# Low-Complexity Maximum Intensity Projection

BENJAMIN MORA
University of Wales Swansea
and
DAVID S. EBERT
Purdue University

Many techniques have already been proposed to improve the efficiency of maximum intensity projection (MIP) volume rendering, but none of them considered the possible hypothesis of a better complexity than either O($n$) for finding the maximum value of $n$ samples along a ray or O($n^3$) for an object-order algorithm. Here, we fully model and analyze the use of octrees for MIP, and we mathematically show that the average MIP complexity can be reduced to O($n^2$) for an object-order algorithm, or to O(log($n$)) per ray when using the image-order variant of our algorithm. Therefore, this improvement establishes a major advance for interactive MIP visualization of large-volume data.

In parallel, we also present an object-order implementation of our algorithm, satisfying the theoretical O($n^2$) result. It is based on hierarchical occlusion maps that perform on-the-fly visibility of the data, and our results show that it is the most efficient solution for MIP available to date.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Viewing algorithms*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Raytracing*; *hidden line/surface removal*

General Terms: Algorithms, Theory, Performance

Additional Key Words and Phrases: Maximum intensity projection, volume rendering, complexity

## 1. INTRODUCTION

Maximum intensity projection (MIP) was one of the first volume visualization techniques and probably is the most widely used in the medical sector because of the surprising simplicity of this user-friendly algorithm. MIP simply finds the maximum value of the signal along each ray. Though the original MIP model is not able to give depth perception and shows volume occlusions, it provides a very good understanding of the structures defined by high signal intensities. It also avoids the problem of occluding structures that prevent the visualization of thin inner parts, which requires a possibly unreliable classification step in traditional optical models. Thus, MIP is not only very useful in medical renderings for the visualization of blood vessels and contrast-enhanced tissues, but can also be applied to other domains.

This research was partially supported by the National Science Foundation under grants NSF ACI-0081581, NSF ACI-0121288, NSF IIS-0098443, NSF ACI-9978032, and NSF ACI-0222675.

Authors' addresses: B. Mora, Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, U.K.; email: B.Mora@swansea.ac.uk; D. S. Ebert, School of Electrical and Computer Engineering, Purdue University, 465 Northwestern Ave., West Lafayette, IN 47907; email: ebertd@ecn.purdue.edu.

However, interactive high-quality volume rendering is still a challenge due to the ever increasing amount of data. In general, the global complexity of an object-order volume rendering algorithm is equal to the amount of data within the volume (i.e., O($n^3$) steps for a $n^*n^*n$ data grid). The complexity can also be expressed per ray for an image-order algorithm, and is usually equal to O($n$), or O($p^2 \times n$) for an image resolution of $p \times p$. This complexity can be improved in some specific volume rendering techniques, as in the visualization of isosurfaces [Lacroute and Levoy 1994; Mora et al. 2002] or in x-ray type renderings [Malzbender 1993], where the complexity is equal to $n^2 * \log_2(n)$ when working in the Fourier domain. However, there are many volume rendering algorithms, including MIP, where better complexity solutions are not known.

Therefore, improving the performance of the MIP algorithm can be crucial for patient diagnosis and treatment planning. Many previous approaches have been presented for optimizing MIP rendering [Heidrich et al. 1995; Sakas et al. 1995; Cai and Sakas 1998; Csébfalvi et al. 1999; Mroz et al. 2000a; Roerdink 2001; Kim and Park 2001; Pekar et al. 2003; Zuiderveld et al. 1994]. However, none of these have fully analyzed the complexity of MIP rendering or proposed a better complexity solution. In this article, we will show that finding the maximum along a ray can be theoretically reduced to an average O(ln($n$)) complexity instead of O($n$) by using octrees inside a ray-casting algorithm (image-order), or to an average O($n^2$) complexity instead of O($n^3$) for an object-order MIP algorithm. This fact leads to a significant performance improvement where a large portion of the rendering time is spent on the low-cost operation of visiting the nodes of the tree. Therefore, we implemented an object-order algorithm that requires a new efficient MIP "visibility" test at each node. Our results show that, regardless of the transfer function,[1] our algorithm provides highly interactive, high-quality renderings with better performance than previously published algorithms.

In this article, we only focus on accelerating the original MIP method. Other variants like local maximum intensity projection (LMIP) [Sato et al. 1998] will not be addressed. Section 2 examines state-of-the-art MIP techniques. In Section 3, we introduce the basis of the algorithm, while its complexity is detailed in Section 4. Section 5 presents a complete object-order implementation of the algorithm with optimizations. Results are discussed in Section 6, and future directions for this research are discussed in Section 7. Appendix A provides a proof of the complexity for $S_k(m)$, while Appendix B contains renderings.

## 2. PREVIOUS WORK

One of the first methods to speed up MIP was proposed by Sakas et al. [1995], where a DDA raytracer was used to find the maximum value along the ray. The major contribution of this article was the observation that 90% of the trilinear interpolations can be skipped just by comparing the (preprocessed) cell maximum and the current ray value because of the logarithmic expectation of this test, which was not reported. Another optimization was the use of a subcube memory organization in order to improve cache coherency. This work also proposed a better way to estimate trilinear interpolation. However, the algorithm still has to traverse a linear number of cells, and image-order algorithms are usually much slower than object-order approaches, as noted by Mora et al. [2002].

Later, Cai and Sakas [1998] proposed a new object-order approach using splatting [Westover 1990] in a shear-warp [Lacroute and Levoy 1994] context. However, splatting, which produces high-quality images, is probably too slow to be used in a real application, and trilinear interpolation may be a better choice.

The first interactive MIP algorithm was proposed by Csébfalvi et al. [1999] and uses a two-pass shear-warp implementation and binary operators. First, the signal intensity is subdivided into a few

[1]Here the goal of the transfer function is only to map the volume intensities into gray-level intensities.

intensity intervals (e.g., four), which allows the use of a few bits per intervals (e.g., 2 bits) and a small encoding of the volume. During the first pass on the interval-encoded volume, the algorithm determines the highest intensity interval for every ray. Here, using 2 bits per voxel and 32-bit integer arithmetic allows a "soft" parallelism that can treat 16 voxels per operation. Therefore, the second pass is able to compute the final image by using the interval image to skip the hidden regions of the volume. The drawback of this method is that the algorithm generates low-quality images, partially because of the use of a shear-warp technique.

The most advanced software approach is probably the one developed by Mroz et al. [2000a; 2000b], as mentioned by Brodlie and Wood [2001]. Unlike other MIP techniques that run the volume in either a front-to-back order or a back-to-front order, Mroz et al. noted that the volume samples can be projected in any order, permitting a different data storage organization. Instead of storing voxels within a three-dimensional (3D) grid, a linear array is used and its elements represent all the spatial coordinates (coded with 32 bits) of all voxels. This array is sorted according to the value of the voxels and 4096 beacons are set to implicitly indicate the values of the elements. Therefore, the algorithm is able to project only the voxels within the range of interest, which is a very aggressive optimization since the irrelevant voxels often represent a large portion of the volume [Lacroute and Levoy 1994]. Furthermore, the volume is traversed in a memory order that avoids cache misses. Finally, the Mroz et al. implemented a very efficient orthogonal projection technique similar to the one presented by Mora et al. [2000a].

In the first version of their algorithm [Mroz et al. 2000b], nearest-neighbor reconstruction is used, allowing near real-time renderings on a low-end PC platform. However, the lack of interpolation limits the accuracy of the final image. In order to circumvent this drawback, the authors later replaced the voxel coordinates by coordinates of a trilinearly interpolated cell that are sorted according to the maximum value of the cell's eight vertices [Mroz et al. 2000a]. The projection was also improved by storing coefficients for fast trilinear interpolation, which is similar to our method. Thanks to these optimizations, high-quality renderings generally remain highly interactive if the transfer function does not include the voxels of low intensity. However, not using the full range of intensities can greatly reduce the interpretation of the image since some information is missing. Finally, the authors mentioned another optimization based upon the observation that some cells can be hidden from the viewpoint. Thus, the algorithm gathers all the viewing directions into 12 viewing domains, and tries to eliminate all the invisible cells within each domain. This optimization seems inappropriate as the size of the data volume is multiplied by 12, and results show that only a low percentage ($\sim$30%) of cells are eliminated. However, we show that an on-the-fly visibility test can be more efficient.

