A New Object-Order Ray-Casting Algorithm

Benjamin Mora, Jean-Pierre Jessel, René Caubet

Institut de Recherche en Informatique de Toulouse (IRIT), Université Paul Sabatier, 31062 Toulouse, France

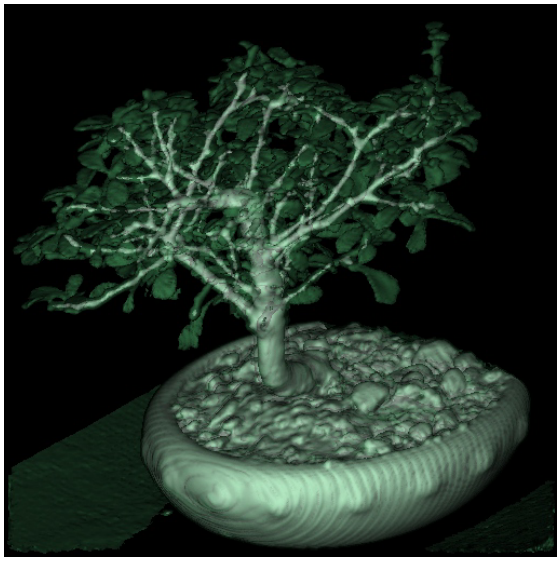


Figure 1: Two-pass volume rendering of a 256^3 bonsai dataset, visualized at approximately 3 frames per second.

ABSTRACT

Many direct volume rendering algorithms have been proposed during the last decade to render 256^3 voxels interactively. However a lot of limitations are inherent to all of them, like low-quality images, a small viewport size or a fixed classification. In contrast, interactive high quality algorithms are still a challenge nowadays. We introduce here an efficient and accurate technique called object-order ray-casting that can achieve up to 10 fps on current workstations. Like usual ray-casting, colors and opacities are evenly sampled along the ray, but now within a new object-order algorithm. Thus, it allows to combine the main advantages of both worlds in term of speed and quality. We also describe an efficient hidden volume removal technique to compensate for the loss of early ray termination.

CR Categories : I.3.3 [Computer Graphics]: Picture/Image Generation, I.3.7 [Computer Graphics]: Three-Dimensional graphics and realism, Raytracing, Visible line/surface algorithms.

Keywords : Volume Rendering, Scientific Visualization, Medical Imaging, Ray Tracing.

1. INTRODUCTION

Volume visualization has been widely studied over the last decade due to the expansion of scientific devices producing such data. Many algorithms have been developed, and some of them are regrouped under the category of direct volume rendering (DVR) methods in which the whole original dataset is used for the rendering without any intermediate representation of the volume. In DVR algorithms, the interaction between rays traced from the viewpoint and the volume is studied, which allows high quality images and a great freedom of action. Several models of interaction are widely used nowadays, like the maximum intensity projection (MIP) model that returns the maximum value encountered along the ray, or the accumulation model that integrates the signal. Here, we will only focus on optical models [10, 16] that are common to many volume rendering applications, even if our algorithm can be easily extended to other models. The Kajiya's optical model in its low albedo form is given by:

$$I_\lambda = \int_0^l C_\lambda(s) \times \tau(s) \times \exp\left(-\int_0^s \tau(t)\, dt \right) ds \qquad (1)$$

Where I_λ is the amount of light of wavelength λ along the ray reaching the viewpoint. The contribution of the ray at the location s is given by $C_\lambda(s)$ weighted by the extinction coefficient $\tau(s)$ and by the percentage of occlusion that depends on the opacity between the viewpoint and s. However this integral cannot be evaluated as it is, and a Riemann sum is often used to approximate it. This way, rays are usually evenly sampled with a distance Δs, and the accumulated color (C_i) and opacity (α_i) are estimated with the recursive process given below (front-to-back order):

$$C_{i+1} = C_i + (1-\alpha_i)\, \alpha_s C_s \qquad (2)$$
$$\alpha_{i+1} = \alpha_i + (1-\alpha_i)\, \alpha_s$$

It is obvious that the accuracy of the integral estimation directly depends on the distance Δs and on the evaluation of the values α_s and C_s. A large sampling distance can accelerate the rendering times, but on the other hand it provides low quality images. Furthermore the sampled values α_s and C_s have to be estimated from the discrete volume data with a reconstruction filter. Thus the choice of the reconstruction filter is crucial and nowadays only the trilinear filter and the Gaussian filter, usually considered as reasonable quality filters, can perform direct volume rendering in acceptable times, even if studies [15] have shown that a better quality can be obtained with more complex filters. Then a good volume rendering application should provide the best compromise between quality and speed.

In this paper we will intend to give such a trade-off by using an efficient approach to compute ray-casting. Usually, trilinear interpolation is made easier with this algorithm, but the pixel-by-pixel approach (also called image-order) drastically slows down the rendering process, even if the hidden regions of the volume are not processed. On the opposite, the projection approaches (also called object-order) are well-suited for skipping empty regions, but the usually associated filters are either low-quality filters or too complex to be interactive, and hidden volume removal is not very efficient. Our approach combines the advantages of both approaches to produce interactive high-quality volume rendering.

First, in section 2, we will look at today's mostly used techniques and we will try to explain the strengths and

{mora, jessel}@irit.fr

weaknesses of each approach. Then, in section 3, we will describe the new object-order ray-casting algorithm. Finally, we will show in section 4 that efficient hidden volume removal is possible with it before giving our results in section 5.

2. PREVIOUS WORK

Four main software algorithms have emerged in the last decade [18] and are widely used: the Shear-Warp method and the Hardware-assisted 3D texture mapping techniques that are speed oriented, and the quality oriented ray-casting and splatting algorithms.

Shear-Warp [12] is currently considered as the fastest software algorithm. It offers many optimizations probably allowing unbeatable rendering times. This object-order method considers the volume as a stack of 2D slices parallel to the face of the volume the most perpendicular to the view axis. Slices are accumulated on an intermediate image that undergoes an ultimate resampling step to produce the final image. The intermediate image is aligned with the slices and has the same pixel density, allowing both the volume and the image to be run in an efficient memory order, and to perform fast projection (i.e. one voxel is projected on one pixel). To improve quality, a bilinear interpolation is made for every projection with constant coefficients within a slice. Finally an efficient pre-classified run-length encoding of the volume allows skipping quickly empty regions.

