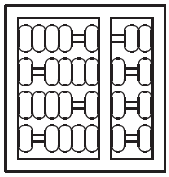


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Shear-Warp Shell Rendering

A.X. Falcão *L.M. Rocha* *J.K. Udupa*

Technical Report - IC-02-011 - Relatório Técnico

September - 2002 - Setembro

Abstract

In Medical Imaging, shell rendering (SR) and shear-warp rendering (SWR) are two ultra-fast and effective methods for volume visualization. We have previously shown the fact that, typically, SWR can be on the average 1.38 times faster than SR, but it requires from 2 to 8 times more memory space than SR. In this paper, we propose an extension of the compact shell data structure utilized in SR to allow shear-warp factorization of the viewing matrix in order to obtain speed up gains for SR, without paying the high storage price of SWR. The new approach is called *shear-warp shell rendering* (SWSR). The paper describes the methods, points out their major differences in the computational aspects, and presents a comparative analysis of them in terms of speed, storage, and image quality. The experiments involve hard and fuzzy boundaries of 10 different objects of various sizes, shapes, and topologies, rendered on a 1GHz Pentium-III PC with 512MB RAM, utilizing surface and volume rendering strategies. The results indicate that SWSR offers the best *speed* and *storage* characteristics compromise among these methods. We also show that SWSR improves the rendition quality over SR, and provides renditions similar to those produced by SWR.

Index terms: Volume visualization, 3D imaging, volume rendering, surface rendering, shear-warp factorization, shell rendering, shear-warp rendering, medical imaging.

1 INTRODUCTION

There are three important steps in volume visualization - classification, representation, and rendering. *Classification* is the process of assigning opacities to image elements in the object region proportional to the degree of emphasis to be given to these objects for visualization. The greatest challenge in classification is to identify the voxels that belong to distinct objects, which is, in essence, a segmentation problem. *Representation* consists of defining a geometric model for the classified object regions and a data structure to store the model together with a set of attributes for visualization. It has a direct impact on the efficiency of the visualization algorithms. Rendering is the process of simulation of light propagation within the object regions and of determining the light that reaches the viewpoint to create different views of the selected objects.

Approaches to volume visualization may be divided according to the classification process into two groups - surface rendering and volume rendering. *In surface rendering*, the voxels are classified as either opaque or transparent [1, 2, 3, 4]. As a consequence, a geometric model of the surface of the objects can be created for visualization. The methods may be further divided based on the two classes of geometric models used- polygonal and digital. Digital methods represent the surface of the objects as a set of primitives - voxels and voxel faces - that are directly associated with the discrete coordinate system of the volumetric data [1, 4]. In polygonal methods, a set of polygons - most commonly triangles - are used to represent the surface [2, 3]. In *volume rendering*, the classification process is fuzzy where the voxels are considered to have a continuous degree of opacity from transparent to opaque [5, 6, 7, 8]. The geometric model of the objects is digital (a set of voxels), and in the case of binary classification, volume rendering is equivalent to digital surface rendering, which makes it an interesting, simple, and flexible approach for volume visualization.

Two further classes, called scene-based (or image-based) and object-based, may be identified within volume rendering (see [9] for a detailed discussion). In *scene-based* methods, renditions are created directly from given 3D scenes. In *object-based* methods, structures (hard or fuzzy) representing object information are explicitly defined first and then rendered. The motivation for developing scene-based methods is the immediacy of response and interactability - by changing the values assigned to a small set of parameters [10, 11], we would like to render information about different objects in the scene. We must note that this latter process is nothing but image segmentation (no matter what name we ascribe to it - classification, applying a transfer function, etc.), although this

is done on the fly and not explicitly, unlike in object-based methods, wherein, prior to rendering, an explicit structure representation is sought via image segmentation. An advantage of scene-based rendering is that an explicit and perfect object segmentation (hard or fuzzy) is not needed, the idea being that all imperfections will be filtered out by the human observer who consumes the rendering. The main motivations for object-based rendering are beyond mere visualization, namely, quantification and analysis (e.g. [12]). Here, perfect segmentation is a necessity. Since methods to achieve accurate segmentation (e.g. [13, 14, 15, 16]) are likely to be complex (invariably requiring some human interaction) and computationally expensive, they cannot be utilized in scene-based redering methods to give the immediacy of response and interactability. More sophisticated rendering strategies (e.g. [17, 18]) will also encounter the same problem. To address the speed issue, specialized hardware [19, 20] and algorithms that trade off image quality for speed [21, 22] have been proposed. While the development of specialized hardware is relevant to advance the state of knowledge in computing, addressing the problem algorithmically and via software is preferable from the practical viewpoint of availability and portability of implementations. In this regard, object-based methods have distinct advantages as has been repeatedly demonstrated in the literature. To sum up, the dichotomy that exists between scene-based and object-based rendering methods is real and both approaches are important and have their niche. Their differences may disappear in the future when sufficient computing power becomes available to perform advanced image segmentation in real time on the fly while rendering. In that case, all segmentation methods, characterized by a manageable-size set of parameters, may be usable as opacity transfer functions. Among object-based rendering approaches, two of the most successful techniques have been *shell rendering* (SR) [5] and *shear-warp rendering* (SWR) [8].

SR uses a compact data structure, called shell, based on a list of visualization attributes for non-transparent voxels and a 2D array of pointers to that list. The shell data structure allows random access to the voxels in the pointer array. SWR uses one list of visualization attributes for non-transparent voxels, and for each principal axis of the 3D coordinate system, it uses one list to store the length of transparent/non-transparent voxel runs and one 1D pointer array to the corresponding run-length list and to the voxel list. SR uses front-to-back voxel projection (voxel projection was first suggested in [23, 24] which was further studied and termed voxel splatting [25, 26, 27]). SWR combines the advantages of front-to-back voxel splatting and ray casting by using the shear-warp factorization of the viewing matrix [28]. They both provide very efficient surface and volume rendering with high-quality images. Surface SR has proven to be from 18 to 31 times faster on a 300MHz Pentium PC than even hardware-based polygonal rendering methods, including those based on very expensive machines such as the Silicon Graphics Reality Engine II [29]. SR has also been extended to provide digital perspective [30] and to render polygonal geometric models within the same framework [31] without compromising speed. SWR has been actively pursued in several aspects - parallel, sequential, software-based, and hardware-based implementations [32, 33, 34, 35, 36, 37, 38, 39, 40].

We have shown that, typically, SWR can be on the average 1.38 times faster than SR, but it requires from 2 to 8 times more memory space than SR [41]. In this paper, we propose an extension of the compact data structure utilized in SR to allow shear-warp factorization of the viewing matrix in order to obtain speed up gains for SR, without paying the high storage price of SWR. The new method is called *shear-warp shell rendering* (SWSR). We present a comparative analysis of the three approaches in terms of speed, storage, and image quality.

The paper is organized as follows. Section 2 describes the methods and points out their major differences in the computational aspects. The experiments for comparison and the results are presented in Section 3. In Section 4, we discuss the advantages and disadvantages of each approach and state our conclusions.

2 METHODS

A *scene* is a pair (V, f) consisting of a finite rectangular array V of *voxels* (points in $\mathcal{Z}^3$), and a mapping f that assigns to each voxel v in V an *intensity* value $f(v)$ in some range of values which are usually integers. A scene is usually stored as a finite 3D rectangular array of voxels, where each voxel represents a small cuboid in $\Re^3$. Without loss of generality, we will assume that all voxels are of identical size and so each voxel can be identified by a triple (x, y, z) in $\mathcal{Z}^3$ of the coordinates of its center. When it is not the case, this form can be achieved by interpolation [9].

An *object* in V is a set O of voxels forming one or more connected components in $\mathcal{Z}^3$. We are interested in visualizing the subset B of voxels in O that belong to the vicinity of the object's boundary. Image segmentation is the process of identifying such a subset. The aim of classification is to transform (V, f) into another scene (B, ω) by assigning an opacity value $\omega(v)$ lying in the interval $[0, 1]$ to each voxel v in B. We call (B, ω) a *fuzzy boundary*, and when the opacities assigned are only one of 0 (transparent) and 1 (opaque)(i.e., the range of ω is {0,1}), we call it a *hard boundary*.

There are many methods of voxel classification [42, 43]. In this paper, we shall assume that classification has already been done by one of these approaches such that the fuzzy/hard boundary is the same for all visualization techniques. Since both methods have been published, we will present only a brief description of these methods and analyze the aspects that affect their overall efficiency for orthogonal projection. Please refer to the original papers for details.