Some other purely software MIP implementations use a multiresolution approach to speed up time critical renderings. In Kim and Park [2001], an 8X-undersampled volume was used to generate a fast low-resolution image before generating the final image. Nonetheless, rendering times were not shown and generating high-quality images seemed to be slow. A better approach [Roerdink 2001] used morphological operators and pyramids to generate a progressive rendering. This approach is clearly well suited to time-critical MIP rendering, but once again the time needed to obtain the full resolution image prevented interactivity. Finally, the most recent publication [Pekar et al. 2003] on MIP suggested dividing the volume in blocks of constant size, where the maximum of every block is precomputed. Then, a ray-casting algorithm using this information is used to skip blocks if either the current maximum is greater than the block maximum or the block is empty, similarly to Sakas et al. [1995]. Blocks are also ordered along the ray according their maximum values, but choosing the optimal block size differs according to the volume and is rather arbitrary. Also, the use of private data makes comparison of their results difficult. Our approach can be seen as a generalization of this technique by using a hierarchical structure, instead of only a two-level grid.

While purely software approaches of MIP now seem very efficient [Mroz et al. 2000a], some researchers have also tried to take advantage of 3D graphics hardware. One of the first approaches was the clever one developed by Heidrich et al. [1995]. Here, the usual cell was decomposed into triangles that were projected into the framebuffer. Before the projection, the three vertex coordinates of every triangle were transformed so that the first two coordinates were already the $(x, y)$ coordinates of the vertex projection onto the screen, and the third coordinate represented the vertex value. Therefore, the 3D hardware could perform the interpolation of the signal and the maximum test using the z-buffer. The final image was obtained from the z-buffer image. However, the algorithm is quite CPU demanding, and all voxels have to be processed.

Today, a better, only GPU demanding approach is the use of 3D textures [Cabral et al. 1994; Resk-Salama et al. 2000], where slices are extracted from the texture and projected (i.e., combined) in the framebuffer using the glBlendEquationEXT(GL_MAX_EXT) OpenGL extension. However, 3D graphics boards have their own limitations, including limited on-board memory, which prohibits the rendering of large-volumes, and limited data precision. Finally, much more expensive approaches have also been developed, such as the use of multiprocessors [Parker et al. 1999] and dedicated hardware [Pfister et al. 1999].

Consequently, purely software MIP algorithms are a good alternative to hardware-accelerated approaches because the algorithms are more flexible, extensible, and affordable. However, high-quality CPU-based approaches currently suffer from a lack of interactivity, especially when the intensity range of the transfer function includes a large portion of the volume dataset. This article will show that one can circumvent this drawback using low-complexity MIP.

## 3. MAXIMUM-INTENSITY PROJECTION AND TREES

In this section, we present the main principles of our octree-based MIP algorithm, and give a definition of visibility in a MIP context. We assume that voxels are independent and identically distributed for most demonstrations (except in Equations (2) to (7), where this is not required).

### 3.1 Ray-Traversal Within a 3D Grid

Before going further into MIP using octrees, we describe in greater detail the traversal of a ray within a 3D grid. First, for a volume with $n^3$ samples (or cells), the maximum number of six-connected cells encountered along a ray is equal to $3\times n$. In the case of axis-aligned rays, this number is reduced to $n$. More generally, we can say that the complexity of the number of cells along a ray is $O(n)$. Thus, if a simple ray-casting algorithm is implemented, $O(n)$ steps are needed for each ray to find the maximum value, which is exactly the problem we address in this article.

Another interesting point is the number of times the current MIP value changes when the ray is progressing within the volume. This question can be answered by considering that the number of changes is equal to the sum of the updating probabilities at each cell (i.e., the expectation of updating). So, if we consider that the cells only contain a single scalar value (i.e., there is no interpolation) and that cells are independent and identically distributed, the probability that the cell value is greater than the current MIP value is equal to the following:

—1, for the first voxel encountered;

—$1/2$, for the second encountered voxel, since the maximum value can be in the first or the second voxel with an equal probability;

—$1/n$, for the $n$th encountered cell, since it is the probability that the maximum value of $n$ cells is in the $n$th cell.

Therefore, the mean number of updates for $n$ voxels is approximately equal to

$$E(n) = \sum_{i=1}^{n} \frac{1}{i} \approx \ln(n). \tag{1}$$

This resulting complexity is impressive because there are only $\ln(n)$ updates. By using the same reasoning, one could easily prove that trilinear interpolation can be avoided in most cells by just comparing the current MIP value with the maximum value of the eight vertices of the cell [Sakas et al. 1995]. Thus, only $O(\ln(n))$ cells have to undergo a trilinear interpolation. We verified this by experimentally generating sequences of random floating point numbers. However, this demonstration only dealt with comparisons where the probability of equality of two values is null, whereas in medical datasets there is commonly an 8- or 12-bit quantization. If equality is possible, which can be seen as a "no update" case, the "$1/n$" form actually represents an overestimation of the update probability, and therefore, $E(n)$ also represents an overestimation.

In conclusion, there are still $O(n)$ cells traversed in the global algorithm, but there are many "hidden" cells between two cells consecutively updating the MIP value. By *hidden*, we mean that the already computed value is greater than the maximum cell value, and therefore the cell does not contribute to the final image. Thus, our idea is to detect and skip these cells by using a hierarchical octree structure. In the next sections, we will prove that using this strategy, the resulting complexity is greatly reduced, which is a significant result for MIP visualization.

### 3.2 Using Octrees in Image-Order MIP

As mentioned in Section 2, multiresolution volumes have also been used in MIP visualization for progressive image rendering. Our approach behaves differently by only using multiresolution volumes for skipping the nonrelevant voxels, and thereby reducing the complexity. Here, our multiresolution approach is in fact an octree, where the leaf nodes store the maximum value of the corresponding cell (i.e., the maximum of the eight vertices in the case of trilinear interpolation) while the other nodes store the maximum of their eight children. Octrees have been widely used in volume rendering [Levoy 1990; Wilhelms and Van Gelder 1992], but their application to MIP has not been analyzed before. The basic algorithm for traversing the octree can be written (Pseudo-C) as follows [Levoy 1990]:

```
INT RAY_TRAVERSAL(NODE N, RAY R, INT CURRENT_MAX) {
        INT VAL;
        IF (N.MAX<=CURRENT_MAX)          // VISIBILITY TEST
                RETURN CURRENT_MAX;
        IF (N.TYPE==LEAF) {
                VAL=INTERPOLATION (N.CELL, R);
                IF (VAL>CURRENT_MAX) RETURN VAL;
                ELSE RETURN CURRENT_MAX;
        } ELSE {
                FOR (I=0;I<8;I++) {
                        NODE SN=N.SUBNODE[I];
                        IF (IS_RAY_NODE_INTERSECTION(R,SN))
                                VAL=RAY_TRAVERSAL(SN, R, CURRENT_MAX);
                        IF (VAL>CURRENT_MAX)
                                CURRENT_MAX=VAL;
                }
                RETURN CURRENT_MAX;
        }
}
```

Basically, the Ray_Traversal function recursively traverses the nodes of the octree being intersected by the ray (a more refined algorithm of such a traversal can be found in Revelles et al. [2000]). The optimization stops the investigation of a node when the already computed MIP value is greater than the maximum value of a node. As previously demonstrated, the mean number of interpolations is very low (complexity $O(\ln(n))$). However, the total number of nodes that are traversed by this algorithm would be $O(n)$ if the visibility test did not occur. The next section, however, shows that the mean number is theoretically reduced to $O(n^{2/3})$ in this case, and can be further reduced to $O(\ln(n))$ when quantizing the gray levels.

Notice that the number of leaf nodes is equal to the number of cells, while the total number of nodes is 8/7th the number of cells. Our current octree implementation takes approximately twice the amount of memory required to store the original volume, which is considered as acceptable for most datasets. All the nodes store a *min* and *max* value, but nonleaf nodes also store a pointer to the eight children nodes. The use of the *min* value is not a requirement, but is used for the multipass optimization and will be discussed further. The accurate octree size can therefore be determined with the formula

$$\begin{aligned} Size &= n \times (2 \times S_{data}) + \frac{n}{8} \times (2 \times S_{data} + S_{pointer}) + \frac{n}{64} \times (2 \times S_{data} + S_{pointer}) + \cdots \\ &= n \times \left( (2 \times S_{data}) + \frac{1}{7} \times (2 \times S_{data} + S_{pointer}) \right), \end{aligned}$$

where $S_{data}$ is the size of the *min* and *max* values stored at every node (16 bits), and $S_{pointer}$ is the size of a pointer (32 bits). The octree is actually stored inside a preallocated array and pointers are integer indices. A better implementation could either not store the last level or only use $2 \times 3$ bits to indicate which vertices of the trilinear cell are the *min* and *max* values if the available memory is critical. Therefore, we do not consider the memory increase induced by the octree as significant.