However, regarding quality, this algorithm has many drawbacks. First, the sampling rate on the z-axis is between 1 and 1.73 according to the viewpoint, which is definitely not enough for the observation of thin volume structures. The pre-classification partially blurs the intermediate image, which is increased by the final resampling step. Finally, artifacts occur when the viewing angle is close to 45^o due to the bilinear interpolation. Thus, the global quality provided by the original implementation turns out to be poor.

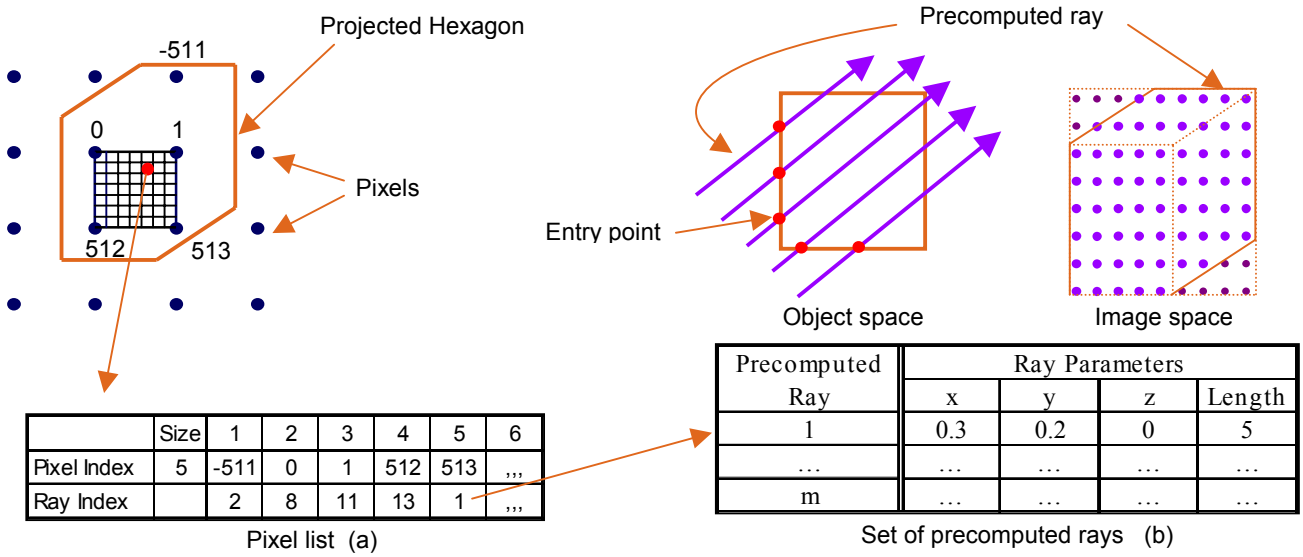
A solution to these drawbacks is to use trilinear interpolation, post-classification and supersampling, as implemented in the VolumePro board [26]. This PCI board can render 500 million interpolated samples per second with a brute-force Shear-Warp algorithm (parallel projection), which is sufficient to render 256^3 volumes at 30 frames per second. Supersampling can be computed on hardware in the z-direction and on software in the x and y directions by rendering several images at different offsets. However supersampling divides the frame rate by the number of samples per voxel, and then if the sampling rate along the 3 axes is doubled to produce high quality images, the frame rate can decrease to under 4 frames per second. Furthermore, applying these improvements on the original algorithm should also reduce greatly its performances. Thus, real time high quality volume rendering should not be really possible yet by using a shear-warp algorithm.

Another popular way to perform interactive volume rendering is to use 3D texture mapping hardware [1, 2, 4, 17, 29] by extracting and compositing 2D planes parallel to the image plane, but until recently the proposed approaches had several limitations, like binary classification or diffuse shading only. Engel and al. [3] have used the *NVidia*'s OpenGL extension available with the new *Geforce3* graphics hardware to circumvent these drawbacks. Although real-time rendering rates are possible on small volumes ($<128^3$) with the current available implementation, it does not exceed 2 frames per second for 256^3 volumes due to the high memory bandwidth needed. Furthermore the large distance (Δs=1) between the extracted slices can miss small details or produce artifacts in spite of the very interesting pre-integrated classification used. Thus the main advantage of this approach is to provide constant and at least interactive perspective renderings with a reasonable quality, even if the flexibility of the 3D texture-based methods is still low.

Like the 3D texture-based algorithms, splatting has also come to maturity over the years since the first release proposed by Westover [31]. A Gaussian filter is usually associated with this method, but its use requires several refinements within the algorithm. In the latest versions proposed by Mueller et al. [23, 24], slices are extracted parallel to the image plane and combined with a sheet buffer in a front-to-back order. Post-classification and post-shading can also be done to improve the image quality. The main drawback is that this algorithm is rather slow and even an optimized release [8] needs several seconds to render an isosurface. However, the global quality that can be obtained with this filter is different from, but not really superior to, the one obtained with a trilinear filter [18]. While this latter produces more aliasing, the Gaussian filter is a low-pass filter blurring small details within the volume. Thus it seems that splatting with a gaussian kernel is not interesting if compared with trilinear interpolation because the larger kernel size limits its efficiency.

Ray-casting is another way to produce good quality images because trilinear interpolation can easily be implemented, although other filters can also be used [21]. Another great advantage is the incoherency between the rays that reduces greatly the *staircase* artifacts visible in algorithms extracting 2D planes, like with 3D texture-based DVR. Last but not least, early ray termination avoids the hidden regions to be treated. However, this algorithm is naturally slow because it is a pixel-by-pixel approach. Every time the ray steps forward within the volume, eight samples have to be loaded before performing trilinear interpolation. This creates cache misses because the samples cannot be stored in memory order. Furthermore, other rays rerunning the same cell may not take advantage of the preloaded data in the cache because the cache lines are often replaced by other data. On the contrary, the object-order methods access the voxels a limited number of times and produce traffic on the frame buffer, which is preferable because the image caching is generally better and easier than the volume caching. The second drawback is the difficulty to skip empty regions of the volume, especially when interactive classification is needed.

