E-Shell Rendering

Leonardo M. Rocha and Alexandre X. Falcão

Institute of Computing - University of Campinas
Av. Albert Einstein, 1251, CEP 13084-851, Campinas, SP, Brasil.
Phone: 55-19-37885881, FAX: 55-19-37885847 {leonardo.rocha, afalcao}@ic.unicamp.br.

Abstract. In Medical Imaging, shell rendering is considered one of the most efficient and effective methods for volume visualization. It requires a compact data structure of voxels and uses the voxel splatting technique for surface and volume rendering. In order to avoid holes in the rendition, the size of the voxels are usually made bigger than the size of the pixels for voxel splatting. In such a case, we have identified a conceptual problem in shell rendering which affects the correctness and quality of the renditions. In this paper, we discuss this problem and propose a solution, which improves image quality without affecting the speed of the method. The new approach is called *E-Shell Rendering* (extended-shell rendering). It requires an extension of the shell data structure and a variant of the original algorithm. We illustrate and discuss the results with examples created by surface and volume rendering.

1 Introduction

Volume visualization consists of three important steps - classification, representation, and rendering. Classification is the process of assigning opacities to objects proportional to the degree of interest in these objects for visualization. The most difficult 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 objects and a data structure to store the parameters of 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 volume and determining the light that reaches the view point to obtain different views of the selected objects.

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

In spite of the inherent difficulties in the classification process, visualization at interactive speeds is mandatory. This is often a challenge, because the amount of information for visualization often requires extensive storage and time-intensive computations. Researchers have proposed specialized hardware [7, 8] and algorithms that trade off image quality for speed [9, 10] in order to overcome the speed problem. 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. Moreover, considering that more than 85% of the voxels are usually classified as transparent, the most interesting approaches to address the speed problem use spatial data structures that encode the presence and/or absence of high-opacity voxels to eliminate computation in transparent regions of the volume [4, 5, 6, 7, 11]. Among these approaches, one of the most successful techniques has been *shell rendering* [4].

Shell rendering uses a compact data structure, called *shell*, based on a list of non-transparent voxels and a 2D array of pointers to that list. It provides surface and volume rendering with the same data structure. The voxels in the shell are usually considered bigger than the pixels in the image for voxel splatting,

in order to avoid holes in the rendition. Surface shell rendering has proved to be from 18 to 31 times faster on a 300MHz Pentium PC than software-based and hardware-based polygonal rendering methods, including those based on very expensive machines such as the Silicon Graphics Reality Engine II [12]. Shell rendering has also been extended to provide digital perspective [13] and to render polygonal geometric models within the same framework [14].

In this paper, we point out and discuss a conceptual problem of shell rendering when the size of the voxels are made bigger than the size of the pixels for voxel splatting. We propose an extension of the compact shell data structure and a variant of the original algorithm to solve that problem. The new approach is called *E-Shell Rendering* (extended-shell rendering). We show that e-shell rendering improves the image quality without affecting the speed of the method.

The paper is organized as follows. Section 2 describes the methods and points out their major differences in the computational aspects. The results are illustrated and discussed with examples created by surface and volume rendering in Section 3, and the conclusions are stated in Section 4.

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 arbitrary value space. A scene is usually stored as a finite 3D retangular 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 [15].

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. Note that segmentation is the process of identifying such a subset. The aim of classification is to transform (V, f) into another scene (B, o) by assigning an opacity value $o(v)$ lying in the interval $[0, 1]$ to each voxel v in B.

There are many methods of voxel classification [16, 17]. In this paper, we shall assume classification is already done by one of these approaches such that the fuzzy/hard boundary is the same for both visualization techniques. Shell rendering [4, 13] support orthogonal and perspective projection, but we will concentrate our analysis on orthogonal projection.

2.1 Shell rendering

Given a scene, shell rendering uses a compact data structure, called *shell*, to encode only non-transparent voxels of the bondary. Each voxel is represented by an opacity value in $(0, 1]$, its x coordinate in the scene and some visualization and manipulation attributes. The 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 voxels in the scene. A 2D pointer array P_{yz} indicates the first voxel in the list V_x associated with a particular coordinate (y, z) in the scene. Figure 1 illustrates a simple example of a shell for a scene of size $3 \times 3 \times 3$ voxels. Figure 1a shows the scene where each voxel is numbered from 1 to 27, following the x-by-x, y-by-y, and z-by-z order. The voxels of the shell (i.e. the non-transparent voxels) are only those numbered from 1 to 9. Figure 1b shows the list V_x and the pointer array P_{yz} for this example.