### 3.3 Using Octrees in Object-Order MIP

As previously mentioned, the complexity of interpolation along a ray is equal to $O(\ln(n))$, and the complexity of the number of traversed nodes along a ray is equal to $O(n^{2/3})$ or $O(\ln(n))$. Therefore, most of the rendering time is spent traversing the octree. We have observed this phenomenon when implementing the previously cited basic ray-casting algorithm, where 30 to 40 nodes per ray on average are traversed when rendering a $256^3$ dataset. This result was not sufficient to get interactive rendering rates. Therefore, to minimize the number of traversed nodes, an object-order implementation of the low-complexity algorithm is better, requiring the examination of each node at most once.

The principle of our object-order low-complexity MIP is simple. The octree is once again recursively traversed in a depth-first top-down order, but now a node is removed from the pipeline only if the current minimum value of all the rays going through this node is greater than the maximum value of that node, which is assimilated as a "visibility" test in this article. When a nonhidden leaf node is reached, the algorithm projects the equivalent cell into the framebuffer, in the same manner as in Mroz et al. [2000a] and Mora et al. [2002]. The next section shows that the number of traversed nodes can still be theoretically reduced to either approximately $\sim O(n^{2.4})$ steps or $O(n^2)$ when reducing the number of gray levels, instead of $O(n^3)$.

## 4. MODELING COMPLEXITY OF THE OCTREE-BASED MIP

### 4.1 Expression of the Image-Order Algorithm Complexity

In order to determine the number of traversed nodes, we indirectly have to estimate the mean number of failures of the visibility test (i.e., how many times $n.max > current\_max$ occurs in the pseudocode).

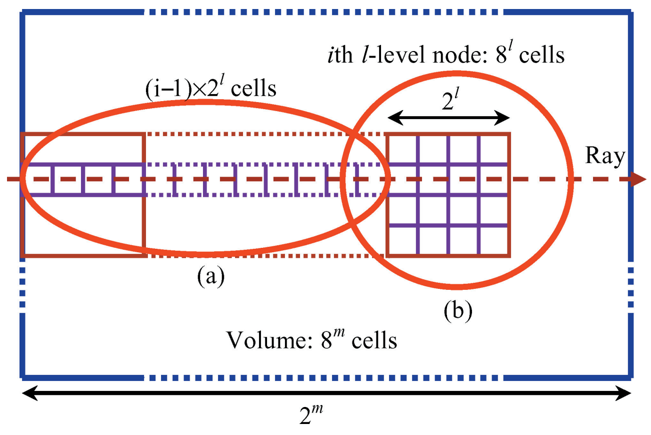


Fig. 1. The probability of failing the visibility test is equal to the probability that the maximum value belongs to the cells within the node, instead of belonging to the already encountered cells. Its value is *cells(b)*/[*cells*$(a)$ + *cells*$(b)$]. Note that the visibility order traversal used in this example is actually not compulsory for MIP algorithms.

This value can be calculated by summing the probability of each node failing the visibility test, which is a measure of expectation.

In order to simplify computations, we will assume here that rays are parallel to a volume axis, there is no interpolation, and the volume size is a power of 2. Let $m$ be this power (i.e., the grid contains $n^3$ elements, with $n = 2^m$). Let $l$ be the depth level in the octree ($m \geq l \geq 0$), with $l$ equal to 0 at the leaf nodes. At a given $l$-level, the total number of nodes that can be traversed is equal to $2^{m-l}$, and, moreover, the node value represents the maximum of a set of $8^l$ voxels (Figure 1(b)). When the test of the $i$th $l$-level node occurs ($2^{m-l} \geq i \geq 1$), $i - 1$ $l$-level nodes have already been either traversed or skipped, which means that the MIP value at this stage represents the maximum of $(i-1) * 2^l$ cells (Figure 1(a)). Thus, the probability that the visibility test fails at the $i$th $l$-level node is equal to the probability that a larger value belongs to the region represented by the octree node, instead of being within the voxels already traversed. This probability is simply expressed as

$$P(i, l) = \frac{8^l}{8^l + (i-1) \cdot 2^l} = \frac{4^l}{4^l + (i-1)}. \tag{2}$$

With this formula, we considered that with two subsets of distinct elements, respectively, made of $n_1$ and $n_2$ values, the probability to find the maximum in the first subset equals $\frac{n_1}{n_1+n_2}$. It is important to note that this formula holds only if all the samples are distinct inside the volume and the two subsets are randomly chosen inside the volume. Therefore, if we now consider a limited number of values inside the volume, Equation (2) is just an overestimation of the actual probability to continue the recursion at this node. However, the second condition cannot be ensured, since only a part of all the possible subsets can be considered by the algorithm. For instance, one subset is made of aligned samples in space, while the other contains samples that are included in a cubic volume area.

Thus, for regular datasets, the geometrically possible subsets should be representative of all the dataset subsets. To be generally more compliant with our assumption, the recursive ray traversal can visit the children of every node in a randomized fashion, which means that every node or cell intersected by the ray has the same probability of being processed as virtually the first, second, or $n$th element. In practice, sorting the children traversal according to the average intensity is better, since the nodes of high intensity are projected earlier, enhancing the occlusion map efficiency (Section 5.3.1). Finally, nodes are shown in front-to-back processing order in Figure 1 for explanatory purposes only.

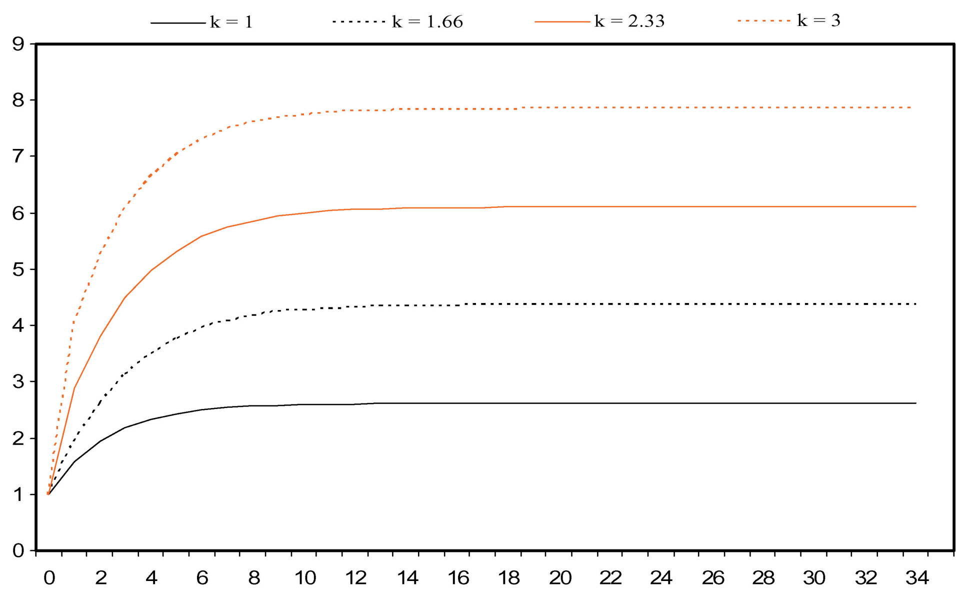


Fig. 2. Convergence of the expression $S_k(m)/2^{2m/3}$, for different values of $k$. Low asymptotic deviation of $S_k(m)$ ($< 2\%$) is reached for practical values of $m > 8$. Notice that these functions tend to a constant proportional to $k$.

To estimate complexity, the next step is to sum all the probabilities of the $l$-level nodes:

$$Q(m,l) = \sum_{i=1}^{2^{m-l}} \frac{4^l}{4^l + (i-1)} = 4^l \sum_{i=0}^{2^{m-l}-1} \frac{1}{4^l + i} \quad . \tag{3}$$

Finally, the mean number of failures of the visibility test is given by the summation of all levels:

$$R(m) = \sum_{l=0}^{m} 4^l \cdot \sum_{i=0}^{2^{m-l}-1} \frac{1}{4^l + i} \quad . \tag{4}$$

An analysis of $R(m)$ reveals that its complexity is equal to $O(4^{m/3})$ (see Appendix A, for the proof). $R(m)$ is also the complexity of the traversed nodes because, every time the test fails, either eight subnodes are computed, or a cell is projected. Thus, the number of traversed subnodes (which are also nodes) is approximately equal to $8^*R(m)$, which has the same complexity as $R(m)$. We therefore conclude that the mean complexity of our algorithm to find the maximum along a ray is $O(n^{2/3})$, for an $n^3$-element grid.