Several acceleration techniques have been proposed to reduce rendering times. Yagel and Kaufman [34] proposed the use of the same template ray to accelerate volume traversal. Sobierajski and Avila [37] proposed a two-step method. First, boundary cells are projected on the image plane using graphics hardware to identify more precisely the relevant parts of the rays. Then, a standard ray-casting is used. Interactive renderings are possible [28] for 256^3 volumes with 256^2 rays. However as this method did not allow efficient interactive classification, Westermann and Sevenich [30] used 3D textures instead of boundary cells to estimate the start of the ray traversal. Although those methods are efficient for isosurface renderings, they do not address the problem of the vertex loading and then are not efficient in other cases. Thus, Knittel [11] proposed the interleaving of voxel addresses, like many hardware implementations do, and deep optimizations to improve cache hits. Tile-casting was also used to take advantage



	Size	1	2	3	4	5	6
Pixel Index	5	-511	0	1	512	513	,,,
Ray Index		2	8	11	13	1	,,,

Pixel list (a)

Precomputed Ray	Ray Parameters			
	x	y	z	Length
1	0.3	0.2	0	5
…	…	…	…	…
m	…	…	…	…

Set of precomputed rays (b)

Figure 2: Lists needed for efficient ray-cell intersections and ray sampling.

of the ray coherency in space. Nonetheless, the acceleration data structure used for empty space skipping has to be recomputed to get optimal rendering times when the classification is changed, which limits the interactivity. Interactive rendering rates are possible on a small perspectives image (256^2) with a bi-PIII clocked at 500 MHz.

Thus, the current image-order ray-casting implementations have several gaps limiting the rendering times. The new object-order ray-casting we are looking at here tries to take advantage of both the quality produced by trilinear interpolation and the efficiency of the object-order methods (optimized volume run and easy space leaping). Thus, it is suitable for high-quality interactive volume rendering. We also propose a hidden volume removal algorithm to compensate for the loss of the early ray termination optimization inherent to image-order algorithms.

3. OBJECT-ORDER RAY-CASTING

The reconstruction filter employed in object-order algorithms is closely related to the projection technique used [32]. Until recently, it was limited to the projection of Gaussian kernels with the splatting algorithm, to the projection of one cell on one pixel with the shear-warp algorithm, or to the projection of the cell faces with polygons [36], which has serious drawbacks. Recent advanced methods [19, 22] have shown that the projection of a parallelepipedic shape can be efficiently performed when using an orthogonal projection. Therefore, it makes object-order ray-casting possible.

For every cell that has to be rendered (i.e. not transparent and not hidden), the pipeline is as follows: first, the values and gradients of the 8 vertexes are loaded. Then, for every ray intersecting the cell, colors and opacities are sampled along it before updating the equivalent pixel. The next sections describe the main parts of this pipeline.

3.1 Efficient Ray-Cell Intersection

When using orthogonal visualization, every cell projection on the image plane is given by the same template hexagon modulo a translation. The projection of the center of the cell is just able to inform us about this translation vector. Therefore, a square made of four neighboring pixels is subdivided and a list of relative coordinates corresponding to the projection of the cell is associated with each subdivision (*Pixel index* in fig. 2a). The rasterized pixels are simply given by the addition of the pixel indexes of the subdivision containing the projected center of the cell with the global coordinate of the square where the center is projected. Here, all the lists are precomputed every time the viewpoint change and the use of a linear indexing improves the efficiency. However, some erroneous pixels can be projected because the same list is used with all the cells so that the centers are projected within the same subdivision. Fortunately the probability is low and can be corrected by testing line equations. See [19] for a complete description about this implementation.

3.2 Evenly Spaced Sampling

The previous section has shown how to determine the rays going through a cell with precision. Now, the sequential process (2) estimating the line integral (1) must be performed for each ray in a front-to-back order to ensure the right evaluation. Thus, the intermediate luminance and density values needed in (2) are stored as 16 bits unsigned integers for every pixel. However, two main difficulties arise from this technique. First, the sampling coordinates within the cell needed for trilinear interpolation have to be computed quickly. Second, sampling points should be evenly spaced to evaluate precisely the Riemann sum. In order to make them possible, the algorithm uses both a set of preprocessed rays and a four-bits depth-sampling indicator aliased to the four lower bits of the pixel opacity.

The set of preprocessed rays (fig. 2b) is used to find out the ray entry point within the cell. Each element stores the 3 entry coordinates (x, y, z) as 16 bits unsigned integers representing values in the range [0..1], plus an additional ray length used for evenly spaced sampling. Thus, a ray number pointing to the best representative preprocessed ray is assigned to every pixel of the projection lists (*ray index* in fig 2a). Here, directly storing a preprocessed ray with every pixel would be less efficient because it would increase the table size. The set of rays is reconstructed

