstrates the impact of each of the data structures used in the shear-warp algorithms. In the table the row labeled "Samples" gives the number of resampling operations, the row labeled "Time" gives the total time in seconds, and the row labeled "Speedup" gives the speedup relative to the brute-force algorithm (Test 1).

The left column gives results for the brute-force algorithm without early ray termination or a spatial data structure. The second column is for the brute-force algorithm with early ray termination enabled. The optimization has virtually no impact by itself because for this data set most of the transparent voxels are not occluded, so a majority of the voxels must be processed. The third column shows the effect of using a run-length encoded volume without early ray termination. This optimization yields a 3-4x speedup. If early ray termination is



Test Number		Render Only 1	2	3	4	Classify 5
Spatial Data Structure		none	none	run-length encoding	run-length encoding	min-max octree
Early Ray Termination		no	yes	no	yes	yes
brainsmall	Samples	1,397,844	1,021,395	301,138	32,495	90,391
	Time (sec.)	1.1	1.0	0.38	0.15	0.33
	Speedup	1.0	1.1	2.9	7.1	3.3
brain	Samples	11,030,183	8,556,478	1,760,992	305,127	629,080
	Time (sec.)	8.2	8.4	2.0	0.82	1.94
	Speedup	1.0	0.97	4.1	9.9	4.2

Figure 5.17: Impact of individual coherence optimizations in the shear-warp algorithm.

also enabled (Test 4) then an order-of-magnitude speedup is achieved. Early ray termination helps in this case because one-half to one-third of the non-transparent voxels are occluded, so a significant fraction can be culled.

The fifth column shows results for the fast classification algorithm using the min-max octree, the summed-area table and early ray termination. As shown earlier in Table 5.3 this algorithm is about twice as slow as the algorithm based on a precomputed run-length encoded volume. There are two reasons for the increase in time: the renderer samples more voxels, and it performs more work for each sample since the voxels must be classified. The increase in the number of samples occurs because the min-max octree and summed-area table data structures produce a conservative estimate of the regions containing non-transparent voxels.

5.3 Low-Coherence Volumes

5.3.1 Categories of Volume Data

The classification functions we used in the previous sections resulted in volumes with a high degree of coherence and voxels with relatively high opacity, so both of the optimization techniques in the shear-warp algorithm (coherence acceleration and early ray termination) were effective. However, there are other types of volumes for which the optimizations will fail.

The different types of classified volumes can be categorized along three axes (Figure 5.18):

1. The *amount of coherence in the volume*, which is measured by counting the number of runs in a run-length encoded representation of the classified volume. If there are only a few runs per voxel scanline then the volume contains a lot of coherence, whereas if each voxel scanline has many short runs of transparent voxels interspersed with short runs of non-transparent voxels then the volume has low coherence.