2.1 Shell rendering

Given a fuzzy/hard boundary, SR uses a compact data structure, called *shell*, to encode only non-transparent voxels of the boundary. Each voxel is represented by an opacity value in (Ω_l, Ω_h), its x coordinate in the scene, and other visualization attributes. A voxel v is stored in the shell if $\omega(v) > \Omega_l$ and if v is not completely surrounded by voxels v' such that $\omega(v') \geq \Omega_h$. Such voxels are stored in a list V_x, starting from $(x, y, z) = (0, 0, 0)$, in a x-by-x, y-by-y, and z-by-z order of the voxel coordinates in the scene. A 2D pointer array P_{yz} indicates the first voxel in the list associated with a particular coordinate (y, z) in the scene. Figure 1a illustrates a scene of size 3x3x3 voxels where the voxels are numbered from 1 to 27, following the x-by-x, y-by-y, and z-by-z order. Assume that the voxels satisfying the above opacity conditions, and hence stored in the shell, are only those numbered from 1 to 9. The pointer array P_{yz} and the list V_x of voxels for this example are shown in Figure 1b.

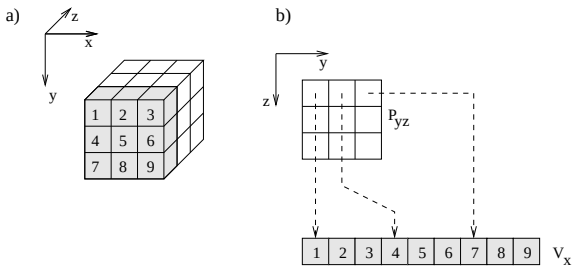


Figure 1: (a) A scene of size 3x3x3 voxels where the voxels are numbered from 1 to 27, following the x-by-x, y-by-y, and z-by-z order, and only those numbered from 1 to 9 are voxels of the shell. (b) The list V_x and the pointer array P_{yz} for this simple example.

Fast orthogonal front-to-back voxel projection is possible by just sweeping the pointer array and the list of voxels in eight different ways. SR exploits the fact that in orthogonal projection, there is no preference of order among x, y, and z. Thus, since the voxels in the list are encoded along

Octant	Voxel Indexing Order
1	$x^- \rightarrow x^+$, $y^- \rightarrow y^+$, $z^- \rightarrow z^+$
2	$x^+ \rightarrow x^-$, $y^- \rightarrow y^+$, $z^- \rightarrow z^+$
3	$x^- \rightarrow x^+$, $y^+ \rightarrow y^-$, $z^- \rightarrow z^+$
4	$x^+ \rightarrow x^-$, $y^+ \rightarrow y^-$, $z^- \rightarrow z^+$
5	$x^- \rightarrow x^+$, $y^- \rightarrow y^+$, $z^+ \rightarrow z^-$
6	$x^+ \rightarrow x^-$, $y^- \rightarrow y^+$, $z^+ \rightarrow z^-$
7	$x^- \rightarrow x^+$, $y^+ \rightarrow y^-$, $z^+ \rightarrow z^-$
8	$x^+ \rightarrow x^-$, $y^+ \rightarrow y^-$, $z^+ \rightarrow z^-$

Table 1: This table shows the voxel indexing order in SR when the viewpoint is situated in each octant in Figure 2. The notation $x^- \rightarrow x^+$, $y^- \rightarrow y^+$, $z^- \rightarrow z^+$ indicates that the voxels should be indexed from the lowest to the highest coordinate along each axis, x, y, and z, respectively, for a front-to-back projection.

x, it is faster to project them onto the viewing plane by going along x before going along y and z. Figure 2 illustrates a scene with its coordinate system where each octant has an identification number from 1 to 8. The indexing order will depend on the octant in which the viewing point is situated. Table 1 shows the front-to-back indexing order for each octant in Figure 2, where the notation $x^- \rightarrow x^+$, $y^- \rightarrow y^+$, $z^- \rightarrow z^+$ indicates that the voxels should be indexed from the lowest to the highest coordinate along each axis, x, y, and z, respectively. Figure 2 and Table 1 point out that no matter where the viewpoint is located (outside the object), correct front-to-back voxel projection to determine visibility can be achieved simply by accessing the voxels always in a column-by-column (x-first), row-by-row (y-next) and slice-by-slice (z-last) order from the shell.

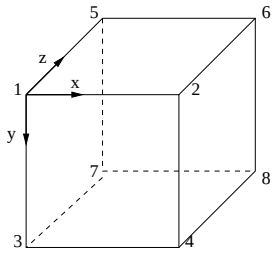


Figure 2: A scene where each octant has an identification number from 1 to 8.

One drawback of voxel splatting [25, 26, 27], and so, of SR, is the difficulty of implementing early ray termination, an effective optimization of ray-casting algorithms that avoids visiting hidden voxels in the scene. Although, SR does not compute shading for hidden voxels, the process of determining whether or not a voxel is hidden requires the application of the viewing matrix to it. Unfortunately, this is the most costly operation during rendering by voxel splatting, and so, the speed in SR is more dependent on the number of voxels in the shell than it is in ray-casting algorithms.

An alternative solution to make voxel splatting less dependent on the number of stored voxels is the shear-warp factorization of the viewing matrix [28]. SWR uses this technique to provide fast visualization with early ray termination.

2.2 Shear-warp rendering

SWR requires first selecting a principal axis among x,y, and z that is the closest to the perpendicular to the viewing plane. Then, the parallel slices of the scene orthogonal to the selected principal axis are sheared in 3D and orthogonally projected to compose a distorted intermediate image (see Figure 3). Afterwards, the final image is produced by a 2D warp transformation. Since the scan-lines of pixels in the intermediate image are parallel to scan-lines of voxels in the scene, early ray termination can be implemented by scanning the intermediate image and the scene simultaneously. Further, all voxels in a given slice are sheared by the same factor, which makes it possible to implement the 3D shear operation by look-up table. The factorization also avoids the resampling complications of voxel splatting algorithms, since the resampling weights are the same for every voxel in a slice.

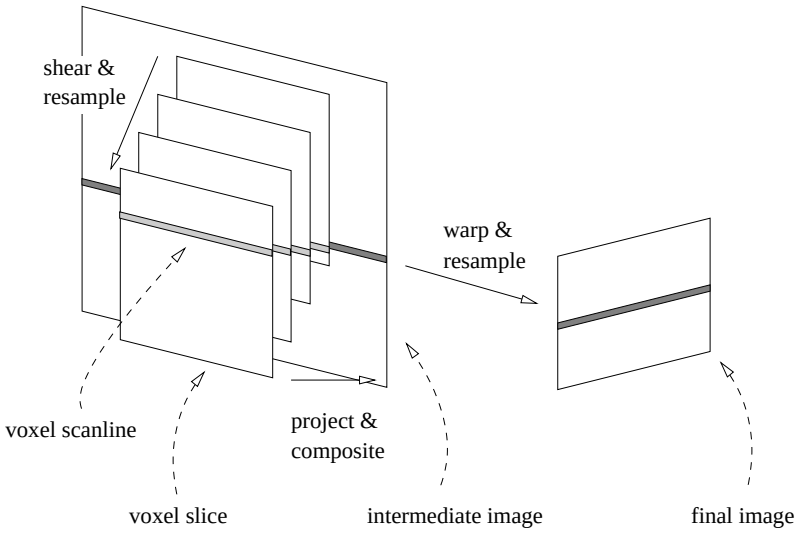


Figure 3: SWR consists of a 3D shear transformation and projection onto the intermediate image followed by a 2D warp transform. Note that, the scan-lines in the intermediate image and in the scene are parallel, making it possible to implement early ray termination.

The data structure in SWR consists of seven parts: one list of visualization attributes for non-transparent voxels, and for each principal axis, one list to store the length of transparent/non-transparent voxel runs and one 1D pointer array to the corresponding run-length list and to the voxel list. Figure 4 illustrates this data structure for the example shown in Figure 1a. Note that each pointer array indicates the beginning of the run-length encoding in each slice orthogonal to the principal axis, and in the list of voxels, it indicates the first non-transparent voxel of each orthogonal slice. The x, y and z coordinates of the voxels are not stored. They can be computed by sweeping the pointer array and the run-length list. Therefore, the access to the voxels is sequential in the run-length list for a given slice orthogonal to the principal axis.