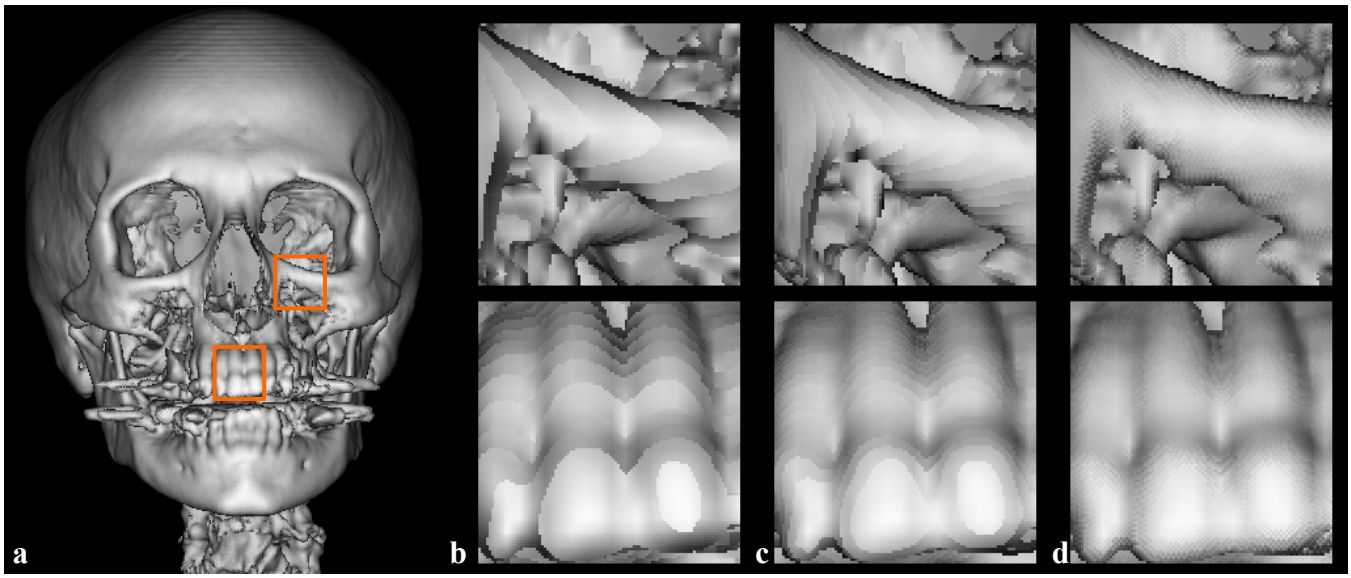


Figure 3: Different renderings with $\Delta s = 1$ (b), $\Delta s = 0.5$ (c) and $\Delta s = 0.5$ plus interleaving of the depth indicators (d).

every time the viewpoint changes by sampling the square surrounding the template hexagon (fig. 2b image space) and computing the ray parameters at each sampling location.

The depth-sampling indicator is now used to ensure a regular sampling of the ray between two consecutive cells. It determinates how many unit translation vectors (UTVs) must be added to the entry point to obtain the first sampling location. The next sampling locations of the ray within the cell are given by adding a constant translation vector depending on the sampling rate. The UTV is parallel to the viewing direction and it is defined as $1/16$ of the longest vector crossing the cell, which means that its coordinate on the major projection axis is also equal to $1/16$. The maximum number of UTVs that can be added to the entry point before going out of the cell gives the length parameter of the preprocessed rays. Thus, the basic algorithm to compute a ray/cell interaction is as follows:

```
Void Accumulate(RAY ray, PIXEL pixel, int sampling_rate) {
    int next_sampling=pixel.density & 0xF; //First sampling point
    While ( next_sampling < ray.Length)  {
            xyz=ray.xyz+next_sampling×UTV.xyz;
             Interpolate & Accumulate (xyz, pixel);
            next_sampling+=sampling_rate;//Next sampling point
    }       // Next sampling is out of the cell
    next_sampling -= ray.Length;
    pixel.density=(pixel.density & 0xFFF0)+next_sampling;
}
```

Here, *sampling_rate* is an arbitrary value within the range [1..16]. We point out the fact that while cells are correctly traversed by rays, the sampling positions (and thus, the trilinear weights) are approximated with our technique (fig. 4). This is due to the limited set of preprocessed rays, to the limited number of projection lists and to the four-bits only depth indicator. Accuracy can be improved by increasing the size of tables and the number of bits in the depth indicator. However it also increases cache misses and reduces randomizing, which is not necessarily good because it trades noise for aliasing (fig. 3b).

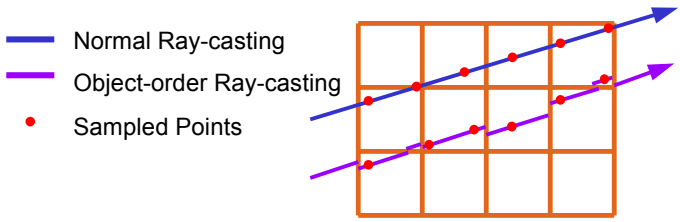


Figure 4: Randomized ray sampling in Object-order ray-casting

3.3 Interleaving Depth Indicators

While usual methods (Shear-Warp, 3D Textures and Splatting) sample the volume at the same depth locations, ray-casting can benefit from the incoherency between the rays to improve rendering by using a shifted sampling (fig. 5b). Theußl et al. have shown [27] that this way to sample the volume provides a better accuracy than the usual rectilinear sampling. In our algorithm, half of the depth indicators are initialized to zero and the others to half of the sampling rate such that every pixel has a different initialization than its four nearest neighborhoods. The use of randomized patterns is also conceivable. Figure 3 shows the resulting improvement. While staircase effects are visible when sampling at constant depths (fig. 3b and 3c), even with high sampling rates, interleaving depth indicators (fig. 3d) clearly reduces this aliasing.

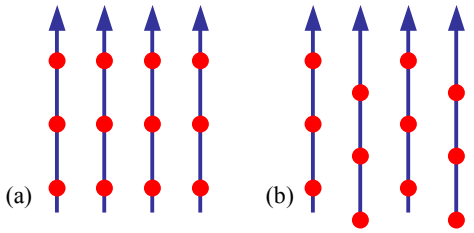


Figure 5: Usual rectilinear sampling (a) and shifted sampling (b)

4. OPTIMIZATIONS

Some improvements in the previous algorithm are implemented to obtain a really efficient approach. Here is a method on how to

skip transparent and hidden regions. Some other optimizations are also mentioned.

4.1 Skipping Transparent Regions

Because the classification process can eliminate a great part of the volume [12], skipping transparent regions is a major improvement in software volume rendering. However, a user-friendly application should always allow changing the classification interactively. In this way, a min-max octree structure is used in addition to the usual volume representation. The leaf nodes store the minimum/maximum values of the 8 vertexes of the corresponding cell while the other nodes bring up the values of their 8 sons.

Here, octrees have several advantages. First, using a trilinear reconstruction kernel makes traversing the octree in a front-to-back visibility order possible, which is not feasible in usual splatting methods without artifacts. Second, the octree can be run in an efficient memory order (with cache hits), whatever the viewpoint is. Third, two cells run consecutively are projected close to one another most of the time, which improves image caching. Fourth, it allows changing the classification interactively [12]. Finally, hidden nodes can be skipped with the new algorithm presented in the following section.