However, rays are not generally parallel to the volume axes. Thus, we can admit that an average value $k$ exists ($1 \leq k \leq 3$, $k$ depending on the viewing angle, and the nodes being six-connected), such that, given an $l$-level, the total number of nodes that can be traversed is approximately equal to $k^*2^{m-l}$, and the mean number of cells already encountered is approximately equal to $k^*(i-1)^*2^l$. The complexity can now be expressed as follows:

$$S_k(m) \approx \sum_{l=0}^{m} 4^l \cdot \sum_{i=0}^{k \cdot 2^{m-l}-1} \frac{1}{4^l + k \cdot i} \quad . \tag{5}$$

This expression has the same complexity as $R(m)$. Figure 2 shows the asymptotic behavior of $S_k(m)$. However, the full potential of the low MIP complexity (rendering times are multiplied by $2^{2/3}$ every time the volume size doubles) is reached for practical values of $m$ greater than 7 or 8 (volume size equals $128^3$ or $256^3$). Notice that the expression $S_k(m+1)/S_k(m)$ is around 2 for low values of $m$ (around zero), and slowly converges to $2^{2/3}(\sim m > 6)$.

### 4.2 Expression of the Object-Order Algorithm Complexity

Modeling complexity for such an algorithm is not easy, and we had to make the following restrictions for the object-order model:

—the volume is made of $n^3$ elements, with $n = 2^m$;
—rays are parallel to the volume axes;
—rendering is made on a $n^2$ image and there is a 1:1 pixel/voxel ratio;
—rays are independent;
—there is no interpolation;
—the visibility test is done in one step.

From these hypotheses, we can now express a new probability for nodes. In the same manner as in Section 4.1, a node will be characterized by its depth $l$ and its projection order $i$ on the frame buffer (Figure 1). The difference from the previous algorithm is now that a node is "invisible" and not processed only if the already computed values of all the rays ($4^l$ rays) traversing it are greater than the maximum value of the node. Due to the assumption that rays are independent, the new probability $P'(i,l)$ for the node to be visible can be expressed as 1 minus the product of the probability that every ray passes the test (2):

$$P'(i,l) = 1-(1-P(i,l))^{4^l}, \tag{6}$$

$$P'(i,l) = 1-\left(\frac{(i-1)\cdot 2^l}{8^l+(i-1)\cdot 2^l}\right)^{4^l}. \tag{7}$$

By summing probabilities for all the $l$-level nodes (notice that there are now $4^{m-l}$ $l$-level nodes having the same depth $i$), we have

$$Q'(m,l) = 4^{m-l}\cdot\sum_{i=1}^{2^{m-l}}\left(1-\left(\frac{(i-1)\cdot 2^l}{8^l+(i-1)\cdot 2^l}\right)^{4^l}\right). \tag{8}$$

So total complexity $R'(m)$ for the object-order algorithm can be expressed as

$$R'(m) = \sum_{l=0}^{m}\left(4^{m-l}\cdot\sum_{i=0}^{2^{m-l}-1}\left(1-\left(\frac{i\cdot 2^l}{8^l+i\cdot 2^l}\right)^{4^l}\right)\right). \tag{9}$$

Proving the complexity of such a formula is not an easy task. However, the expression can be computed for values of $m$ less than 34 (see Figure 3), which ensures that the complexity will be compliant with volumes produced in the near future. Therefore, one can empirically suspect an asymptotic regime (orange curve) close to $n^{2.4}$ (with $n = 2^m$) from such a figure. By dividing this number by $n^2$, which is the number of rays (hypotheses), we can assume that the "equivalent" complexity per ray is approximately O($n^{0.4}$), which is better than the previous implementation. The black curve shows the derivative of $R'(m)$ on a logarithmic scale. While a linear algorithm should tend to 3, our theoretical model quickly tends to approximately 2.4, showing that the algorithm should be efficient for practical volume sizes.

In this model, we considered that the $4^l$ rays that traverse an $l$-level node are independent, but in reality those rays are likely to traverse the same homogeneous regions. We could then consider that

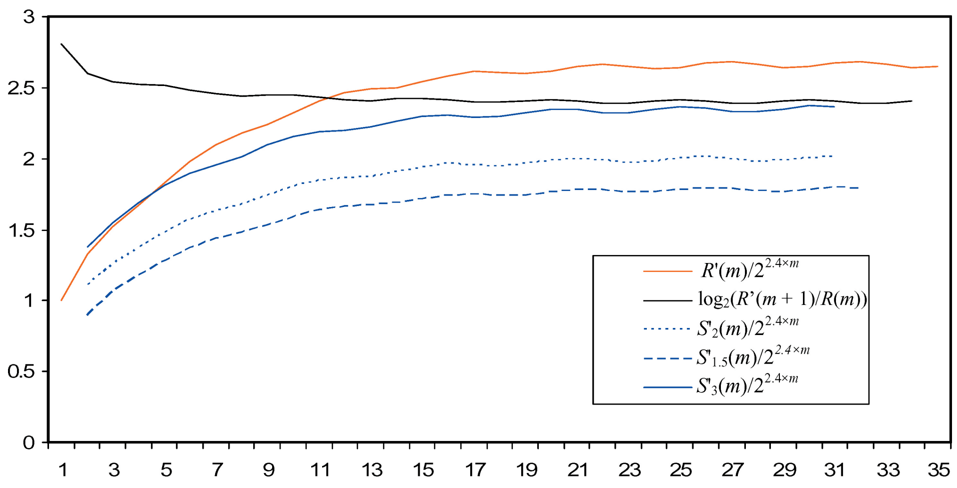


Fig. 3. Convergence of the expressions $R'(m)/2^{2.4\times m}$ and $S'_k(m)/2^{2.4\times m}$ for different values of k (orange and blue curves). The black curve represents the expression $\log_2(R'(m+1)/R(m))$, showing that the approximate 2.4 constant is quickly reached.

there is a specific number $p$ of rays that are representative of all the rays crossing the node ($p \leq 4^l$). In this case, we could replace the $4^l$ term in Equation (7) by p, and then Equation (7) would become an overestimation of the actual probability. Thus, having homogenous regions should speed up the rendering.

Generating an exact nonparallel complexity model is difficult. Therefore, we just give a function that overestimates the cost in order to be more generic. However, we think that the theoretical results we get from axis-aligned renderings are probably valid for the general case.

In $R'(m)$, every $l$-level node has an order-number along a ray between 1 and $2^{m-l}$. Since rays can now be diagonal, we can now suppose that the maximum depth of $l$-level nodes is $k\times 2^{m-l}$, where $k$ depends on the viewing angle. Also, while the number of $l$-level nodes having the same depth $i$ was previously constant ($4^{m-l}$), it can now vary according to $i$. However, this number can be overestimated by the maximum number of $l$-level nodes that can be crossed by just one plane (e.g., $3\times 4^{m-l}$). Therefore, the overestimation can be given by

$$S'_k(m) = \sum_{l=0}^{m} \left( 3 \times 4^{m-l} \cdot \sum_{i=0}^{2^{m-l}\times k-1} \left( 1 - \left( \frac{i \cdot 2^l}{8^l + i \cdot 2^l} \right)^{4^l} \right) \right). \tag{10}$$

Empirical tests on this model show that this expression has the same complexity as $R'(m)$, that is, $O(n^{2.4})$, as shown in Figure 3.