Note that, for a given fuzzy/hard boundary, SWR works only in non-transparent regions of the scene. It also run-length encodes the intermediate image in runs of opaque and non-opaque pixels to skip occluded voxels in the scene during rendering, thus allowing early ray termination. Note that, run-length encoding in the intermediate image changes during rendering, but it is maintained by a *disjoint set data structure* [44]. One drawback, however, is that one run-length encoding in the scene is required for each principal axis direction. Consequently, the scene must be run-length encoded starting from $(x, y, z) = (0, 0, 0)$, in a x-by-x, y-by-y, and z-by-z order, when z is the principal axis; in a y-by-y, z-by-z, and x-by-x order, when x is the principal axis; and in a x-by-x, z-by-z, and

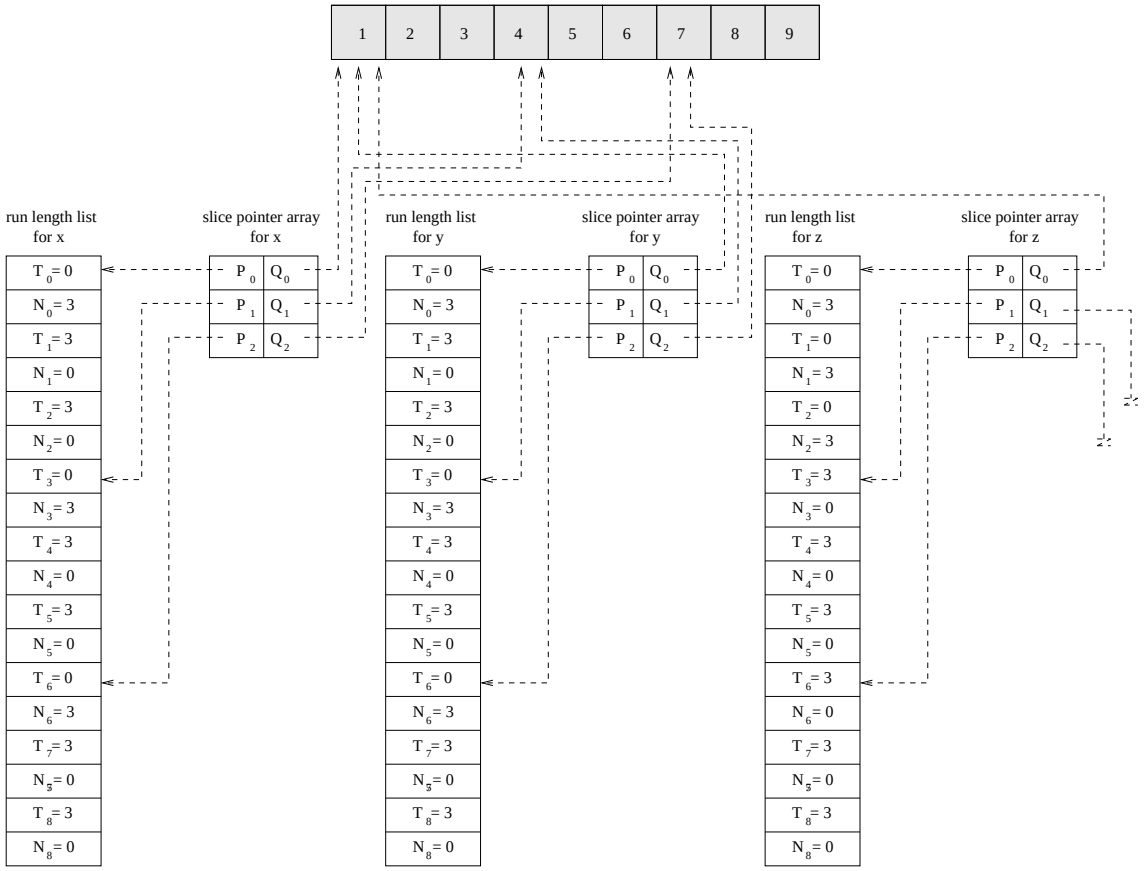


Figure 4: Data structure for SWR: the run-length lists, the slice pointer arrays, and the voxel list for the example of Figure 1a.

y-by-y order, when y is the principal axis. Table 2 presents the front-to-back indexing order in SWR for each octant and principal axis shown in Figure 2. Observe that now there is a precedence order among x, y, and z, because the principal axis must be indexed at the end.

We observe that the memory space requirement and the implementation complexity in SWR rendering are much higher than in SR. In fact, we have shown that SWR requires from 2 to 8 times more memory space than SR [41]. On the other hand, SWR can be on the average 1.38 times faster than SR because it skips occluded voxels in the scene and projects non-occluded voxels on the intermediate image by a look-up table operation. We encounter here the classical dilemma between memory space and computational time, where the former favors SR and the latter favors SWR. Therefore, a question that naturally arises is how feasible would be a method that combines the compact data structure of SR with the speed-up advantages of the SWR. The answer to this question is addressed next.

2.3 Shear-warp shell rendering

The main difficulty in incorporating shear-warp factorization in SR is the imposed order of precedence among the axes x, y, and z, where the principal axis must be indexed at the end. This is counterproductive in SR whenever x is the principal axis (see the first column in Table 2), because

Octant	Voxel indexing order when the principal axis is x	Voxel indexing order when the principal axis is y	Voxel indexing order when the principal axis is z
1	$y^- \rightarrow y^+$,$z^- \rightarrow z^+$,$x^- \rightarrow x^+$	$x^- \rightarrow x^+$,$z^- \rightarrow z^+$,$y^- \rightarrow y^+$	$x^- \rightarrow x^+$,$y^- \rightarrow y^+$,$z^- \rightarrow z^+$
2	$y^- \rightarrow y^+$,$z^- \rightarrow z^+$,$x^+ \rightarrow x^-$	$x^+ \rightarrow x^-$,$z^- \rightarrow z^+$,$y^- \rightarrow y^+$	$x^+ \rightarrow x^-$,$y^- \rightarrow y^+$,$z^- \rightarrow z^+$
3	$y^+ \rightarrow y^-$,$z^- \rightarrow z^+$,$x^- \rightarrow x^+$	$x^- \rightarrow x^+$,$z^- \rightarrow z^+$,$y^+ \rightarrow y^-$	$x^- \rightarrow x^+$,$y^+ \rightarrow y^-$,$z^- \rightarrow z^+$
4	$y^+ \rightarrow y^-$,$z^- \rightarrow z^+$,$x^+ \rightarrow x^-$	$x^+ \rightarrow x^-$,$z^- \rightarrow z^+$,$y^+ \rightarrow y^-$	$x^+ \rightarrow x^-$,$y^+ \rightarrow y^-$,$z^- \rightarrow z^+$
5	$y^- \rightarrow y^+$,$z^+ \rightarrow z^-$,$x^- \rightarrow x^+$	$x^- \rightarrow x^+$,$z^+ \rightarrow z^-$,$y^- \rightarrow y^+$	$x^- \rightarrow x^+$,$y^- \rightarrow y^+$,$z^+ \rightarrow z^-$
6	$y^- \rightarrow y^+$,$z^+ \rightarrow z^-$,$x^+ \rightarrow x^-$	$x^+ \rightarrow x^-$,$z^+ \rightarrow z^-$,$y^- \rightarrow y^+$	$x^+ \rightarrow x^-$,$y^- \rightarrow y^+$,$z^+ \rightarrow z^-$
7	$y^+ \rightarrow y^-$,$z^+ \rightarrow z^-$,$x^- \rightarrow x^+$	$x^- \rightarrow x^+$,$z^+ \rightarrow z^-$,$y^+ \rightarrow y^-$	$x^- \rightarrow x^+$,$y^+ \rightarrow y^-$,$z^+ \rightarrow z^-$
8	$y^+ \rightarrow y^-$,$z^+ \rightarrow z^-$,$x^+ \rightarrow x^-$	$x^+ \rightarrow x^-$,$z^+ \rightarrow z^-$,$y^+ \rightarrow y^-$	$x^+ \rightarrow x^-$,$y^+ \rightarrow y^-$,$z^+ \rightarrow z^-$

Table 2: This table shows the voxel indexing order in SWR when the viewpoint is situated in each octant in Figure 2, where the principal axis determines an order of precedence among x, y, and z, respectively. The notation $x^- \rightarrow x^+$, $y^- \rightarrow y^+$, $z^- \rightarrow z^+$ indicates that the voxels should be indexed from the lowest to the highest coordinate along each axis, x, y, and z, respectively, for a front-to-back projection.

it requires binary searches in the list V_x of voxels for each (y, z) coordinate. The same holds for early ray termination. A binary search is necessitated every time occluded voxels are skipped in the scene. If there exist many runs of length one, which is very likely, the number of binary searches makes SR expensive.