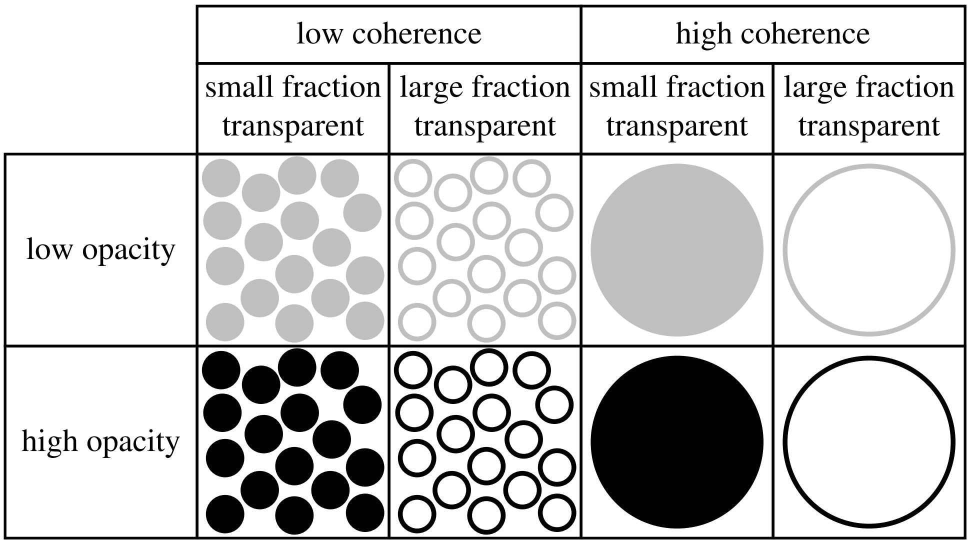


Figure 5.18: Classified volume data can be categorized along three axes: the amount of coherence in the volume (measured by the number of runs in a run-length encoded representation), the fraction of voxels that are transparent, and the average opacity of the non-transparent voxels. The diagram shows schematic cross-sections of volumes at eight points within the categorization.

2. The *fraction of voxels that are transparent*. Given sufficient coherence, these voxels can be skipped efficiently using a spatial data structure.

3. The *average opacity of the non-transparent voxels*. This quantity determines the effectiveness of early ray termination. The higher the average opacity, the earlier each pixel becomes opaque.

This categorization can be used to make first-order predictions about rendering performance. In the remainder of this section we discuss performance qualitatively, and in the next two sections we present some quantitative results for our implementation of the shear-warp algorithm.

There are four performance regimes. In the first regime neither of the optimizations is effective. Early ray termination is ineffective if the average opacity of the non-transparent voxels is low, since few (if any) image pixels become opaque. Spatial data structures may be ineffective for either of two reasons. First, if there is little coherence then the rendering algorithm cannot skip over large regions of the volume, so the spatial data structure cannot reduce the time spent traversing the volume. Second, if the volume does have high coherence but most of the voxels are non-transparent then the rendering algorithm must still traverse most of the volume. These situations correspond to the first three cases in the top row of Figure 5.18. In this regime the rendering time is proportional to the number of voxels in the data set, and the optimized shear-warp algorithm cannot outperform a brute force algorithm.

In the second performance regime early ray termination is ineffective but spatial data structures do improve performance. This regime applies to volumes that have low average opacity, high coherence, and a large fraction of transparent voxels (the last case in the top row of Figure 5.18). The rendering algorithm can use a spatial data structure to efficiently skip over the transparent voxels, so the rendering time depends linearly on the number of non-transparent voxels (which is proportional to the size of our run-length encoded representation). There is also overhead due to traversing the spatial data structure. In the case of the shear-warp algorithm the overhead is proportional to the number of runs.

In the third performance regime spatial data structures are ineffective but early ray termination improves performance. For this regime to apply the average voxel opacity must be relatively high. Furthermore the side of the volume facing the viewer must be covered with high-opacity voxels so that viewing rays do not penetrate far into the volume before they are terminated. These conditions apply to the first and third cases on the bottom row of Figure 5.18. The second case on the bottom row may be in this performance regime or in the first regime depending on whether or not the spaces between the non-transparent voxels leave "holes" that can be penetrated by viewing rays. For the third regime rendering time depends linearly on the number of non-occluded voxels.

Finally, in the fourth performance regime both optimizations are effective. The last case in the bottom row of Figure 5.18 falls into this regime, as do all of the data sets in the performance tests described earlier (except the CT engine data set which has low average opacity). Rendering time depends linearly on the number of visible voxels (voxels that are neither transparent nor occluded). For this type of volume the number of visible voxels is proportional to the projected surface area of the objects in the volume, so the rendering algorithm only processes $O(n^2)$ voxels.

5.3.2 Voxel Throughput

To demonstrate the characteristics of the shear-warp algorithm in each performance regime we conducted two experiments. For both experiments we used the parallel projection algorithm with a run-length encoded volume.

The first experiment measures how rendering time depends on the number of visible voxels. In this experiment the amount of coherence and the number of transparent voxels are constant, but the average voxel opacity varies. This experiment also measures the renderer's voxel throughput.