Fast orthogonal voxel projection is possible from front-to-back by just sweeping the pointer array and the list of voxels in eight different ways. Shell rendering 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 x, it is faster to project them onto the viewing plane by going along x before going along y and z. Figure 4 illustrates a scene where each octant has an identification number from 1 to 8. The indexing order will depend on what octant is the viewing direction. Table 1 shows the front-to-back indexing order for each octant in Figure 4, 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.

A well known problem of visualization methods based on voxel projection is the apparency of holes in the rendered image for certain viewing directions (see Figure 2a). Post-processing algorithms may be used to "close that holes", but their success is not guaranteed and the resulting image quality may not be good. A more effective and efficient solution is to make the size of the voxels bigger than the size of the pixels. Figure 2b shows the result of voxel splatting for the same viewing direction shown in Figure 2a. In such a case we have identified a conceptual problem in shell rendering, which affects the quality of the renditions.

Shell rendering exploits the property of non-preference order among x, y, and z for fron-to-back orthogonal projection to increase efficiency by indexing the axis x before y and z (see Table 1). Unfortunately, this property is no longer valid when splatting voxels bigger than pixels. Figure 3 illustrates an example where the observer is in octant 6, the voxel

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 shell rendering for each octant in Figure 4, 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, for a front-to-back projection.

indexing order is as indicated in Table 1, the axis x is perpendicular to the viewing plane, and each voxel paints 3×3 pixels. Note that farther and hidden voxels are painted in place of closer and non-hidden voxels. In other words, shell rendering does not perform hidden-surface removal correctly, and the same problem may also occur in other visualization methods which exploit the same property. In the case of surface rendering, the problem still has a solution if we use the z-buffer for depth-sorting, but unfortunately it remains unsolved for volume rendering.

The effective solution for the problem requires the determination of *the principal viewing axis* - the axis most perpendicular to the viewing plane among the axes x, y, and z. That is, front-to-back voxel splatting requires the principal axis to be indexed at last, as indicated in Tables 2a-c. However, this creates another problem in shell rendering, because its efficiency is only guaranteed when the axis x is indexed before the others. We present an effective solution to the problem in the next section.

2.2 E-shell rendering

Suppose the voxel indexing order in shell rendering is as indicated in Table 2a (when the principal axis is x). Note that, for each x coordinate in the scene, we need a binary search in the list V_x for each (y, z) coordinate in the pointer array P_{yz} to verify the existence of a voxel with the (x, y, z) coordinate in the shell. In the worst case, the number of binary searches is the size of the scene, which is usually 85% more than the size of the shell, making shell rendering impractical.

We solve the problem by adding to the original shell data structure a second 2D pointer array P_{xz} and a second list V_y of voxels (see example in Figure 1c). By indexing the scene, starting from $(x, y, z) = (0, 0, 0)$, in a y-by-y, x-by-x, and z-by-z order, we store in V_y the y coordinate of each voxel of the shell and a pointer to its corresponding position in the first list V_x. Similarly, each pointer in P_{xz} indicates the first voxel in V_y associated with a particular (x, z) coordinate in the scene. Note that, the other attributes for visualization and manipulation are stored only in the first list V_x. Thus, we use P_{xz} and V_y whenever the principal axis is x. This extended shell and the new algorithm which takes into account the voxel indexing order, as indicated in Tables 2a-c, form the proposed visualization method called *E-Shell Rendering*.

Observe that, the e-shell requires more memory space than the original shell, but this is acceptable since the later is a very compact data structure. Note also that, the speed of both methods is practically the same. Therefore, we need only to compare them in terms of image quality. This issue is addressed next.

3 Results

In practice, the conceptual mistake of shell rendering is very difficult to be detected, 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, but there are abrupt changes on the object's surface.

Figure 5a shows an image created by volume shell rendering with lower-opacity values. Note that hidden voxels appear as dark stains on the object's surface. Figure 5b illustrates that this problem does not happen in volume e-shell rendering, where the same view of the object with the same lower-opacity values is presented. Figure 6a illustrates the other situation when voxels are opaque. In this case, we observe in surface shell rendering a dark-and-bright pattern wherever occurs abrupt changes on the object's surface (i.e. the manibular region and among the teeth). It looks like aliasing, but it is not. As one can observe for the same view of the opaque object presented in Figure 6b, the pseudo-aliasing pattern does not appear in surface e-shell rendering.