## 4.3 Working with Quantized Gray Levels

A useful improvement to the previous technique utilizes the fact that the human visual system can only distinguish a limited number of gray levels (known as *Weber's law* [Glassner 1995]). In volume rendering, the volume data are classified using a lookup table (LUT) that maps the signal intensity into gray levels. Usually in MIP, this classification can be made after the rendering step (called here *MIP postclassification*), which is preferable since mapping is performed only once per pixel instead of once per interpolated sample. When working with 12-bit medical data, for example, the classification has to map 4096 gray levels into either 1024 or 256 or 64 gray levels on current regular display systems. This means that two pixels, having a close value before the classification, will usually appear with the same color in the final image. Therefore, the visibility test comparing the maximum of the node with the minimum value of the pixel within the node projection (object-order algorithm) does not need to be "strictly less than or equal" to skip the node. A constant error on this test can be added to reject more nodes, without affecting the final interpretation of the image. Assuming that the LUT is linear with a minimal and maximal intensity value (*min* and *max*), and where all the intensities under or beyond these thresholds are respectively mapped with either black or white, this error can be computed as

$$\varepsilon = \frac{max - min}{number\ of\ gray\ levels}. \tag{11}$$

However, this technique also prevents updating the hierarchical occlusion maps (a description will be given in the next section), and seems to actually affect the efficiency of our object-order algorithm if the error is too high. Therefore, the speedup gained with a low value is withdrawn when using high error values.

To circumvent this problem, we use an on-the-fly MIP preclassification. For every interpolated volume sample or node maximum that still belongs to the original value domain, the algorithm first maps these values into quantized gray values using a LUT before doing the regular tests and pixel updates. Therefore, the visibility algorithm works in the quantized domain instead of working in the signal domain. The quantization takes place after mapping the values into gray scale through transfer function manipulation and windowing. Therefore, our algorithm can still handle high-precision data without any loss of information when windowing the intensity range (e.g., to improve contrast). By taking 128, 64, or 32 gray levels in the final image, rendering times can be greatly improved. Once again, an error value of 1 in the visibility test is also acceptable to increase the efficiency. Finally, a linear LUT is no longer required, as long as the LUT is monotonic (always increasing).

By analyzing this optimization, we discovered that it allows strikingly reduced complexity. Previously, our demonstration dealt with floating point values, where the probability of $(a = b)$ was supposed to be zero, and was followed by the estimated probability of nodes (2). Let's now suppose that data has been mapped into $g$ gray levels and two sets $A$ and $B$ consisting of $N_a$ and $N_b$ elements. We want to estimate the probability that *Max(A) < Max(B)*. By counting cases, we get

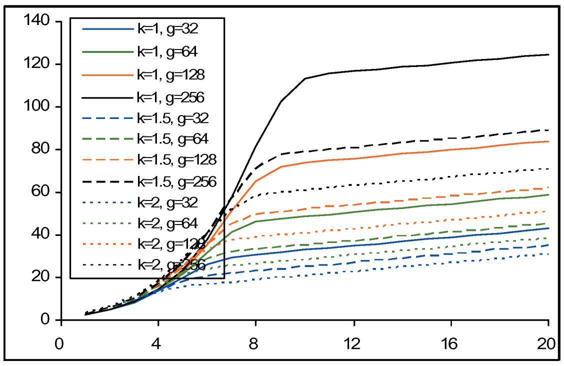
$$Prob_g(Max(A) < Max(B)) = Pmax_g\,(N_a, N_b) = \frac{\sum_{i=1}^{g} (i-1)^{N_a} \times \left(i^{N_b} - (i-1)^{N_b}\right)}{g^{N_a+N_b}}, \tag{12}$$

$$Pmax_g\,(N_a, N_b) = \sum_{i=1}^{g} \left(\frac{i-1}{g}\right)^{N_a} \times \left(\left(\frac{i}{g}\right)^{N_b} - \left(\frac{i-1}{g}\right)^{N_b}\right). \tag{13}$$

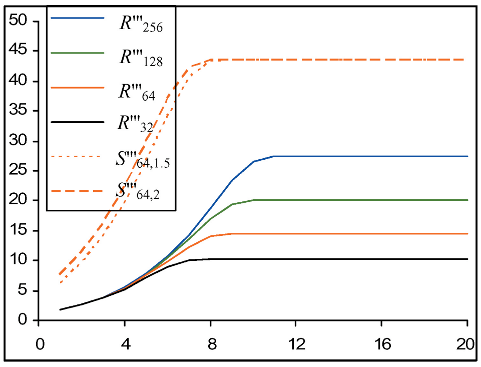
With the first term of the multiplication in (12) representing the number of cases where all elements of $A$ are less than $i$, and the second term representing the number of cases where the maximum of $B$ equals $i$. Therefore, replacing (2) by this formula in (4), we get the following average number of nodes that are traversed (image-order algorithm):

$$R''_g(m) = \sum_{l=0}^{m} \sum_{i=0}^{2^{m-l}-1} Pmax_g\left(i \times 2^l, 8^l\right). \tag{14}$$

Here, we assume that a ray traverses only two of the eight children of every node. For the nonparallel case, we once again assume that, for every ray, there is an average of $k \times 2^l$ $l$-level nodes that are crossed, with $k$ depending on the viewing angle $(3 > k \geq 1)$. By similarly replacing (2) by (13) in (5),



(a)



(b)

Fig. 4. (a) Evaluation of $S''_{g,k}(m)$ for some values of $k$ and $g$. This function varies linearly to the octree depth $(m)$, which proves the logarithmic complexity. (b) Evaluation of $R'''_g(m)/4^m$ and $S'''_{g,k}/4^m$ for some values of $g$ and $k$. The asymptotically constant behavior tells us that the average complexity of our object-order version is $\mathrm{O}(n^2)$.

the complexity model for nonparallel rays can be expressed as

$$S''_{g,k}(m) = \sum_{l=0}^{m} \sum_{i=0}^{k\times 2^{m-l}-1} Pmax_g\left(k \times i \times 2^l, 8^l\right). \tag{15}$$

While it is likely that a mathematical demonstration exists for proving the convergence of these series, we prefer to compute them up to $m = 20$ (Figure 4(a)). These formulas are asymptotic to the function $y = x + b$, which means a logarithmic average complexity $\mathrm{O}(ln(n))$ (with $n = 2^m$) for finding the maximum along a ray. However, the $b$ constant is quite large and will prevent real-time rendering if many rays are cast. Also, the worst case for finding the maximum still requires $\mathrm{O}(n)$ steps.

By applying the same reasoning as in Section 4.2 (Equations (6) to (9)), we can model the complexity of our object-order algorithm for parallel rays, which leads us to the new formula

$$R'''_g(m) = \sum_{l=0}^{m} 4^{m-l} \sum_{i=0}^{2^{m-l}-1} \left(1 - \left(1 - Pmax_g\left(i \times 2^l, 8^l\right)\right)^{4^l}\right). \tag{16}$$

This function is asymptotic to the function $y = 4^x + b$ (Figure 4(b)), which means that the average complexity is $\mathrm{O}(n^2)$ for rendering an $n^3$ volume. However, the best case is $\mathrm{O}(n^2)$ while the worse case is still obviously $\mathrm{O}(n^3)$.

Proposing a good non-axis-aligned model is still not a very easy task. By applying the same idea as for $S'_k(m)$ as in Equation (10), and replacing Equation (2) by (13), we can obtain an overestimation for the average number of processed nodes $S'''_{g,k}(m)$:

$$S'''_{g,k}(m) = \sum_{l=0}^{m} 3 \times 4^{m-l} \sum_{i=0}^{k\times 2^{m-l}-1} \left(1 - \left(1 - Pmax_g\left(i \times 2^l, 8^l\right)\right)^{4^l}\right). \tag{17}$$

Unsurprisingly here, the asymptotic behavior is the same as in Equation (16), meaning a complexity of $\mathrm{O}(n^2)$ for the non-axis-aligned version.

## 4.4 Summarizing Complexity Results

While some assumptions about our complexity models may be open to further discussion, especially for the non-axis-aligned projections, this section has shown some new ways to model complexity in volume

Table I.

| | Regular MIP | | MIP on quantized gray levels | |
|---|---|---|---|---|
| Complexity | Image-order | Object-order | Image-order | Object-order |
| Average | O($p^2n^{2/3}$) | ~O($n^{2.4}$) | O($p^2\ln(n)$) | O($n^2$) |
| *Min* | O($p^2$) | O($n^2$) | O($p^2$) | O($n^2$) |
| *Max* | O($p^2n$) | O($n^3$) | O($p^2n$) | O($n^3$) |

rendering and has finally answered the question about the possible existence of a reduced average complexity for MIP. Our theoretical complexity results when using octrees in MIP are summarized (given an $n^3$ volume and $p^2$ rays for the image-order algorithm) in Table I.