In order to allow shear-warp factorization in SR, we add to the shell data structure a second 2D pointer array P_{xz} and a second list of voxels V_y, as illustrated in Figure 5 for the example of Figure 1a. By indexing the scene, starting from $(x, y, z) = (0, 0, 0)$, in a y-by-y, z-by-z, and x-by-x order, we store in the second list the y coordinate of each voxel of the shell and a pointer to its corresponding position in the first list. Similarly, each pointer in the second array indicates the first voxel in the second list associated with a particular (x, z) coordinate in the scene. Note that, the other attributes for visualization are stored only in the first list V_x. Thus, we use the second pointer array and the second list whenever the principal axis is x.

In SWSR, the 3D shear operation is implemented by look-up table, avoiding the floating-point multiplications of the viewing matrix for each voxel of the shell. The number of multiplications is considerably reduced to that of the 2D warp transformation of the intermediate image. Unfortunately, other optimizations, such as early ray termination, are still counterproductive in SWSR due to the aforementioned reasons.

We are now ready to analyze how these three visualization techniques compare in terms of speed, storage and image quality. We address these issues next.

3 COMPARISON AND RESULTS

For our experiments, we have chosen 10 objects O_i, $i = 1, 2, \ldots, 10$, of different sizes and derived from scenes constituting different modalities, as described in Table 3. The number of voxels in the scenes ranges from 3,300,300 to 77,332,480 after interpolation. For each object, we have created two types of boundaries by classification. A hard boundary whose thickness is 1-voxel and a fuzzy boundary whose thickness is 3-voxels and each voxel has opacity 1/3. Their sizes range from 22,390 voxels to 2,833,821 voxels, covering all cases from small to very large boundaries. Table 3 also shows in parenthesis the percentage of reduction in the number of voxels in the boundaries compared to that in the original scenes. It illustrates the advantage of storing and processing only non-transparent

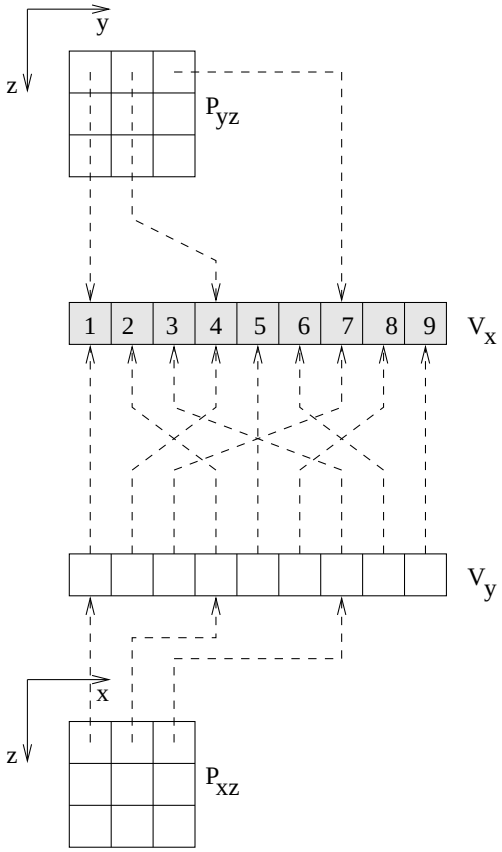


Figure 5: Data structure for SWSR: the list of voxels V_x, the pointer arrays P_{yz} and P_{xz}, and the pointer list V_y for the example of Figure 1a.

voxels.

3.1 Memory space

Table 4 shows the space requirement in Kbytes for SR, SWR, and SWSR to store the hard and fuzzy boundaries presented in Table 3. Note that, SWR requires from 2.21 (fuzzy boundary of O_5) to 8.14 (hard boundary of O_7) times more memory space than SR. SWSR represents an intermediary storage solution. It can save from 27% (fuzzy boundary of O_8) to 75% (hard boundaries of O_4, O_7, O_8 and O_9) of memory space when compared to SWR, but it requires from 1.38 (fuzzy boundary of O_5) to 2.0 (hard boundaries of $O_2, O_4 - O_7$ and O_9) times more memory space than SR.

3.2 Speed

The experiments were performed on a 1GHz Pentium-III PC with 512MB RAM. We have compared our own implementation of SR and SWSR with the implementation of SWR by Lacroute [8], which is available at $http://www-graphics.stanford.edu/software/volpack$. We use orthogonal projection with a single light source situated at the viewpoint in all methods. All programs use the C language, and we have been careful to set the same optimization options for their code compilations

Object	Scene (number of voxels)	Hard Boundary (number of voxels)	Fuzzy Boundary (number of voxels)
O_1 (knee - bones - CT)	3,300,300	134,370 (95.93%)	241,953 (92.67%)
O_2 (talus - MR)	8,584,216	22,390 (99.74%)	72,946 (99.15%)
O_3 (skull - CT)	11,769,912	261,676 (97.78%)	836,889 (92.89%)
O_4 (head - skin - MR)	11,927,552	419,186 (96.49%)	1,349,048 (88.69%)
O_5 (vessels - MRA)	16,777,216	29,753 (99.82%)	65,092 (99.61%)
O_6 (orbit - skin - CT)	25,165,824	250,974 (99.00%)	791,090 (96.86%)
O_7 (orbit - bones - CT)	25,165,824	263,961 (98.95%)	763,071 (96.97%)
O_8 (child skull - MR)	57,451,680	933,061 (98.38%)	2,833,821 (95.07%)
O_9 (head and spine - bones - CT)	77,332,480	745,824 (99.04%)	2,119,109 (97.26%)
O_{10} (head and spine - skin - CT)	77,332,480	780,135 (98.99%)	2,180,696 (97.18%)

Table 3: The objects selected for the experiments, the number of voxels in the interpolated scene in the hard and fuzzy boundary, respectively. The percent reduction in the number of voxels in the boundaries compared to that in the original scenes is shown in parenthesis.

Object	Hard boundary			Fuzzy boundary		
	SR	SWR	SWSR	SR	SWR	SWSR
O_1	600	4,404	1,192	1,324	4,328	2,505
O_2	218	1,164	437	618	1,504	974
O_3	1,260	9,640	2,473	4,515	12,720	8,601
O_4	1,819	14,732	3,639	6,951	19,204	13,538
O_5	372	1,776	744	830	1,832	1,148
O_6	1,172	9,132	2,345	4,247	13,256	8,109
O_7	1,223	9,952	2,446	4,110	10,972	7,836
O_8	4,284	34,104	8,412	14,959	39,436	28,796
O_9	3,503	27,772	7,007	11,527	32,576	21,874
O_{10}	3,637	25,468	7,275	11,828	36,148	22,476

Table 4: Space requirements in Kbytes to store the hard and fuzzy boundaries in the three approaches.

and to make sure that all programs render the same boundaries. In all programs, we use the function gettimeofday() to measure the average time to compute the rendering of 100 images of the same size from different directions. Observe that we are not considering here the time for classification and representation. These average computational times (in milliseconds) spent for rendering in the three approaches are shown in Table 5. We may make several observations from this table.

All methods can perform surface rendering in less than 0.5 second, and volume rendering in less than 1.0 second, except for SR that requires 1.42 second to render the fuzzy boundary of O_8. Note that SWR is not always faster than SR and SWSR. On the contrary, for the hard boundaries of objects O_2, O_5, O_6 and O_7, SR can be up to 2.05 times faster (object O_5) than SWR, and SWSR is always faster than SWR for hard boundaries. A common mistake made in many published papers is to assert that the speed of a visualization method is always inversely proportional to the boundary size. We graphically illustrate a counterexample in Figure 6, where rendering time for the hard boundaries of the objects are presented in the increasing order of their boundary size. In SWR, for example, the rendition speed strongly depends on the morphological and topological features of

Object	Hard boundary			Fuzzy boundary		
	SR	SWR	SWSR	SR	SWR	SWSR
O_1	76	49	45	135	75	86
O_2	32	54	32	60	62	49
O_3	144	119	99	426	198	248
O_4	208	127	115	639	229	333
O_5	41	83	44	66	88	64
O_6	168	209	152	478	292	351
O_7	173	189	136	480	257	290
O_8	475	380	317	1421	647	873
O_9	430	379	286	1226	597	729
O_{10}	421	408	330	1175	646	857

Table 5: Computational time in milliseconds (ms) to render the hard and fuzzy boundaries in the three approaches.

the boundary. Boundaries that provide long runs of opaque voxels tend to speed up the method, because they increase the effectiveness of early ray termination.