Figures 7a-d present other examples which show that the image quality in e-shell rendering is superior than the rendition quality of shell rendering.

4 Conclusions and discussion

We proposed an extension of the shell rendering technique for volume visualization, called E-Shell Rendering, and presented a comparative analysis of them in terms of image quality. A conceptual mistake in shell

a)

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

b)

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

c)

The principal axis is z	
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 2: This table shows the voxel indexing order in e-shell rendering for each octant in Figure 4, 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.

rendering was pointed out when voxels are made bigger than pixels for voxel splatting. We showed that this mistake affects the correctness and quality of the renditions. The results showed that e-shell rendering corrects the problem and provides better image quality than shell rendering.

We emphasize that the same conceptual problem may happen in other visualization methods which use voxel splatting, whenever the principal axis is not indexed at the end. In particular, the solution presented here for orthogonal projection is also valid for digital perspective projection [13].

References

[1] 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.

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

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

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

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

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] J.K. Udupa. 3D visualization of images. Technical Report MIPG196, Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Jun 1993.

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

[17] 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.

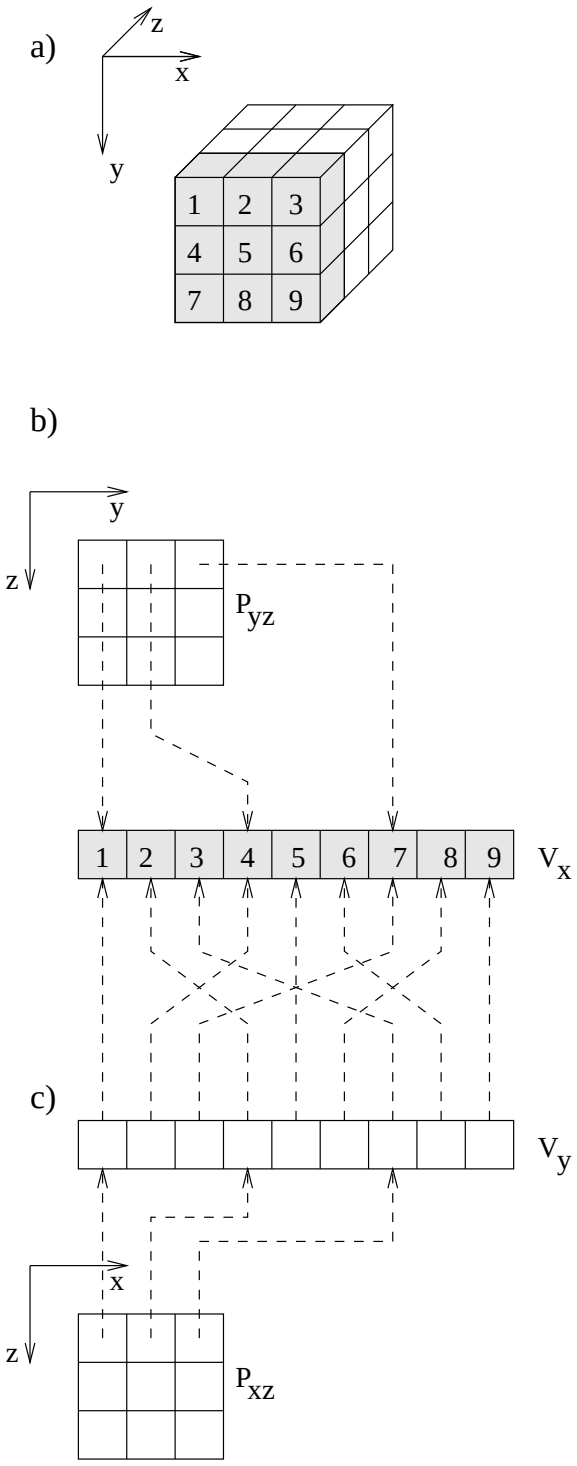
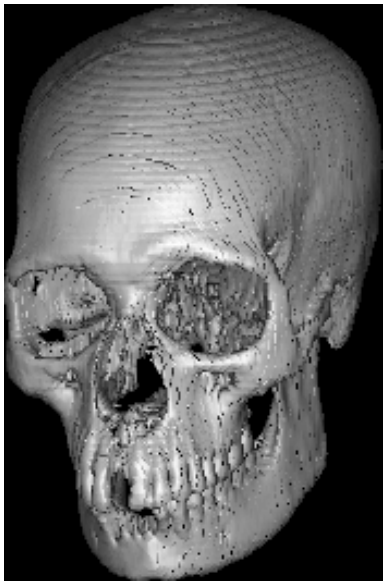
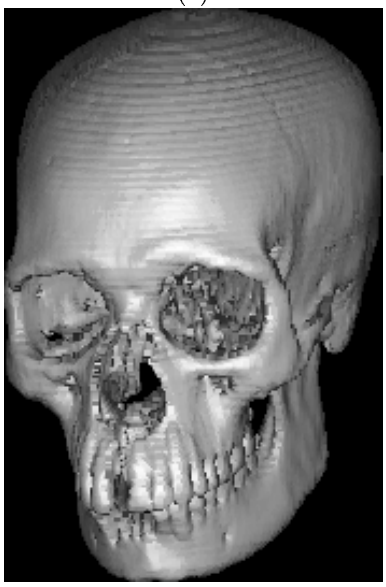