The *Min* case that was not discussed before is trivial: every pixel needs to be initialized, but the algorithm may stop recursion right after the root node test if the volume only contains zero values.

Therefore, the object-order version seems to be more suitable than the image-order version, since the logarithmic part disappears. The next section will describe a way to implement our new object-order algorithm, and Section 6 will strengthen these theoretical results.

## 5. GLOBAL IMPLEMENTATION

Previous results showed that the object-order version could potentially be faster than the image-order one, which was observed in practice. Therefore, we need an efficient way to determine the visibility of every node to implement the object-order algorithm. In order to satisfy these constraints, we implement a slightly modified version of the fast object-order ray-casting (OORC) algorithm described in Mora et al. [2002]. In the next sections, we will focus on the main differences with the original algorithm.

### 5.1 Object-Order Ray-Casting

The main idea of the OORC algorithm is to execute the same pipeline as a usual ray-casting algorithm, but in an object-order way. The reasons are quite simple: having more data traffic on the image than on the volume is preferable because cache coherency is easier with the former alternative. Furthermore, it is also easier to skip empty regions in object-order algorithms (e.g., by using an octree). Thus, for every cell that has to be rendered (i.e., not transparent and not hidden), the pipeline is as follows: first, the values of the eight vertices are loaded. Then, for every ray intersecting the cell, colors and opacities are sampled along it before updating the equivalent pixel.

However, there are three difficulties that arise when using such a method. First, the algorithm has to quickly determine the ray-cell intersections and the sampling positions. An efficient solution comes from the use of precomputed projection arrays when limited only to orthogonal projection, as described in Mroz et al. [2000a] and Mora et al. [2000]. The second difficulty comes from how to ensure an evenly spaced sampling between two consecutive voxels along a ray. A depth indicator stored on every pixel can circumvent this problem. However, evenly spaced sampling is useful only in some specific cases, like for integral estimations, so this feature can be disabled in MIP renderings. In our algorithm, the first sampling point is the (preprocessed) entry point of the ray within the cell, and the next sampling positions are computed by adding a constant vector until the position goes out of the voxel. This procedure ensures that the sampling distance along the ray is less than or equal to a predefined value and is very similar to the one used by Mroz et al. [2000a].

Finally, the last difficulty is how to compensate for the loss of early ray termination. Thus, a hidden volume removal (HVR) technique based on Hierarchical Occlusion Maps (HOMs) [Zhang et al. 1997] is used. However, the on-the-fly visibility originally presented for optical models cannot be applied to MIP rendering. Therefore, the next section describes the first on-the-fly MIP visibility technique, which is the main difference from the original OORC algorithm [Mora et al. 2002].

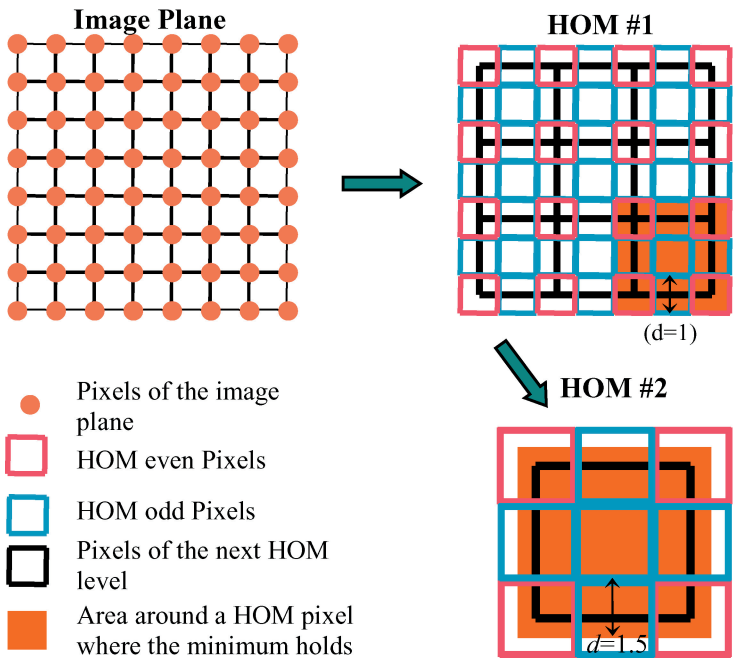


Image Plane
HOM #1
(d=1)
HOM #2
d=1.5
Pixels of the image plane
HOM even Pixels
HOM odd Pixels
Pixels of the next HOM level
Area around a HOM pixel where the minimum holds

Fig. 5. Spatial layout of the Hierarchical Occlusion Maps used in MIP. While the corners of the pixels of the first HOM match four neighboring pixels of the image plane, the corners of pixels from other HOMs fit into the centers of four neighboring sublevel even pixels. In both cases, a HOM pixel contains the minimum of the four elements setting the corners. This way, a HOM pixel gives the minimum of an extended area on the image centered on itself. The range of action ($d$) actually depends on the map level.

## 5.2 Hidden Volume Removal in MIP Renderings

As in the original algorithm, the MIP version uses a min-max octree in addition to the 3D grid both to recursively traverse the volume and to skip transparent or hidden regions. When a visible leaf node is reached, the algorithm projects the equivalent cell into the frame-buffer. The leaf nodes of the octree store the minimum/maximum values of the eight vertices of the corresponding cell, while the other nodes store the min-max values of their eight children. During an object-order low-complexity MIP rendering, the visibility of every node has to be estimated. A naïve way to perform this test would be to project the node into the image plane and to test every pixel on the projection. However, we believe that this scheme would be quite inefficient, since it would perform the same number of operations (i.e., tests) as the image-order implementation. Thus, the use of MIP dedicated HOMs [Mora et al. 2002] was preferred here because it only requires one test per node, even if the estimation of the visibility is now less accurate, and therefore fails more often. Our HOM approach has the advantage over quadtrees [Lee and Ihm 2000] to combine a correct estimated visibility (nonhidden cells will not be detected as hidden), with an efficient neighbor finding. The algorithm we developed is somewhat complex. Therefore, we will accurately describe the two main components of this algorithm: how to update HOMs and how to perform the visibility test.

HOMs are a set of multiresolution images (Figure 5) computed from the framebuffer, which store for each pixel the minimum value of a surrounding region in the framebuffer. The width and height of the first HOM are equal to those of the image minus 1, the width and height of the $n$th HOM represent the half of those of the $(n-1)$th HOM minus 0.5, and finally the size of the last HOM of the hierarchy only represents one pixel. Every time a pixel of the framebuffer is changed during the rendering (i.e., its value is increased), HOMs are updated recursively from the most to the least accurate map. Notice that, while the pixels of the framebuffer are considered as points, HOM pixels represent square regions.

A HOM pixel represents the minimum value of either four neighboring HOM even pixels of the inferior level, or four neighboring pixels of the framebuffer in the case of the first HOM. A HOM pixel is said to be even if its two coordinates are even numbers. When a $p$ pixel is increased, the minimum value of the four pixels surrounding $p$ of the first HOM (black-edged squares on Figure 5, left side) have to be recomputed from both $p$ and the eight pixels around $p$. Finally, if among the four recomputed pixels, the minimum value of the even pixel changes, then the algorithm recursively starts again to updates the four pixels of the superior level that surround the even pixel, and so forth.

Therefore, a square area exists around every HOM pixel where the value of the pixels of the framebuffer within this area is certified to be greater than or equal to that of the HOM pixel. A range of action for every area depending of the HOM level can be defined as the distance between the border of the HOM pixel and the border of its surrounding area. This value is given by the formula $Range(n) = 2^{n-2} + 1/2$, where $n$ is the HOM number, starting at 1.

Testing the visibility of a node is quite simple now. Due to the orthogonal projection, the same hexagonal shape, in which the size only depends of the octree level, delimits the projection of every octree node. Therefore, the algorithm first associates a HOM number for every octree level such that it is the finest map where its range of action is greater to the radius of the hexagonal shape. Here the radius of a hexagon is defined as the maximum distance-on-axis between the center of the hexagon and its 2D vertices. Thus, the visibility test just consists of projecting the center of the node within a HOM pixel and then comparing the pixel value with the node value. If the value of the pixel (representing the minimum of all pixels inside the surrounding area) is greater than the value (maximum) of the octree node, the octree node is invisible and can be skipped.