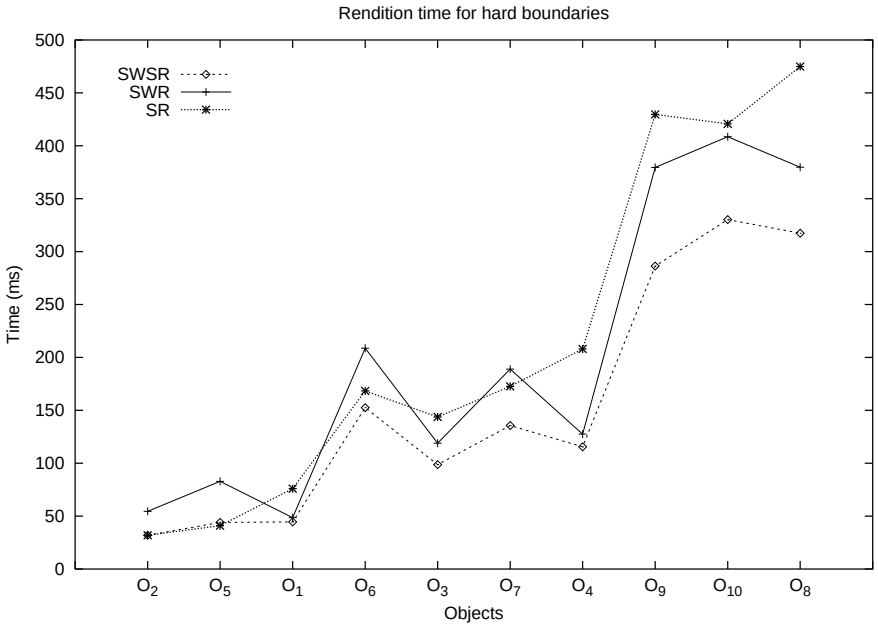


Figure 6: The rendition time (averaged over 100 views) for the three methods for hard boundaries as a function of their size. Objects are presented in the increasing order of their boundary size along the horizontal axis.

On the other hand, for fuzzy boundaries, the speed of SWR and of SWSR is certainly less affected by the size of the boundaries when compared to SR. This is because the former methods perform voxel splatting by look-up table operations while SR computes the multiplications of the viewing matrix. Note that, SWR and SWSR can be up to 2.79 and 1.92 times faster than SR, respectively (object O_4). Figure 7 illustrates this behavior of the methods, where the fuzzy boundaries of the

objects are presented in the increasing order of their size.

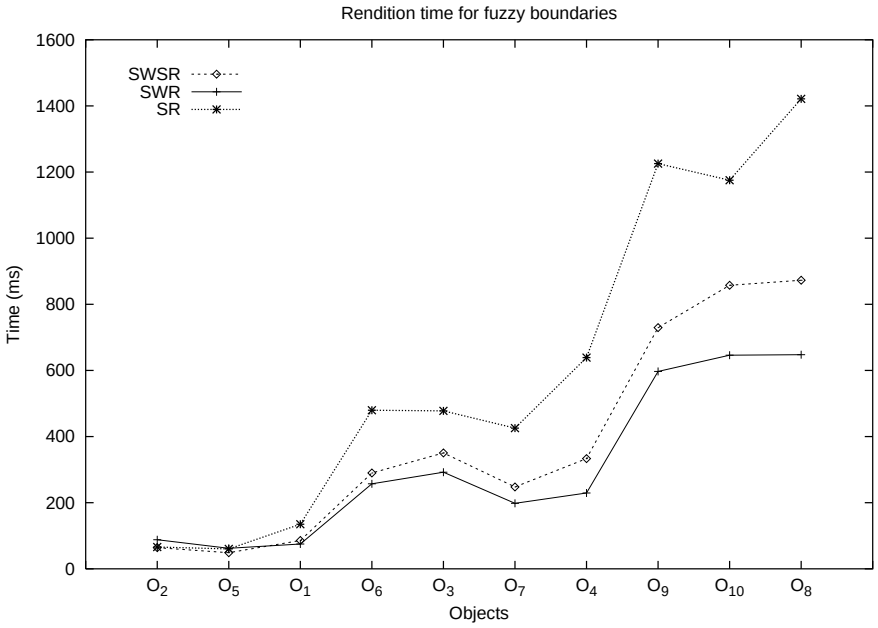


Figure 7: The rendition time (averaged over 100 views) for the methods for fuzzy boundaries as a function of their size. Objects are presented in the increasing order of their boundary size along the horizontal axis.

On average, we may say that SWSR and SWR are faster than SR, but only at the cost of memory space requirement, and SWSR offers the best *speed* vand *storage* characteristics compromise among them. Table 6 presents the speed gain (left entry) and the memory space cost (right entry) of SWSR over SR for hard and fuzzy boundaries. We observe that on average, SWSR is 1.35 and 1.52 times faster than SR for hard and fuzzy boundaries, respectively. This speed gain has the penalty of about 2 times more memory space requirement on average. According to the numbers presented in Table 4, this is certainly not prohibitive in modern personal computers, where the RAM memory available can reach 1 Gbytes. Table 7 presents the speed gain (left entry) and the memory space cost (right entry) of SWSR over SWR for hard and fuzzy boundaries. Note that on average, SWSR is 1.35 faster than SWR for hard boundaries and almost as fast as SWR for fuzzy boundaries. Besides, SWSR is 71% and 34% more space-efficient than SWR for hard and fuzzy boundaries, respectively.

3.3 Image quality

A well known problem of visualization methods based on one-to-one voxel projection is the apparence of holes in the rendered image for certain viewing directions (see Figures 8a and 8b). Post-processing algorithms may be used to "close" these holes, but their success is not guaranteed and the resulting image quality may not be optimal. A more effective and efficient solution [45, 5] is to make the size of the splatting mask bigger than the size of the pixels. Figure 8c shows the result of such an approach for the same viewing direction shown in Figure 8b. In such a case, however, we have discovered a conceptual problem in SR, which affects the quality and correctness of rendition.

SR exploits the property of non-preference order among x, y, and z for orthogonal projection to increase efficiency by indexing the axis x before y and z. Unfortunately, this property is not valid

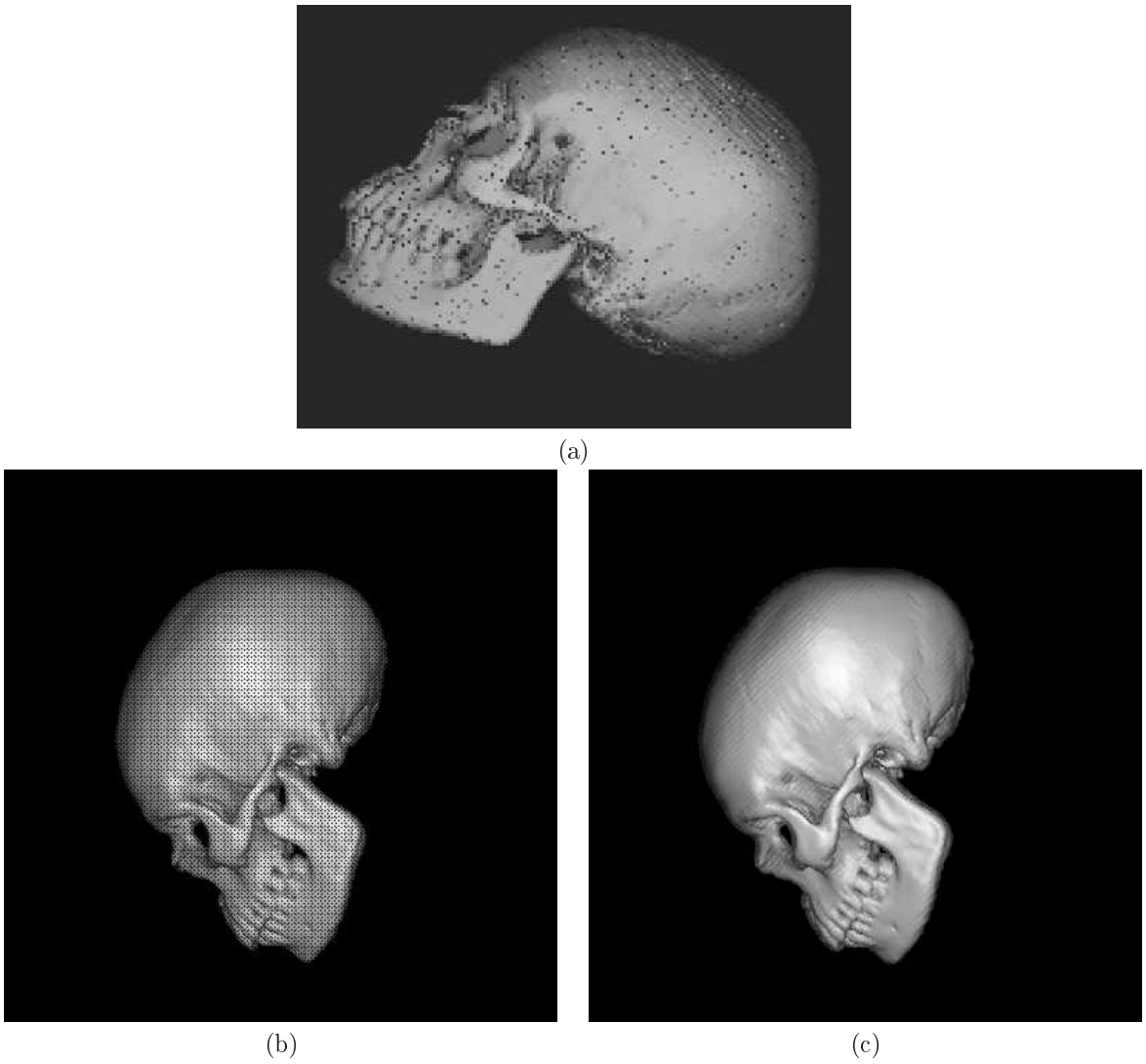
Object	Hard Boundary	Fuzzy Boundary
O_1	1.69 — 1.99	1.57 — 1.89
O_2	1.00 — 2.00	1.22 — 1.58
O_3	1.45 — 1.96	1.72 — 1.91
O_4	1.81 — 2.00	1.92 — 1.95
O_5	0.93 — 2.00	1.03 — 1.38
O_6	1.11 — 2.00	1.36 — 1.91
O_7	1.27 — 2.00	1.66 — 1.91
O_8	1.50 — 1.96	1.63 — 1.92
O_9	1.50 — 2.00	1.68 — 1.90
O_{10}	1.28 — 2.00	1.37 — 1.90
Average	1.35 — 1.99	1.52 — 1.82