4.2 Hidden Volume Removal

In contrast with usual computer graphics applications where efficient occlusion culling methods have been widely studied [6, 7, 35], elimination of hidden volumes must be deeply improved in object-order DVR. A scan-line method was used in [12], but the efficiency of such a method seems limited. Mueller et al. [25] have proposed an occlusion test for splatting avoiding the projection process, but this test performed on every voxel remains. A more interesting approach is given by Lee and Ihm [9] where a min-max octree is used in association with either a range tree or a quadtree. Nonetheless, while the visibility test is not efficient with the former, the node visibility is approximated with the latter, providing artifacts. In fact the main problem of this approach comes from the use of trees that do not allow efficient neighbor finding. Instead, our new algorithm is based on hierarchical occlusion maps (HOMs) [35], which is by far superior for skipping the hidden nodes of the octree. This section only describes the two main lines of this algorithm: how HOMs are updated and how to perform the visibility test.

HOMs are images where the pixels store an integer value in the range [0..16] and are initialized to 0. The first occlusion map size is equal to quarter of that of the image and the last occlusion map

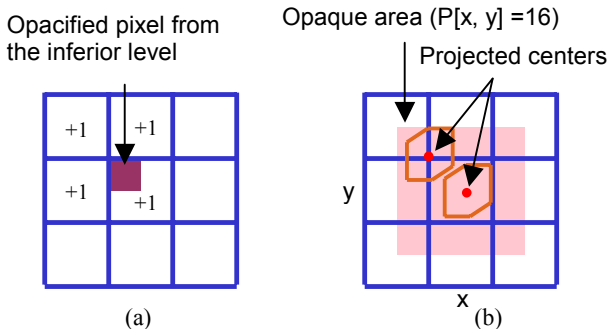


Figure 6: Examples of an occlusion map used during the updating process (a) and the visibility test (b).

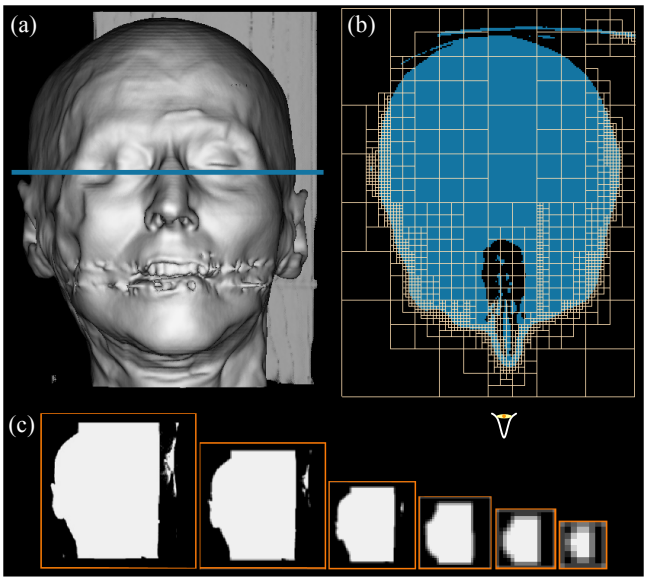


Figure 7: (a) Rendering example. The blue line gives the cutting plane. (b) Segmented cross section plus octree traversal. Here the non-leaf nodes crossing the blue area are removed by the occlusion test. (c) HOMs after the rendering (Level 0 [256^2] => Level 5 [8^2]).

of the hierarchy only represents one pixel. The updating process begins on the first HOM (i.e. the finest map) every time a pixel of the image plane becomes opaque. In this case the pixel of the HOM including the opaque pixel and its 3 nearest pixels in the map are incremented (fig. 6a). When an HOM pixel reaches its maximum value (i.e. incremented 16 times), updating starts again recursively at the superior level. Thus, the complexity of this process is equal to $m^2.Log_2(m)$ only, where m is the image width. However, an opacity test must also be added for every treated ray.

The visibility process consists in determining the hidden nodes during the rendering. Because the projection of a node is also represented by a hexagon, an HOM level is now associated with every octree level. This HOM level is equal to the finest level such that a HOM pixel can include the entire projection of the octree node (fig. 6b). Thus, the visibility test is performed for each node by looking if the pixel of the corresponding occlusion map where the center of the node projects is equal to 16. This value means that an extended square around the pixel made of 16 quarter of pixel (fig. 6b) is opaque.

Figure 7 shows an example of an octree run with efficient hidden volume removal. In this example, most of the encountered leaf nodes are located near the visible surface, though some of them are surprisingly far beyond because the recursive run of the volume allows efficient but non-optimal hidden volume removal. Another observation is the rareness of big block skipping. This is mainly due to the low accuracy given by the corresponding occlusion maps.

4.3 Other Optimizations

4.3.1 Fast Projection

The projection of the center of the octree nodes is a key element in our application because it is used in the visibility process (§ 4.2) and for determining the rays intersecting the cell (§ 3.1). Here, the use of an orthogonal projection allows the projected

Volume	Size	Rendering Mode	Cells	Occluded Cells	Octree Nodes	Sampled Points	Tbird 1.4 GHz	Duron 600 MHz
UNC Head (a)	256x256x225	Single	343K (4093K)*	91.70%	907K (4711K)*	645K	6.2 fps	2.6 fps
UNC Head (b)	256x256x225	Single	268K (1255K)*	79.70%	694K (1475K)*	453K	7.7 fps	3.4 fps
UNC Head (c)	256x256x225	Double	617K (5120K)*	88%	1570K (6300K)*	1077K	3.1 fps	1.4 fps
UNC Engine (d)	256x256x110	Single	236K (1453K)*	83.80%	621K (1686K)*	356K	9.1fps	4.5 fps
UNC Engine (e)	256x256x110	Double	371K (1175K)*	68.50%	759K (1570K)*	564K	5.6 fps	2.7 fps
UNC Engine (f)	256x256x110	Single	1544K (2150)*	28,2%	1822K (2750K)*	6747K	1.4 fps	0.55 fps
UNC Brain (g)	256x256x167	Single	226K (2434K)*	90.80%	593K (2808K)*	654K	7.7 fps	3.2 fps
Aneurism (h)	256x256x256	Single	71K (104K)*	31.80%	113K (190K)*	223K	20 fps	10 fps

* Without Hidden Volume Removal

Table 1: Measurements for different renderings.

center of a node to be computed from the projected center of the parent node by a simple 2D translation. In this way, eight constant 2D translation vectors are preprocessed for every level of the octree. 32 bits integer arithmetic is also used to quickly determine the subdivisions and the pixels affected by the projection and to quickly update recursively occlusion maps. Finally, the depth component used for fast depth cueing is computed in the same way.