Even if the visibility test requires only a few operations, we must examine the complexity of updating the HOMs. Let's consider $n^3$ volumetric samples and $n^2$ pixels, with an average of $t$ updates per pixel. Every pixel update requires at most $\log_2(n)$ HOM updates, but $4/3$ HOM updates on average if we consider that the probability to update a HOM even pixel is $1/4$ (resulting in a geometric series). So the average number of HOM updates during one rendering pass can be estimated by $4/3 \times t \times n^2$. Thus, this result shows that the number of updates has the same complexity as the algorithm and will not slow down the rendering process. Indeed, if we use quantized gray levels, $t$ should be independent of the volume size because the number of nodes that are run will be in $O(n^2)$ and therefore, at most $O(n^2)$ cells will be projected on $n^2$ pixels.

## 5.3 Other Optimizations

The object-order low-complexity MIP rendering is significantly sped up by some additional optimizations detailed below.

5.3.1 *Presorted Octree Run.* A great difference between optical renderings and MIP is that MIP permits projecting the samples in any order, as mentioned in Mroz et al. [2000a]. Therefore, projecting the cells of high value before the cells of low value, instead of running the volume in an arbitrary order, will increase the HVR efficiency. In this way, our algorithm also sorts the eight subnodes of every node when computing the octree. The sorting criterion is the mean value of the signal within the subnode, since sorting by the maximum value is actually less efficient. This technique only requires the use of 3 additional bits per octree node, which are stored in the 16-bit maximum value (but only 12 significant bits) of every node. Notice that a random walk is possible too, which would produce an algorithm better respecting the assumption we used for modeling complexity.

5.3.2 *Multipass Rendering.* However, running a presorted tree is still not perfect, since, for two subnodes having the same parent, the low-range samples of the node which is traversed first will be projected before the high-range samples of the second node. In order to optimize this bottleneck, our algorithm divides the intensity domain into a small number of disjoint intervals and traverses the octree once for each interval. The signal intervals are empirically chosen using a precomputed histogram such that the first domain gathers the $10^5$ highest cells, the second domain gathers the next $4\times10^5$ highest cells, and the $n$th domain gathers the next $4^n\times10^5$ highest cells. This technique requires that the minimum of all cell maximums are now also stored in the octree, in order to only traverse the nodes in the range of interest at each pass.

## 6. RESULTS

The entire program, except trilinear interpolation and memory copying (which were optimized using MMX SIMD assembly instructions), is written in C++. In contrast with many other rendering methods using small window sizes, our results are for an image of $512^2$ pixels, which is sufficient in most cases. The sampling rate determining the maximum distance between two consecutive samples along a ray is set to 0.5, which allows between two and four interpolations along a ray within a cell. Benchmarking has been performed on a high-end AMD Athlon 64 3400+ (2.2 GHz) with 2 GB of memory. We used three widely used datasets for reproducibility: the UNC Head ($256^2 \times 225$ voxels) volume, a hand ($256^2 \times 100$ voxels), and the Xmas Tree dataset ($512^2 \times 499$ voxels) from Vienna University. All the volumes are 12 bits and all the times are averaged from several viewpoints (we noticed variations of at most $\pm 40\%$ according the view angle). We were unfortunately not able to load volumes containing more than $512^3$ voxels with our current implementation, since the 32-bit version of Windows XP does not allow allocating more than 2 GB per process.

A comparison (on the same platform) was also made to the approach developed by Mroz et al. [2000a], which is, in our opinion, the best software implementation of MIP to date. Notice that this application has been written in Java; we think that a good C++ or assembly version might improve the frame rate significantly, and Mroz et al. [2000a] mentioned a speedup of 30% when implementing a C++ version. In this software, the tolerance parameter is equal to 1%, and the preprocessing step removing hidden cells was not used because it was not available. It makes the algorithm less practical by requiring 12 times as much memory, while the performance gain was only around 30% in Mroz et al. [2000a].

Unfortunately, the Java application was also not able to load volumes greater than $384^3$ voxels, because of the same 2-GB limit per process under Windows XP and represents more than twice the theoretical amount described in Mroz et al. [2000a]. It is important to note that optimized implementations of both algorithms should perform much better in this domain.

Rendering times were measured with different optimizations and are abbreviated according the following nomenclature. *No Optimization* indicates that the visibility of the octree nodes is applied without any further optimization except node sorting. The *G64* and *G32* options indicate the quantization of gray levels used in preclassified renderings. $E1$ indicates that an error of 1 is allowed in preclassified MIP renderings. *MP* stands for the multipass optimization. Finally, *Java App.* indicates the performances of the Java software used for the comparison Mroz et al. [2000a]. Thus, *G64* + *E1* + *MP* is considered as the default setting, providing high-quality fast renderings (also referred as *All optimizations*).

### 6.1 Complexity Analysis

Our first experimental tests aimed to analyze complexity. The UNC Head, Xmas Tree, and Hand datasets were resampled with simple trilinear interpolation at different resolutions from $64^3$ ($m = 6$) to $512^3$ ($m = 9$). We also used two implicitly made multiresolution volumes: a *negative distance to center*

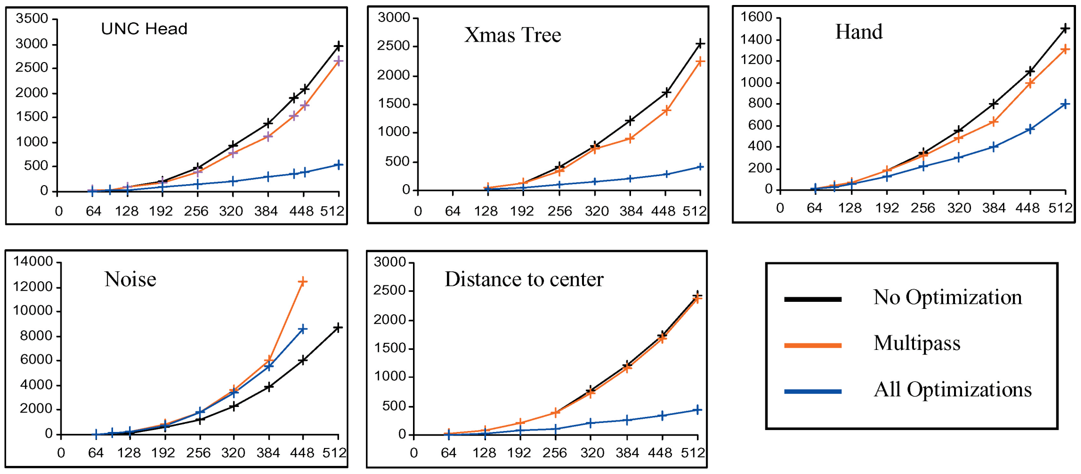


Fig. 6. Evolution of the rendering times (ms) for different datasets with different optimizations according to the data size.

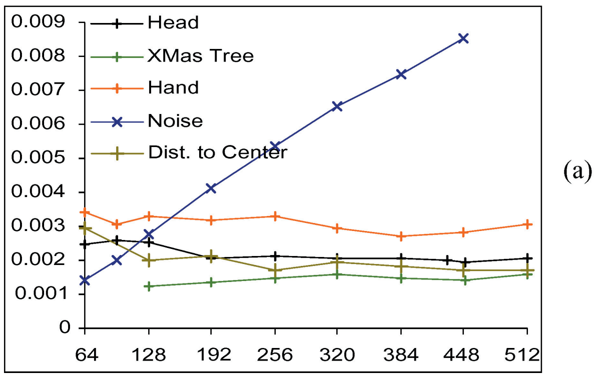
dataset and pure noise volumes, to ensure that trilinearly interpolated rescaling was not biasing the results. The size of a voxel projection was chosen as constant (approximately $0.7 \times 0.7$ pixels to ensure that big volumes would not be rendered off-screen). Three different optimizations were tested: *visibility* and *sorting* only (*no optimization*), *visibility* plus *multipass* (*MP*), and *all optimizations* (*G64* + *E1* + *MP*). Finally, the transfer function was set as a linear function including the entire intensity range. Results are shown in Figure 6.

The first observation is that the quantization of gray levels had a significant impact on rendering times, as theoretically expected, except for the noise dataset that will be discussed latter. The *multipass* optimization seemed also valuable for large regular datasets. In order to determine complexity, we empirically divided these curves by $n^{2+a}$, where $a$ is a number between 0 and 1, and tried to observe a constant behavior. For the *no optimization* and *multipass* rendering cases (excluding the noise dataset), we evaluated $a$ to 0.6 (theoretically around 0.4), but the standard deviation from a perfectly flat curve was probably too much to state this as a result. This is visible in Figure 9 (head 100%), where the green curve and the blue curve, respectively, have an $O(n^3)$ and $O(n^2)$ complexity.