Table 6: Speed gain (left entry) and memory space cost (right entry) of SWSR over SR for hard and fuzzy boundaries.

Object	Hard Boundary	Fuzzy Boundary
O_1	1.09 — 0.27	0.87 — 0.58
O_2	1.69 — 0.38	1.27 — 0.65
O_3	1.20 — 0.26	0.80 — 0.66
O_4	1.10 — 0.25	0.69 — 0.70
O_5	1.89 — 0.42	1.38 — 0.63
O_6	1.38 — 0.26	0.83 — 0.61
O_7	1.39 — 0.25	0.89 — 0.71
O_8	1.20 — 0.25	0.74 — 0.73
O_9	1.33 — 0.25	0.82 — 0.67
O_{10}	1.24 — 0.29	0.75 — 0.62
Average	1.35 — 0.29	0.90 — 0.66

Table 7: Speed gain (left entry) and memory space cost (right entry) of SWSR over SWR for hard and fuzzy boundaries.

when the splatting mask is bigger than the pixel size in the rendered image. Figure 9 illustrates an example where the observer is in octant 6, the voxel indexing order is as indicated in Table 1, the x axis is perpendicular to the viewing plane, and each voxel paints at most 3×3 pixels, depending on the direction of visualization [5]. Note that farther (e.g.; voxel 2) and hidden voxels are painted in place of closer (e.g.; voxel 1) and non-hidden voxels. The problem has solution based on depth-sorting, but the great idea of the front-to-back indexation is to avoid depth-sorting and this would compromise the efficiency of the method. This conceptual mistake is very difficult to detect, because it depends on the thickness of the shell and the voxel opacity values. It is more evident when the voxel opacities are lower, or even when they are higher, in cases when there are abrupt changes in the object's surface. Figure 10 shows image renditions with voxel splatting of a CT scene of a child skull (object O_3) for SR and SWSR. Figure 10a illustrates the problem of SR when the opacity values are low. Note that dark stains appear on the object's surface and these artifacts do not occur in SWSR as shown in Figure 10b. Figures 10c and d illustrate the other situation when voxels have high-opacity values. In this case, SR produces a dark-and-bright pattern in the mandibular region wherever abrupt changes on the object's surface occurs (Figure 10c). It appears to be an aliasing



(a)

(b) (c)

Figure 8: SR images of object O_3 (CT skull) by splatting (a-b) one voxel on to one pixel and (c) one voxel onto at most 3×3 pixels.

artifact, but it is not. In fact, the problem may appear in SR whenever the principal axis is not indexed at the end, and it does not occur in SWSR as shown in Figure 10d. In this sense, in reality SWSR and SWR offer better image quality than SR, because they use the voxel indexing order as indicated in Table 2.

Apart from the problem mentioned above, SR and SWSR should provide similar renditions. Figures 11- 14 present several examples providing the comparison of image quality between SWSR and SWR. Unfortunately, it was impossible to produce precisely the same viewing direction and display size for both methods, because users do not have full control over these parameters in Lacroute's implementation. Nevertheless, they are enough to demonstrate that both methods can produce comparable high-quality images.

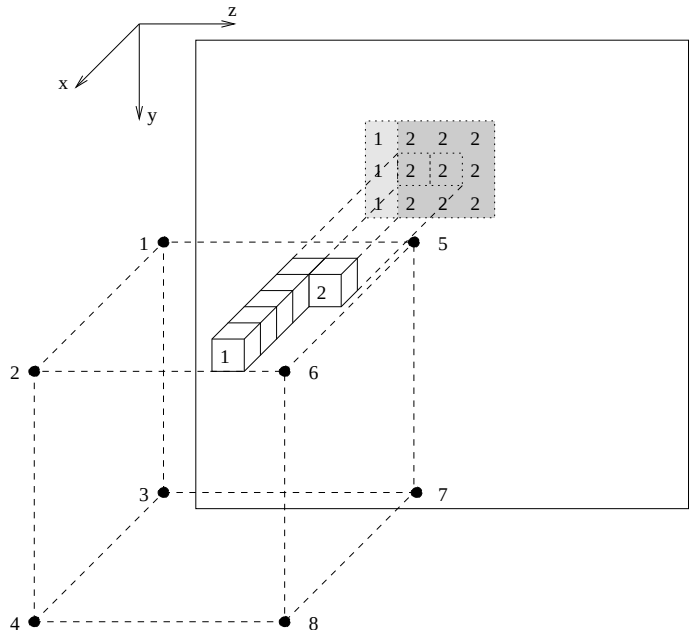


Figure 9: Splatting of one voxel onto at most 3×3 pixels when the observer is in octant 6, x is perpendicular to the viewing plane, and the voxel indexing order is as indicated for octant 6 in Table 1.

4 CONCLUSIONS AND DISCUSSION

We have proposed a new method for volume visualization, called shear-warp shell rendering (SWSR), and presented a comparative analysis of SWSR and the two other related methods - shell rendering (SR) and shear-warp rendering (SWR). SWSR is a variant of SR which allows speed up gains by shear-warp factorization of the viewing matrix, a mathematical property also exploited in SWR, without paying the high storage price of SWR.

We have chosen 10 different objects of various sizes, shapes and topologies and one 1GHz Pentium-III PC with 512MB RAM for our experiments. Hard and fuzzy boundaries of up to 2,833K voxels in size have been created to test the methods in surface and volume rendering, respectively. The results show that SWSR is 1.35 times faster than SR and SWR for surface rendering, and for volume rendering, SWSR is 1.52 times faster than SR and almost as fast as SWR. In surface rendering, SWSR requires 2 times more memory space than SR, but it saves 71% of memory space when compared to SWR. In volume rendering, SWSR requires 1.8 times more memory space than SR and saves 34% of memory space with respect to SWR. Even considering the recent work, which reduces to 2/3 the memory space of SWR [40], SWSR still provides less storage costs than SWR. Another problem of SWR is the sequential access to the voxels in the run-length lists. This may be efficient for visualization, but it is not for data manipulation. In this sense, SWSR and SR provide faster access to particular voxels in the shell. SWSR also provides better image quality than SR and high-quality renditions comparable with SWR. In addition to the fact that surface SR has proven to be one-order of magnitude faster than polygonal methods [29], we may conclude that SWSR is the best option for surface and volume rendering modes at present.