4.3.2 Classification and Shading

Classification and shading are computed on every sampled point along the ray from the volume samples and the gradient by using a trilinear interpolation. Usually, classification and shading are performed either before the interpolation step (preclassification and preshading) or after (postclassification and postshading). The first technique is fast while the latter gives better results. However, in the case of preclassification and preshading it is strongly recommended to use opacity weighted color interpolation [13, 33]. Our approach is a hybrid method using postclassification but only preshading, in order to not degrade speed. Here the volume gradient is precomputed like many algorithms do [11, 12, 19], and a number indexing a set of quantized space directions is associated with every voxel. Shading is applied on every quantized direction before every rendering and the resulting pre-shaded reflectance map is used during the rendering to shade the 8 vertexes before the color interpolation at the sampling location.

Mueller et al. [24] have shown an example of preshading and postclassification with bad artifacts. We want to point out that no artifact is visible with our application and we think that the given interpretation of this phenomenon is probably erroneous.

4.3.3 Optimized Trilinear Reconstruction

MMX, SSE or 3DNow! SIMD processor extensions have been used to improve memory copying and trilinear interpolation. Color and opacity have been computed on 16 bits unsigned integers. However, code running on MMX only processors does not allow the line equation tests to take place (cf. 3.1) and then minor artifacts can occur on the image.

4.3.4 Volume Interleaving

To reduce cache misses, the volume is decomposed in small blocks containing 32^3 samples [11] where the bits of the coordinates are interleaved as follows:

$$(X_{n\ldots0}, Y_{n\ldots0}, Z_{n\ldots0}) \Rightarrow Z_{n\ldots5}Y_{n\ldots5}X_{n\ldots5}Z_4Y_4X_4\ldots Z_0Y_0X_0$$

A 32-entry table and binary operators are used to interleave the five lower bits and to generate the new memory address.

5. RESULTS

Here, we are looking at the first results of this new algorithm. The entire program except trilinear interpolation and memory copying (SIMD instructions) is written in C++. In contrast with many other DVR methods using small window sizes, here, all rendering are made on 512^2 pixel images not to degrade image quality. The number of quantized space directions used for shading is equal to 8192. Pixel subdivisions (fig.2a) and precomputed rays (fig. 2b) are both fixed to 16^2, allowing up to 4x zoom without visible noise. The *sampling_rate* variable determining the distance between two consecutive interpolations is initialized such as Δs=0.5, which allows between two and four samplings along a ray within a cell. Thus, these settings make high quality possible. Finally, due to the high efficiency of our algorithm when visualizing isosurfaces, the current implementation can also superimpose two rendering passes stemming from different transfer functions.

Benchmarks have been performed on two platforms allowing 3DNow! instructions: a low-end platform based on an AMD Duron 600 MHz (64/64 KB L1/L2 data cache) with 320 MB (SDRAM 100 MHz) and a more recent AMD Athlon 1.4 GHz (64/256 KB L1/L2 data cache) with 512 MB (DDR SDRAM 266 MHz). Four datasets have been used: the usual head, brain and engine from UNC Chapel Hill plus a highly compact angiography dataset (courtesy of Philips Research Labs). An additional PIII platform has also be used for a comparison with the Ultravis system.

The main results are summarized in table 1. All the measurements are averaged with 24 renderings. We observe a minimal/maximal divergence of less than 30% about rendering times. Octree processing and volume preshading take approximately 20 seconds, but are computed only once per volume. The fourth column indicates the number of cells (in thousands) within the volume that are neither transparent nor removed by the hidden volume test, even if all the rays going through them can be already opaque. The next column gives the ratio of occluded cells. The sixth column is about the number of octree nodes that are run. The values within parentheses are measured without hidden volume removal, which is disabled to rate its efficiency. The seventh column indicates the number of sampled points along the rays and the last two columns give the rendering times.

Volume	Ultravis	OO RC
UNC Head (a)	0.8 fps	2.4 fps
UNC Engine (d)	3.5 fps	3.5 fps
UNC Brain (g)	1.0 fps	3.2 fps
Aneurism (h)	0.45 fps	6.75 fps
Bonsai (fig. 1)	0.5 fps	2.1 fps

Table 2: Comparisons with the Ultravis system (PIII 600MHz, 512 MB)

The results show that interactive high-quality volume rendering is possible on current high-end platforms when visualizing isosurfaces. Better still, rendering remains interactive even with highly complex transfer functions that include a great part of the volume in the rendering process (f). No other algorithm running on a standard workstation is able to produce such a frame rate with this level of detail today. While methods based on 3D texture hardware currently do not exceed 2 fps on 256^3 volumes with a limited accuracy, previously mentioned ray-casting techniques are really slower. Only a Shear-Warp implementation should be able nowadays to deliver a superior frame rate, but once again with an important loss of quality that clearly limits its use. We have compared the efficiency of our algorithm with the Ultravis system [11], which is one of the most advanced ray-casting platforms. The same parameters have been used with both methods, but the Ultravis system, which generates 256^2 pixel images only, uses post-shading and can handle perspective renderings. This latter is the main drawback of our algorithm, but it is only required in some specific applications and many professional systems do not implement it [26]. The results clearly show the superiority of our approach, and like many other ray-casting algorithms, Ultravis performs badly on datasets with much empty space (cf. bonsai and aneurism datasets). Rendering times are only equals for the engine dataset where early ray termination is very important, showing us that HVR is also an efficient alternative to the lack of early ray termination in object-order volume rendering. Last but not least, we have noticed that our algorithm produces a much better image quality, partially due to the low image resolution of Ultravis.