To perform the experiment we constructed a series of 256x256x256 voxel volumes with constant opacity. Each volume had a different opacity, ranging from 0.0 (entirely transparent) to 1.0 (entirely opaque). Since the volumes had constant opacity all of the run-length encoded data structures contained exactly one non-transparent run per voxel scanline, resulting in a constant (high) amount of coherence and a constant (low) fraction of transparent voxels.

In this experiment the opacity associated with each volume determines the number of

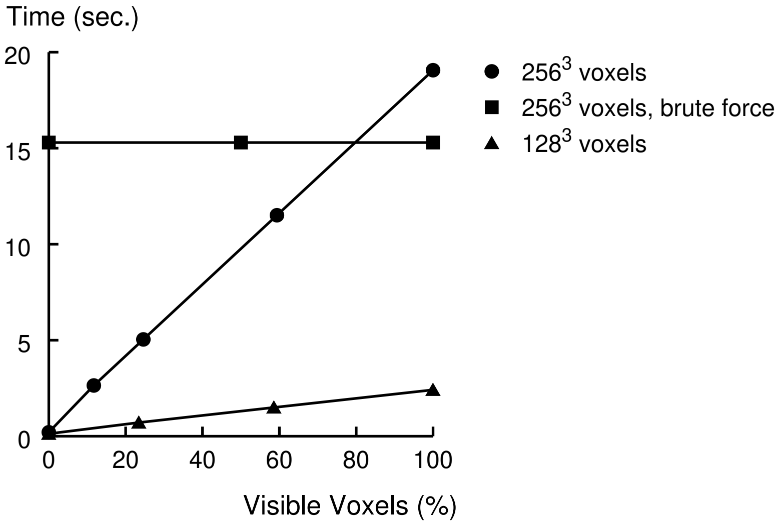


Figure 5.19: Results of the voxel throughput experiment for a 256^3 voxel volume with the shear-warp algorithm, a 256^3 voxel volume with a brute-force algorithm, and a 128^3 voxel volume with the shear-warp algorithm. In each experiment the amount of coherence is held constant while the voxel opacity is varied so that the number of visible voxels changes. The results show that the rendering time using the optimized shear-warp algorithm is proportional to the number of visible voxels (voxels that are neither transparent nor occluded).

visible voxels (since we also hold the minimum opacity threshold constant). Only the front slices of the high-opacity volumes are visible, while all of the voxels are visible in the low-opacity volumes. We expect rendering time to vary linearly with the number of visible voxels since the overhead due to the coherence data structures is constant.

Figure 5.19 shows the results of the experiment. When plotted against the number of visible voxels, the timings for the optimized shear-warp algorithm do in fact lie on a straight line. The slope of the line indicates that the voxel throughput is 880 Kvoxels/sec. (on a 150 MHz R4400 SGI Indigo2).

The figure also shows the rendering time for a brute-force shear-warp algorithm with both early-ray termination and run-length encoding disabled (the same algorithm as we used for Test 5 in Figure 5.11). The crossover point at which the brute-force algorithm matches the optimized algorithm occurs when 80% of the voxels are visible. When all voxels are visible the optimized algorithm is about 25% slower than the brute-force algorithm. The additional cost in the optimized algorithm is due to early-ray termination tests, but the worst-case

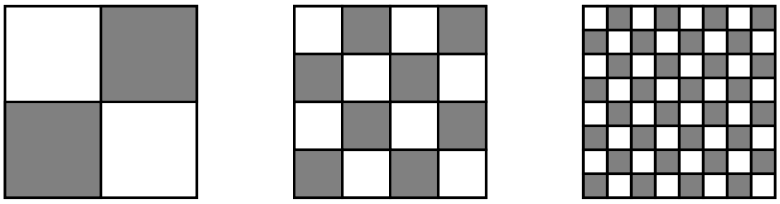


Figure 5.20: Test volumes for the voxel coherence experiment: Each volume in a series contains a checkerboard pattern constructed from blocks half as big as in the previous volume. The white blocks represent transparent voxels and the gray blocks represent voxels with a constant, non-zero opacity. The number of visible voxels is constant but the number of voxel runs increases by a factor of two in each volume.