Figure 1: (a) A scene of size 3x3x3 voxels where the voxels are numbered from 1 to 27, following the x-by-x, y-by-y, and z-by-z order, and only those numbered from 1 to 9 are voxels of the shell. (b) The list V_x and the pointer array P_{yz} which represent the shell. (c) The list V_y and the pointer array P_{xz} which extend the original data structure presented in (b).



(a)



(b)

Figure 2: Surface shell rendering of a CT skull by splatting (a) one voxel on to one pixel and (b) one voxel on to 3×3 pixels.

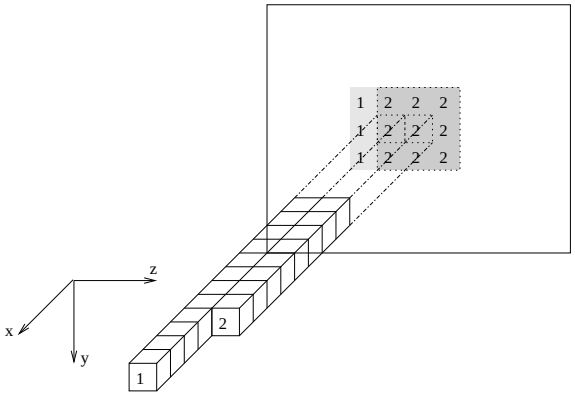


Figure 3: Splatting of one-voxel onto 3×3 pixels when the observer is in octant 6, x is perpendicular to the viewing plane, and the voxel indexing order is as indicated in Table 1.

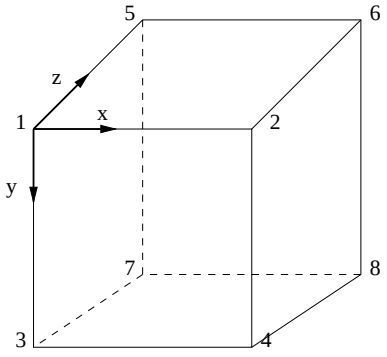
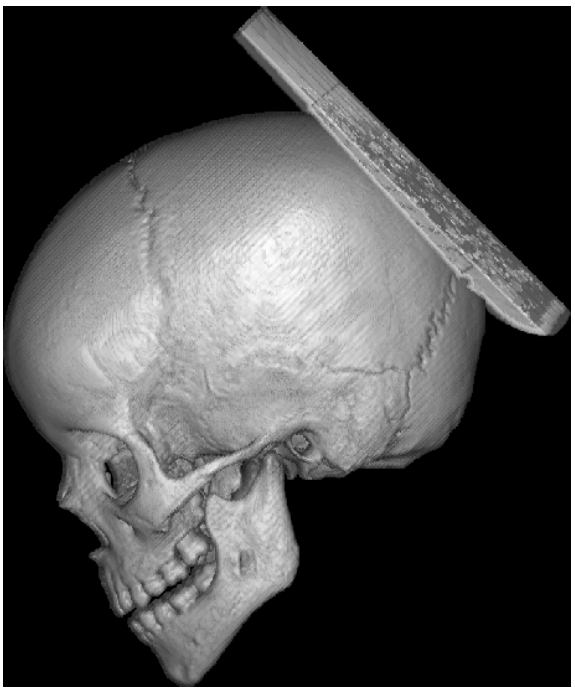
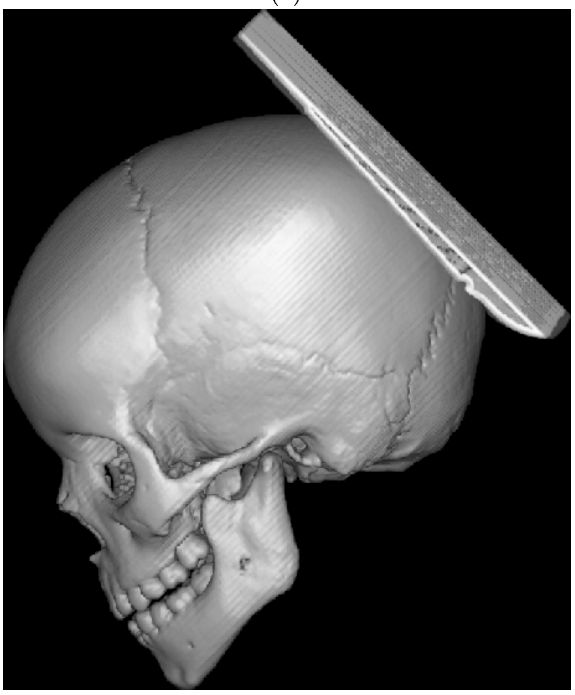