By studying table 1 in great details, we have come to many interesting conclusions. First, we can clearly see that hidden volume removal eliminates a large fraction of the cells and octree nodes within the opacity range when visualizing isosurfaces. Here, the predominant parts of the rendering are the octree run and the voxel loading, while trilinear interpolation becomes predominant in the case of semi-transparent transfer functions. An important fact is that the efficiency of HVR is very superior to the method proposed by Lee and Ihm [9] where the ratio of occluded splats for similar renderings (b) and (g) is respectively only 25% and 67%. Actually, this ratio can also be considered as a good HVR speed-up estimation because our algorithm delivers an approximately constant cell throughput (between 1.4 and 2.1 millions cells per second). Thus, hidden volume removal is a very aggressive optimization here, but the recursive run of the octree does not optimize it (as seen previously). A better approach in future works might be a plane-by-plane volume run, allowing a better efficiency in occlusion tests. However, the non-leaf nodes will be run several times. Another observation is that the rendering times seem to scale well with the processor clock frequency, even if the two configurations are quite different (memory clock, L2 cache size).

6. CONCLUSION

A volume rendering application must be carefully designed and should always provide user-friendliness and accuracy. Hardware implemented techniques, although they are fast and sometimes not expensive, have limited on-board memory and are not flexible. On the other hand software algorithms can handle a wide variety of problems but suffer from a lack of performances. Our method offers a new way to perform ray-casting on rectilinear grids, achieving almost real-time volume rendering when visualizing isosurfaces and at least, interactive renderings in general. These achievements are mainly due to the efficient object-order ray-casting approach that we have introduced and to its optimizations like hidden volume removal. But in contrast with the rare software techniques able to produce such a frame rate, our algorithm reaches the high-level of details that scientific visualization requires. Indeed, its features such as randomized high sampling rate with trilinear interpolation, large image size (512^2 pixels), and interactive post-classification are facilitating devices.

We have found two drawbacks to our method. First, it is the use of preshading that slightly degrades the quality and finally, the lack of perspective projection. This latter is needed in stereo viewing applications or in virtual reality for example, but it is not required most of the time for scientific visualization where parallel projection is often preferred. In the future, we should look for a way to implement an efficient post-shading version of this algorithm in order to prevent the frame rate from decreasing too much. We are also planning to improve the rendering engine and to use multi-processor based PCs.

7. REFERENCES

[1] B. Cabral, N. Cam and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. IEEE/ACM Siggraph symposium on volume visualization, 1994, pp. 91-97.

[2] F. Dachille, K. Kreeger, B. Chen, I. Bitter and Arie Kaufman. High-quality volume rendering using texture mapping hardware. Proc. 1998 Siggraph/Eurographics workshop on graphic hardware, pp. 69-76.

[3] K. Engel, M. Kraus and T. Ertl. High Quality pre-integrated volume rendering using hardware-accelerated pixel shading. In Proc Eurographics/Siggraph workshop on graphic hardware, 2001.

[4] A. Van Gelder and K. Kim. Direct volume rendering via 3D texture mapping hardware. Proc. Volume Rendering Symposium 1996., pp. 23-30, 1996.

[5] J. S. Gondek, G. W. Meyer and J. G. Newman. Wavelength dependant reflectance functions. Siggraph'94 proc., 1994.

[6] N. Greene, M. Kass, G. Miller. Hierarchical Z-Buffer Visibility. SIGGRAPH'93 Proc., 1993, pp. 231-238.

[7] N. Greene. Hierarchical polygon tiling with coverage masks. SIGGRAPH'96 Proc., 1996, pp. 65-74.

[8] J. Huang, K. Mueller, N. Shareef, R. Crawfis. Fast splats: optimized splatting on rectilinear grids. IEEE Visualization'00 proceedings, October 2000.

[9] R. K. Lee and I. Ihm. On enhancing the speed of splatting using both object-and-image space coherence. Graphical models and image processing, vol. 62, no. 4, 2000. pp 263-282.

[10] J. Kajiya and B. Von Herzen. Ray tracing volume densities. SIGGRAPH'84, July 1984, pp. 165-174.

[11] G. Knittel. The Ultravis System. IEEE/ACM SIGGRAPH Volume visualization and graphics symposium 2000, October 2000, pp. 71-78.

[12] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. SIGGRAPH'94, 1994, pp. 451-458.

[13] M. Levoy. Display of surfaces from volume data. IEEE Comp. Graph. & App., Vol. 8, no. 5, 1988, pp. 29-37, 1988.

[14] M. Levoy. Efficient raytracing of volume data. ACM Transactions on graphics, vol. 9, no. 3, 1990, pp. 245-261.

[15] S.R. Marschner and R.J. Lobb. An evaluation of reconstruction filters for volume rendering. Proceedings of visualization'94, October 1994, pp. 100-107.

[16] N. Max. Optical model for direct volume rendering. IEEE Transaction on visualization and computer graphics, 1995, Vol. 1, no. 2, pp. 99-108.

[17] M. Meißner, U. Hoffman and W. Straßer. Enabling classification and shading for 3D texture mapping based volume rendering using OpenGl and extensions. Proc. Visualization'99, 1999, pp. 207-214.

[18] M. Meißner, J. Huang, D. Bartz, K. Mueller, R. Crawfis. A practical comparison of popular volume rendering algorithms. IEEE/ACM SIGGRAPH Volume visualization and graphics symposium 2000, October 2000, pp. 81-90.

[19] B. Mora, J.P. Jessel and R. Caubet. Accelerating volume rendering with quantized voxels. IEEE/ACM SIGGRAPH Volume visualization and graphics symposium 2000, Oct. 2000, pp. 63-70.

[20] B. Mora, J.P. Jessel and R. Caubet. Visualization of isosurfaces with parametric cubes. Eurographics'01 Proc., vol. 20 no. 3, pp. 377-384, September 2001.