25% performance degradation is small compared to the order-of-magnitude performance increase for more typical volumes. The only disadvantage of the shear-warp algorithm for low-coherence volumes is that the algorithm requires more memory than a brute-force algorithm since it stores three copies of every non-transparent voxel. However, as we argue later in Section 5.3.4, volumes with high percentages of visible voxels result in cluttered, fuzzy images and are therefore not useful.

5.3.3 The Impact of Coherence on Rendering Time

The second experiment measures how rendering time depends on the amount of coherence in the volume data. In this experiment the amount of coherence varies, but the number of transparent voxels is constant. We also keep the average voxel opacity constant, although we repeat the experiment for several opacity values.

To perform the experiment we construct several series of 256x256x256 voxel volumes containing checkerboard patterns (Figure 5.20). Half of the blocks in each checkerboard contain transparent voxels and the other half contain voxels with a constant, non-zero opacity. We use a different opacity for the non-transparent voxels in each series. For a given series, the non-transparent voxels in the different volumes all have the same opacity but we vary the size of the blocks to produce different numbers of voxel runs. For a viewing direction perpendicular to one face of the volume the number of visible voxels is constant over an entire series of volumes since exactly 50% of the voxels in each slice of each volume are transparent.

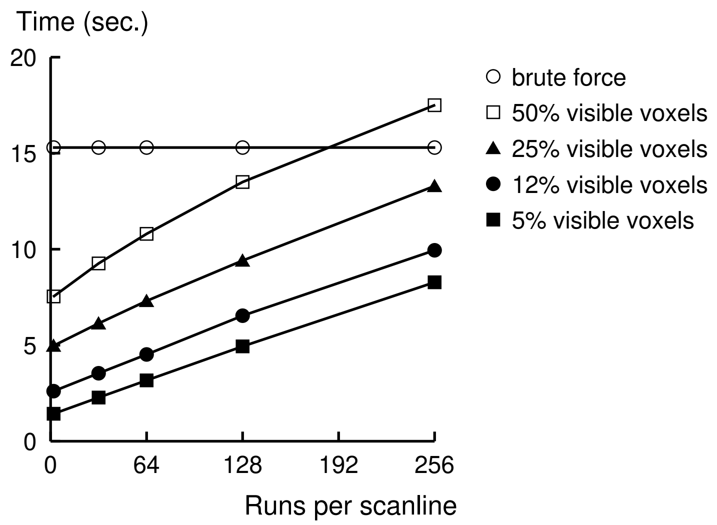


Figure 5.21: Results of the voxel coherence experiment: with a fixed number of visible voxels, the rendering time using the optimized shear-warp algorithm is proportional to the number of voxel runs in the volume. A small number of runs per scanline corresponds to high coherence.

In this experiment the number of blocks in the checkerboard pattern determines the number of voxel runs in the run-length encoded volume. We expect rendering time to vary linearly with the number of runs since the cost of rendering the voxels is the same for every volume in a given series. The number of runs per scanline includes both transparent runs and non-transparent runs, and every scanline has an even number of voxel runs as described in Section 4.1.2 (so a completely-transparent voxel scanline contains two runs, including a zero-length non-transparent run).

Figure 5.21 shows the results of the experiment. As expected, the plot of rendering time versus number of runs yields a straight line. From the slope of the line we deduce that the overhead incurred by traversing the run-length encoded data structures increases at a rate of 400 nsec./run on our test machine.

Typical data sets such as the classified CT and MR scans used earlier in this chapter have 5-15 runs/scanline. We can compute the overhead incurred by the spatial data structures by multiplying the number of runs, the runs per scanline, and the cost per run. The overhead for these data sets ranges from 130 msec. to 180 msec., a small fraction of total rendering time.

For volumes with 40% or less of the voxels visible the optimized algorithm always outperforms the brute-force algorithm, regardless of the number of voxel runs. For a volume with 50% of the voxels visible the crossover point occurs at 192 runs/scanline. In the worst case, with 256 runs/scanline (implying 50% of the voxels visible), the optimized algorithm is only 15% slower than the brute force algorithm. Furthermore, volumes with such large numbers of runs are unlikely to occur in practice: the number of runs is proportional to the depth complexity, and useful classification functions rarely produce volumes with such high depth complexity.