Figure 4: A scene where each octant has an identification number from 1 to 8.



(a)



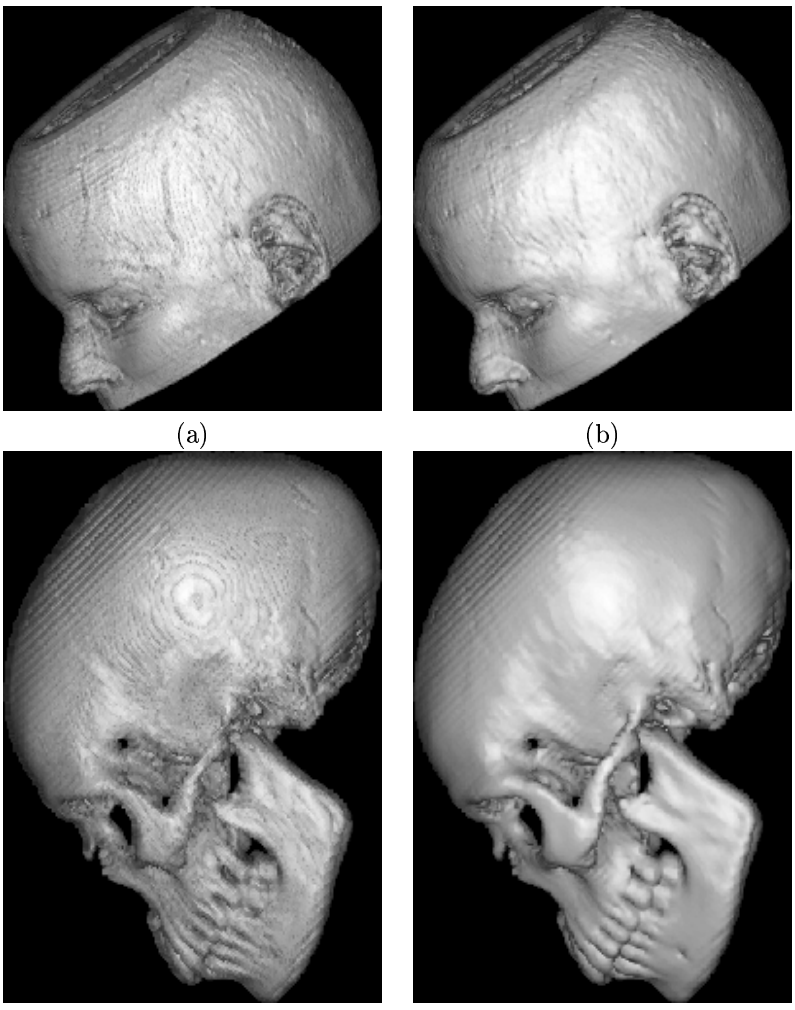
(b)

Figure 5: Image renditions of a CT child skull with voxel splatting. (a-b) Volume shell and e-shell renditions with lower-opacity values, respectively.

(a)

(b)

Figure 6: Image renditions of a CT child skull with voxel splatting. (a-b) Surface shell and e-shell renditions, respectively.



(a) (b)

(c) (d)

Figure 7: (a-d) present other renditions which show that the image quality in e-shell rendering is superior than the rendition quality of shell rendering.