[21] T. Möller, R.Machiraju, K.Mueller, R.Yagel. Evaluation and Design of Filters Using a Taylor Series Expansion. IEEE Transactions on Visualization and Computer Graphics, ITVCG 3(2): 184-199, June 1997.

[22] L. Mroz, H. Hauser and E. Groller. Interactive high quality maximum intensity projection. Eurographics'00, vol. 19, no 3, 2000.

[23] K. Mueller and R. Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. Proc. Visualization'98, 1998, pp. 239-245.

[24] K. Mueller, T. Möller and R. Crawfis. Splatting without the blur. Proc. Visualization'99, 1999, pp. 363-371.

[25] K. Mueller, N. Shareef, J. Huang and R. Crawfis. Splatting. High-quality splatting on rectilinear grids with efficient culling of occluded voxels. IEEE TVCG, Vol. 5, No. 2, 1999, pp. 116-134.

[26] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer and L. Seiler. The volumepro real-time ray-casting system. SIGGRAPH'99, 1999, pp. 251-260.

[27] T. Theußl, T. Möller, M. E. Gröller. Optimal Regular Volume Sampling. IEEE Visualization 2001 proceedings, October 2001, San Diego.

[28] M. Wan, A. Kaufman and S. Bryson. High performance presence-accelerated ray casting. proc. Visualization'99, 1999, pp. 363-371.

[29] R. Westermann and T. Ertl, Efficiently using graphics hardware in volume rendering applications. SIGGRAPH'98, 1998, pp. 169-177.

[30] R. Westermann and B. Sevenich. Accelerated volume ray-casting using texture mapping. In proc. 2001

[31] L. Westover. Interactive volume rendering. Proceedings of the Chapel Hill Workshop on volume visualization, may 1989.

[32] L. Westover. Footprint evaluation for volume rendering. SIGGRAPH'90 Proc., 1990, pp. 367-376.

[33] C. Wittenbrink, T. Malzbender, and M. Goss. Opacity-weighted color interpolation for volume sampling. Symposium on Volume Visualization, pp. 135-142, 1998.

[34] R. Yagel and A. Kaufman. Template-based volume viewing. Proc. Eurographics'92, vol.11, no.3, pp. 153-167.

[35] H. Zhang, D. Manocha, T. Hudson and K. E. Hoff. Visibility culling using hierarchical occlusion Maps. SIGGRAPH'98 Proc., 1998, pp. 77-88.

[36] J. Wilhelms and A. Van Gelder. A coherent projection approach for direct volume rendering. SIGGRAPH'91 Proc., pp. 275-284.

[37] L. Sobierarjski and R. Avila. A Hardware Acceleration Method for Volume Ray Tracing. IEEE Visualization 1995.

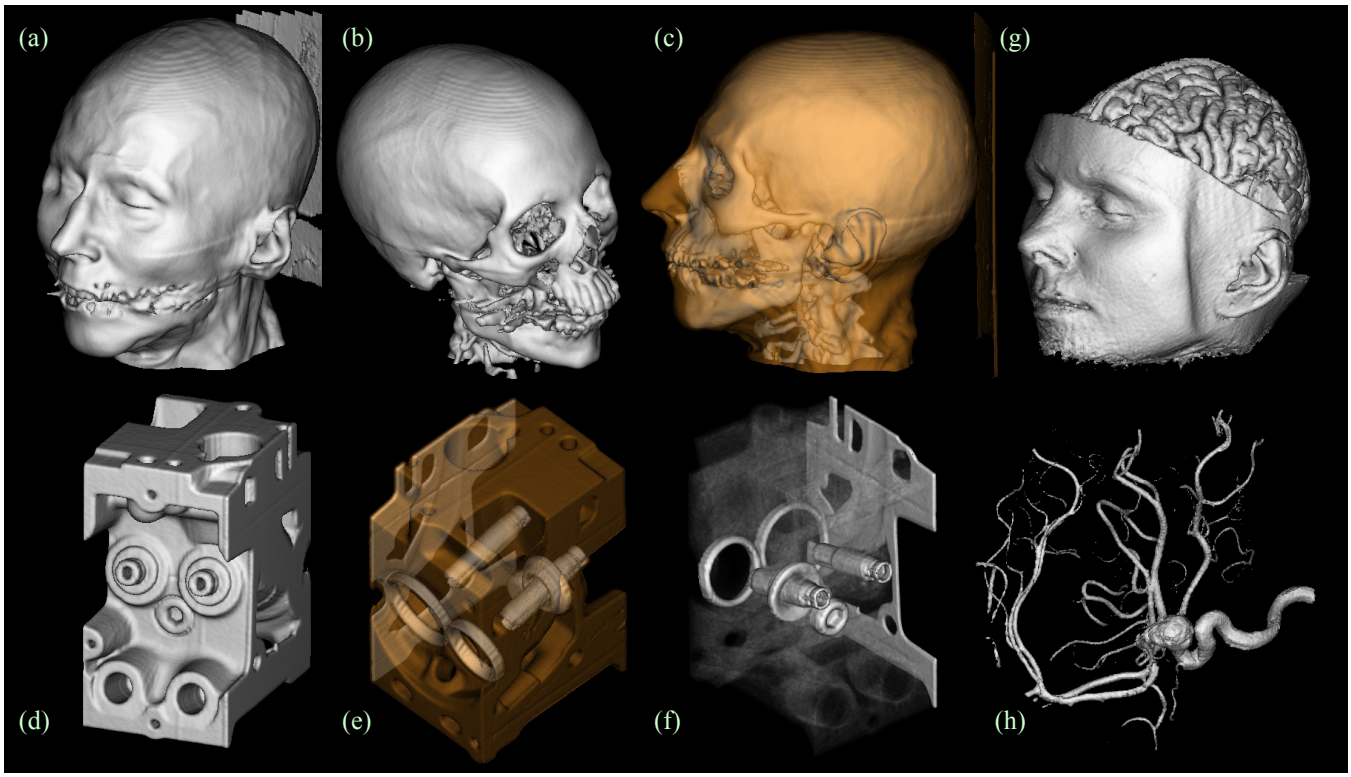


Figure 8: Different renderings used for the benchmarks