From the two experiments we conclude that even for volumes with worst-case coherence (only one voxel per run) the optimized shear-warp algorithm never performs significantly worse than a performance-tuned brute-force algorithm.

These results also provide us with an empirically-derived first-order cost polynomial for the shear-warp algorithm:

$$T_{\text{render}} = \frac{N_v}{800 \text{ Kvoxels/sec.}} + N_r \cdot 400 \times 10^{-9} \text{ sec.}$$

where T_{render} is the time to render a volume, N_v is the number of visible voxels (voxels that are neither transparent nor occluded) and N_r is the number of voxel runs in the entire volume. The constants in this equation assume a 150 MHz SGI Indigo2 workstation and that shading is implemented with a gray scale lookup table. Second-order effects such as the cost of the 2D warp have not been included, so this equation is most valid when the rendering time is large.

The cost polynomial is useful for estimating whether or not a volume falls into a category that can be rendered at interactive rates. For example, the CT head data set (Figure 5.2) contains 800,000 non-transparent voxels and about half of them are visible (the average opacity is high and the depth complexity is about two) so $N_v = 400,000$. The number of runs is $N_r = 340,000$. Thus the predicted rendering time is about 0.6 sec. The CT engine data set (Figure 5.6) has about 600,000 voxels, most of which are visible since the average opacity is relatively low, and the number of runs is 300,000. Thus the predicted performance is about 0.9 sec. for a grayscale rendering.

5.3.4 The Role of Coherence in Visualization

In this chapter we have shown how the shear-warp algorithm exploits coherence in classified volumes to achieve fast rendering rates. Coherence is also important for producing an informative visualization: an effective classification function emphasizes structure and eliminates clutter.

This statement does not imply that the original volume data has obvious structure or coherence. Visualization is an art that requires choosing appropriate visual representations of the information in the underlying data. Haber & McNabb [1990] and Delmarcelle & Hesselink [1995] propose frameworks for scientific visualization based on three steps: data preprocessing, visualization mapping, and rendering. Visualization mapping is the process of choosing an appropriate "icon" that represents the information in a data set. An icon should efficiently convey the essential information and avoid clutter that obscures the information.

In volume rendering, the icon is the classified volume. The visualization mapping step consists of choosing a classification function and the rendering step consists of transforming the classified volume into an image. Good classification functions expose structure, thereby guaranteeing coherence that we can take advantage of. Examples of structure include opaque or translucent surfaces, fuzzy surfaces, or small, indistinct clouds in a larger structured context.

Volume rendering is often called "direct volume rendering" to emphasize the fact that the volume is not first mapped to polygons, in contrast to surface rendering. Nevertheless, volume rendering does not eliminate the fundamental need for careful visualization mapping. The human visual system cannot extract useful information from a fuzzy, translucent cloud.

Useful classifications functions produce high-coherence classified volumes, precisely the type of volume that can be rendered at interactive rates with the shear-warp algorithm.

5.4 Chapter Summary

The performance results in this chapter lead to several general conclusions about coherence-accelerated volume rendering algorithms. Most importantly, the cost of traversing spatial data structures is the primary overhead. The shear-warp algorithm achieves speedups of 4x or more relative to octree-accelerated ray casters by reducing this cost: the ray caster must perform analytic geometry calculations to traverse octree nodes, whereas the shear-warp algorithm simply traverses run-length encoded scanlines in storage order.

We also demonstrated that the performance of coherence-accelerated volume rendering algorithms degrades gracefully for volumes with lower coherence, and that in these cases the scanline-order shear-warp algorithm has better memory performance than a ray caster.

These performance gains come without sacrificing image quality, provided that the volume data is band-limited and properly classified. For these reasons, scanline-order volume rendering algorithms such as the shear-warp algorithm are likely to dominate.