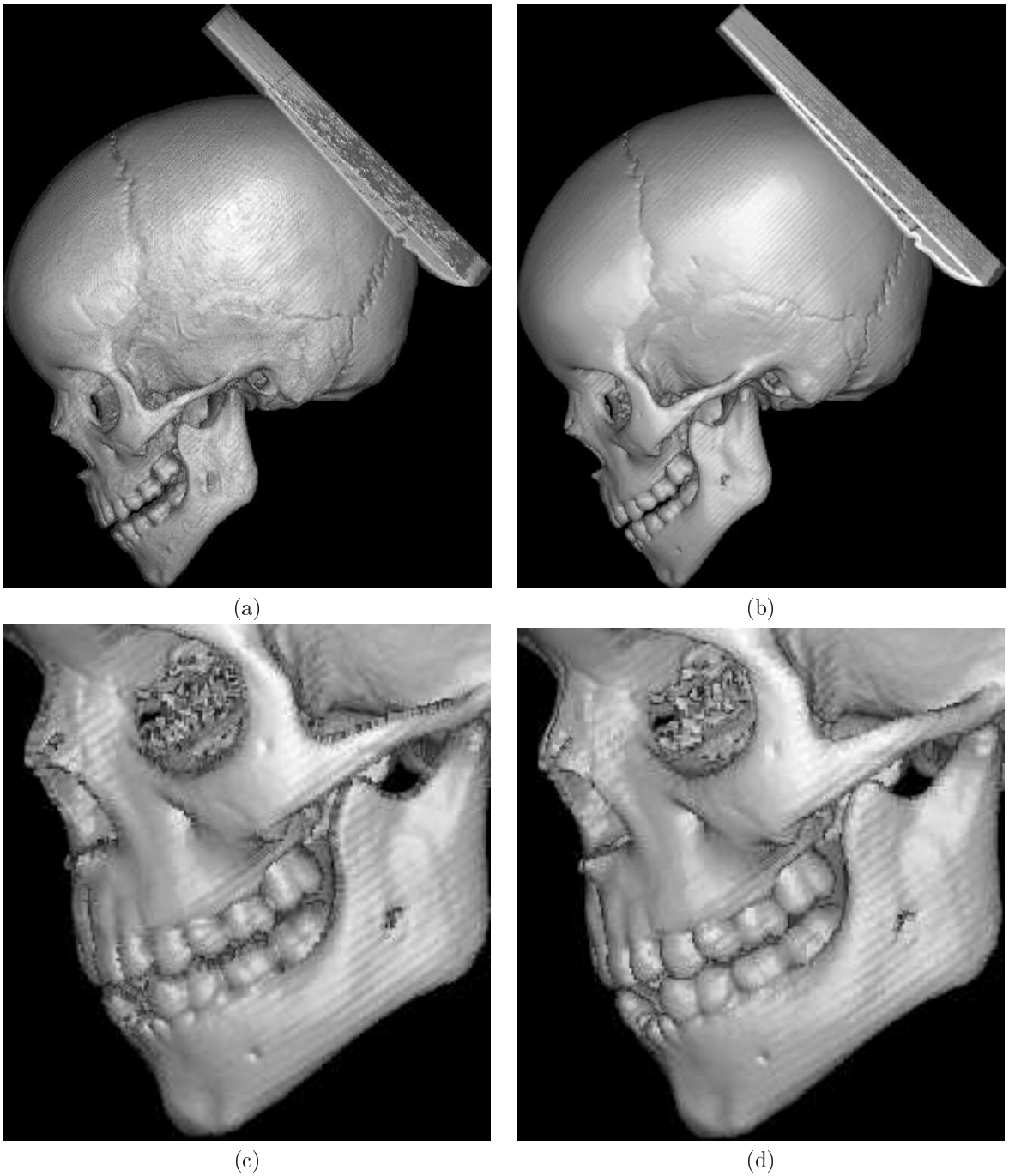
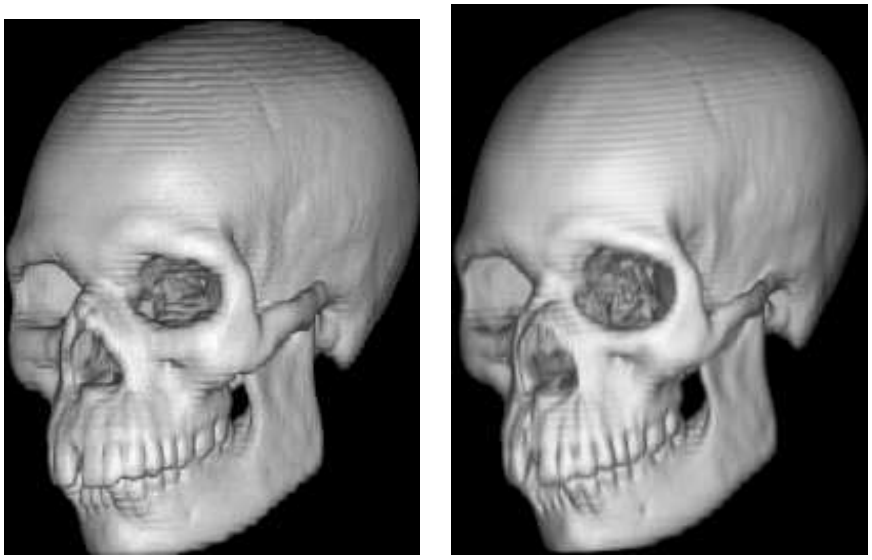
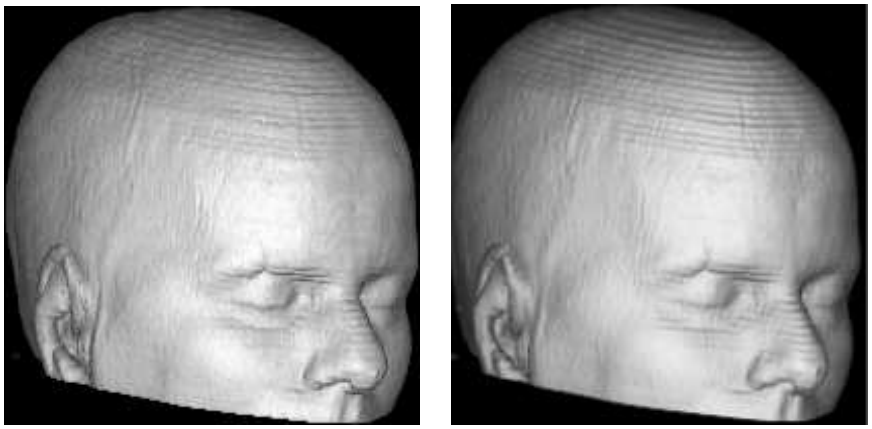


Figure 10: Image renditions of object O_3 (CT child skull) by splatting one-voxel on to at most 3×3 pixels. (a-b) SR and SWSR renditions for lower-opacity values, respectively. (c-d) SR and SWSR renditions for higher-opacity values, respectively.



(a) (b)

Figure 11: Image renditions of object O_3 (CT skull) by (a) SWSR and (b) SWR.



(a) (b)

Figure 12: Image renditions of object O_4 (MR head) by (a) SWSR and (b) SWR.

Acknowledgments

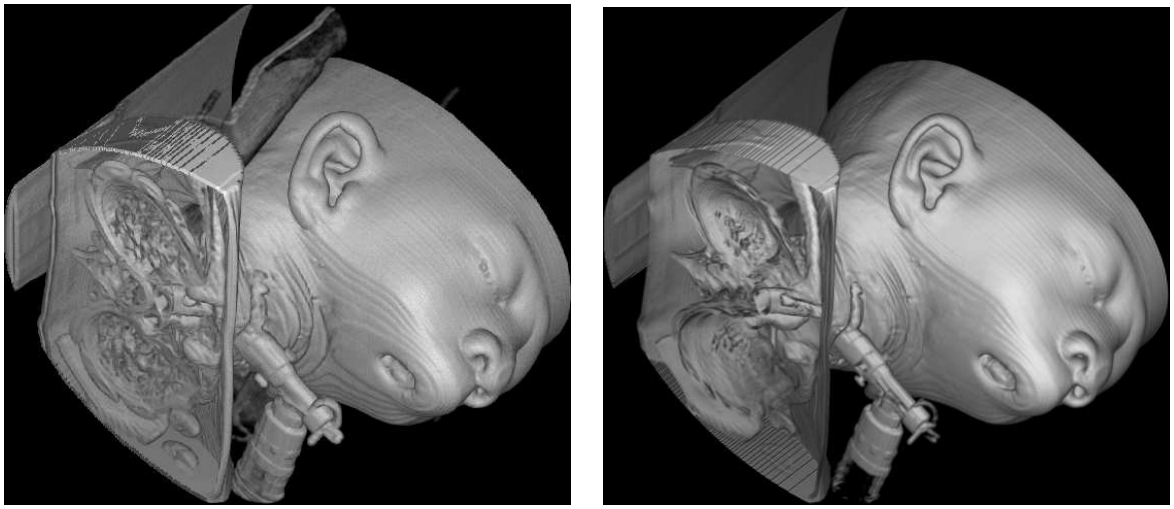
Falcão is grateful to CNPq (Proc. 300698/98-4) for the financial support. Rocha thanks CAPES for his scholarship. Udupa's work is supported in part by DHHS grants LM O-3502-MOD3, NS 37172 and AR46902.

References

[1] G.T. Herman and J.K. Udupa. Display of 3d-digital images: Computational foundations and medical applications. *IEEE Trans. on Computer Graphics and Applications*, 3(5):39–46, Aug

(a) (b)

Figure 13: Image renditions of object O_9 (CT head and spine - bones) by (a) SWSR and (b) SWR.



(a) (b)

Figure 14: Image renditions of object O_{10} (CT head and spine - skin) by (a) SWSR and (b) SWR.

1983.

[2] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (In Proc. of ACM SIGGRAPH)*, 21(4):163–169, Jul 1987.

[3] M.J. Durst. Letters: Additional reference to marching cubes. *Computer Graphics (In Proc. of ACM SIGGRAPH)*, 22(2):72–73, Apr 1988.

[4] I. Gargantini and H. Atkinson. Multiple-seed 3D connectivity filling for innacurate borders. *CVGIP: Graphical Models and Image Processing*, 53(6):563–573, 1991.

[5] J.K. Udupa and D. Odhner. Shell rendering. *IEEE Computer Graphics and Applications*, 13(6):58–67, 1993.

[6] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

[7] K.R. Subramanian and D.S. Fussell. Applying space subdivision techniques to volume rendering. In *Proceedings of Visualization'90*, pages 150–159, San Francisco, CA, 1990.

[8] P. Lacroute. *Fast volume rendering using a shear-warp factorization of the viewing transformation.* PhD thesis, Dept. of Electrical Engineering and Computer Science, Stanford University, Stanford, CA, Sep 1995.

[9] J.K. Udupa and G.T. Herman. *3D Imaging in Medicine.* CRC Press, Boca Raton, Florida, second ed. edition, 2000.

[10] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *IEEE Visualization*, pages 255–262, San Diego, CA, Oct 2001.

[11] G. Kindlmann and J.W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE Symposium on Volume Visualization*, pages 79–86, Research Triangle Park, NC, Oct 1998.

[12] E. Stindel, J.K. Udupa, B.E. Hirsch, and et al. An in vivo analysis of the motion of the peri-talar joint complex based on mr imaging. *IEEE Transactions on Biomedical Engineering*, 48(2):236–247, Feb 2001.

[13] A.X. Falcão, J.K. Udupa, and F.K. Miyazawa. An ultra-fast user-steered image segmentation paradigm: Live-wire-on-the-fly. *IEEE Transactions on Medical Imaging*, 19(1):55–62, Jan 2000.

[14] A.X. Falcão and J.K. Udupa. A 3D generalization of user-steered live wire segmentation. *Medical Imaging Analysis*, 4(4):389–402, Dec 2000.

[15] P.K. Saha and J.K. Udupa. Relative fuzzy connectedness among multiple objects: theory, algorithms, and applications in image segmentation. *Computer Vision and Image Understanding*, 82:42–56, 2001.

[16] T. McInerney, G. Hamarneh, M. Shenton, and D. Terzopoulos. Deformable organisms for automatic medical image analysis. *Medical Image Analysis*, 6(3):251–266, Sep 2002.

[17] G. Thurmer and C.A. Wuthrich. Normal computation for discrete surfaces in 3D space. In *Computer Graphics Forum (Proc. of Eurographics)*, pages 15–26, 1997.

[18] J.K. Zuiderveld, A.H.J. Koning, and M.A. Viergever. Acceleration of ray-casting using 3d distance transforms. In *Proceedings of Visualization in Biomedical Computing*, pages 324–335, Chapel Hill, North Caroline, Oct 1992.

[19] M. Bentum. *Interactive visualization of volume data*. PhD thesis, Dept. of Electrical Engeneering, University of Twente, Enschede, The Netherlands, Jun 1996.

[20] H. Pfister, J. Hardenbergh, Knittel J., H. Lauer, and L. Seiler. The volumepro real-time ray-casting system. *ACM Computer Graphics (In Proceedings of SIGGRAPH'99)*, pages 251–260, Aug 1999.

[21] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6(1):2–7, 1990.

[22] J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Proceedings of the 1992 Workshop on Volume Rendering*, volume 19, pages 91–98, 1992.

[23] G. Frieder, D. Gordon, and R.A. Reynolds. Back-to-front display of voxel-based objects. *IEEE Trans. on Computer Graphics and Applications*, 5(1):52–60, Jan 1985.

[24] D. Gordon and R.A. Reynolds. Image space shading of 3-dimensional objects. *Computer Vision, Graphics and Image Processing*, 29:361–376, 1985.

[25] L. Westover. Footprint evaluation for volume rendering. In *Proc. of SIGGRAPH*, pages 367–376, 1990.

[26] J. Huang, K. Mueller, N. Shareef, and R. Crawfis. Fastsplats: Optimized splatting on rectilinear grids. In *IEEE Visualization*, pages 219–227, Salt-Lake City, Oct 2000.

[27] M. Zwicker, H. Pfister, J. van Barr, and M. Gross. Ewa splatting. *IEEE Visualization and Computer Graphics*, 8(3):223–238, 2002.

[28] G.G. Cameron and P.E. Undrill. Rendering volumetric medical image data on a SIMS-architecture computer. In *Proceedings of the Third Eurographics Workshop on Rendering*, pages 135–145, Bristol, UK, May 1992.

[29] G.J. Grevera, J.K. Udupa, and D. Odhner. An order of magnitude faster isosurface rendering in software on a PC than using dedicated, general purpose rendering hardware. *IEEE Transactions on Visualization and Computer Graphics*, 6(4):335–345, Oct-Dec 2000.

[30] G.P. Carnielli, A.X. Falcão, and J.K. Udupa. Fast digital perspective shell rendering. In *XII Brazilian Symposium on Computer Graphics and Image Processing*, pages 105–111, Campinas, SP, Brazil, Oct 1999.

[31] G.J. Grevera, J.K. Udupa, and D. Odhner. T-shell rendering. In *Proceedings of SPIE on Medical Imaging: Visualization, Display, and Image-Guided Procedures*, volume 4319, pages 413–425, Feb 2001.

[32] P. Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):218–231, Sep 1996.

[33] A.E.F. Schmidt, M. Gattass, and P.C.P. Carvalho. Combined 3d visualization of volume data and polygonal models using a shear-warp algorithm. *Computers & Graphics*, 24(4):583–601, Aug 2000.

[34] B.H. Kim, J. Seo, and Y.G. Shin. Binary volume rendering using slice-based binary shell. *The Visual Computer*, 17(4):243–257, 2001.

[35] T.Y. Kim and Y.G. Shin. Fast volume rendering with interactive classification. *Computer & Graphics*, 25(5):819–831, Oct 2001.

[36] D.M. Jiang and J.P. Singh. Improving parallel shear-warp volume rendering on shared address space multiprocessors. *ACM SIGPLAN NOTICES*, 32(7):252–263, Jul 1997.

[37] W.S. Edwards, C. Deforge, and Y. Kim. Interactive three-dimensional ultrasound using a programmable multimedia processor. *International Journal of Imaging Systems and Technology*, 9(6):442–454, 1998.

[38] W.L. Cai and G.B. Sakas. Maximum intensity projection using splatting in sheared object space. *Computer Graphics Forum*, 17(3):113–124, 1998.

[39] H.S. Kang, B.H. Kim, J.W. Ryu, S.H. Hong, H.W. Chung, S.Y. Cho, Y.H. Kim, S.I. Hwang, D.K. Jeong, and Y.G. Shin. The visible man: Three-dimensional interactive musculoskeletal anatomic atlas of the lower extremity. *Radiographics*, 20(1):279–286, Jan-Feb 2000.

[40] J. Sweeney and K. Mueller. Shear-warp deluxe: the shear-warp algorithm revisited. In *Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 95–104, Barcelona, Spain, May 2002.

[41] A.X. Falcao, L.M. Rocha, and J.K. Udupa. A comparative analysis of shell rendering and shear-warp rendering. In *Proceedings of SPIE on Medical Imaging 2002*, volume 4681, pages 472–482, San Diego, CA, Feb 2002.

[42] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics (In Proc. of ACM SIGGRAPH)*, 22(4):65–74, Aug 1988.

[43] D.H. Laidlaw, K.W. Fleischer, and A.H. Barr. Classification of material mixtures in volume data for visualization and modeling. Technical Report CS-TR-94-07, California Institute of Technology Computer Science Department, 1994.

[44] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, New York, NY, 1991.

[45] J.K. Udupa and D. Odhner. Fast visualization, manipulation, and analysis of binary volumetric objects. *IEEE Computer Graphics and Applications*, 11(6):55–62, 1991.