However, the optimization quantifying gray levels better confirmed theoretical results, and the experimental results were coherent enough to state an $n^2$ like average complexity ($a = 0$).

Figure 7(a) shows rendering times with this optimization enabled, divided by $n^2$ (the noise dataset function has also been scaled by a factor of 0.2). One can clearly distinguish the constant behavior of these functions, demonstrating the $O(n^2)$ complexity, except for the noise dataset. This latter result was due to the fact that preclassified renderings of the noise dataset (Figure 7(c)) can provide results close to the worst-case rendering (Figure 7(b)), which occurs when the final image is a blending of interleaved black and white pixels. In this case, the HOM efficiency is null because they are never updated. To be more exact, we noticed that $a = 0.8$ when not using optimizations and, $a = 1$ with optimizations enabled. Therefore, this case tells us that an imperfect visibility test has an impact on performance, but reaching a better percentage for the detection of invisible nodes may be difficult and we currently don't see a better way to do it.

We also tested our algorithm with many other regular datasets, including some renderings that only consider a small range of the volume histogram, but the complexity still remained $O(n^2)$. This was still the case when using 128 gray levels.



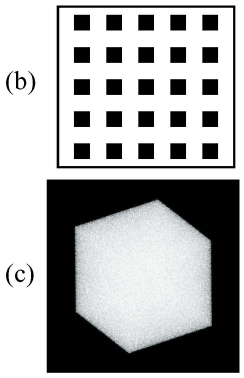


Fig. 7. (a) Rendering times (ms) with all optimizations activated, divided by $n^2$. The noise dataset curve has also been scaled by a factor of 0.2 to fit the diagram. (b) Worst-case rendering scenario. One pixel out of four is black, preventing updating the HOMs. Therefore, every node will be processed as visible. (c) shows that rendering a noise dataset can provide images similar to (b).

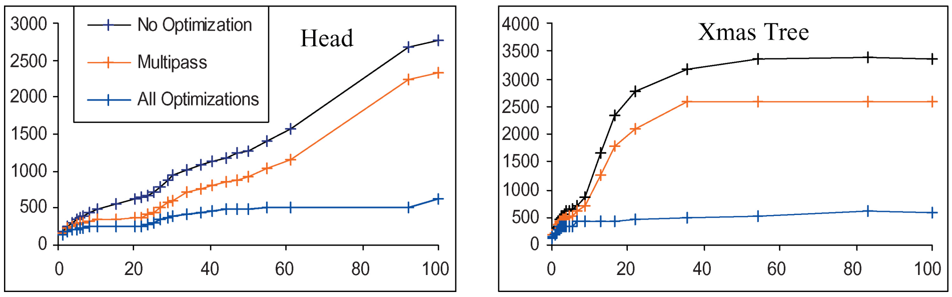


Fig. 8. Evolution of rendering times (ms) according the precentage of cells included by the transfer function for the UNC Head and the Xmas Tree ($512^3$).

In order to show more results, Figure 8 shows the evolution of rendering times according to the percentage of included cells by the transfer function (TF) for the $512^3$ versions of the UNC Head and the Xmas Tree (the minimum value of the TF was scrolled). An interesting fact is that the times were almost constant when using the optimized version of our algorithm; however, transitions were visible within the curves. This phenomenon is explained by looking at the evolution of the image at these transitions. Thus, the rendering footprint grew significantly around these transitions, because new differently located materials were being projected onto the image. When projecting cells of a new material, it is likely that many cells of this new material are going to be projected onto a still black area of the image and therefore many cells won't be hidden and will need to be projected.

## 6.2 Practical Comparison with a State-of-the-Art Algorithm

We compared our algorithm with the Mroz et al. [2000a] approach for completeness. This time, every rendering tried to fit a $512^2$ window, and therefore the zooming parameter was tied to the volume dimensions. Figure 9 shows the evolution of rendering times according to the size, the different optimizations, and percentage of voxels included by the transfer function. Ratios between the Java application Mroz et al. [2000a] and our algorithm with optimal settings are also shown.

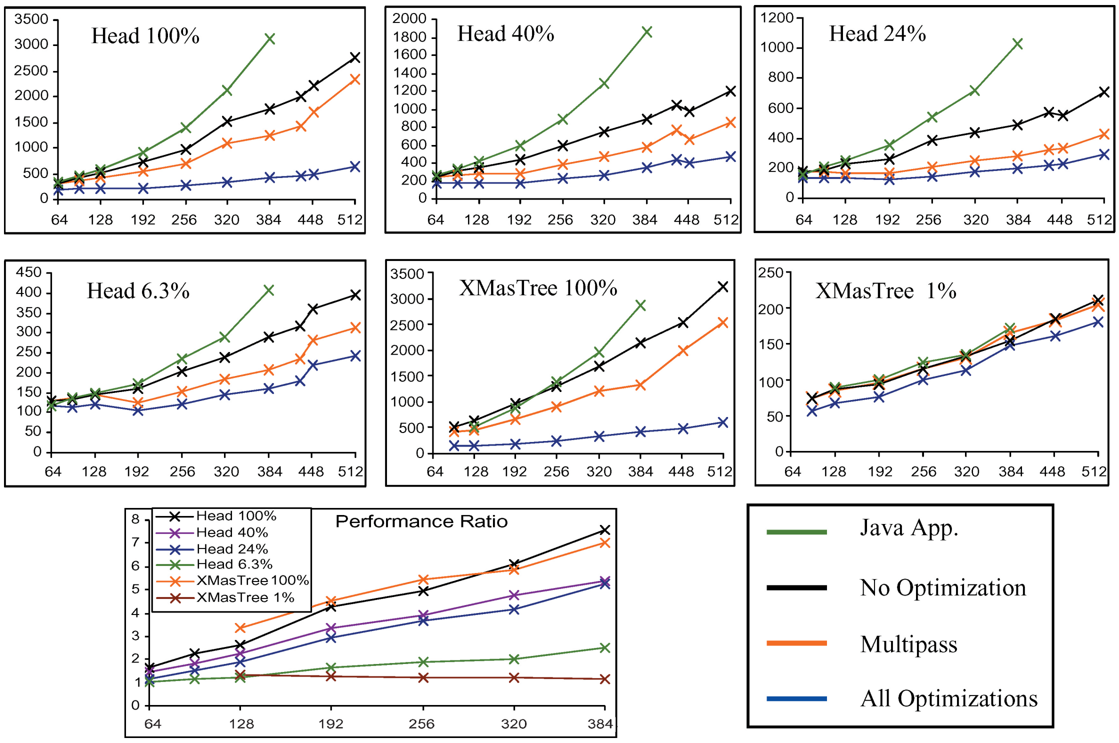


Fig. 9. Rendering times (ms) for the Xmas Tree and UNC Head datasets made from different transfer functions that include different voxel percentage. Performance ratios between the Java application [Mroz et al. 2000a] and our optimized algorithm show a quite linear increase of this ratio according to $n$.

One can clearly distinguish that rendering times for the Java application increased more quickly than those of the different versions of our algorithm. The ratio between algorithm performances shows an approximately linear increase of the speedup, which makes sense if we consider that our optimized algorithm had an average $O(n^2)$ complexity while the Java application had an $O(n^3)$ complexity. We can also notice that the slope was related to the percentage of included voxels by the transfer function. Therefore, there was an approximate sevenfold speedup for rendering a $384^3$ dataset if the TF included all the intensity range, while the speedup was not noticeable if only 1% of the cells were used. To be general, we can say that, the greater the number of projected cells, the better the ratio. Finally, similar performances when rendering few cells indicate that the performance of cell projection was the same for both applications. Besides the fact that we used C and assembly code for this operation instead of a Java interpreter, we also used more projection lists ($16 \times 16$ instead of $4 \times 4$), which affected cache performances, and the Java application could approximate the cell contribution with one bilinear interpolation instead of several trilinear interpolations.

## 6.3 Image Quality

While many techniques sacrifice quality for speed, here the use of object-order ray-casting with trilinear interpolation ensures the high quality of the renderings. However, errors may result from the different optimizations. Figure 10 shows such effects on the UNC Head dataset. Actually, the errors made in all renderings are not noticeable and should not hamper the interpretation of the images. Indeed,

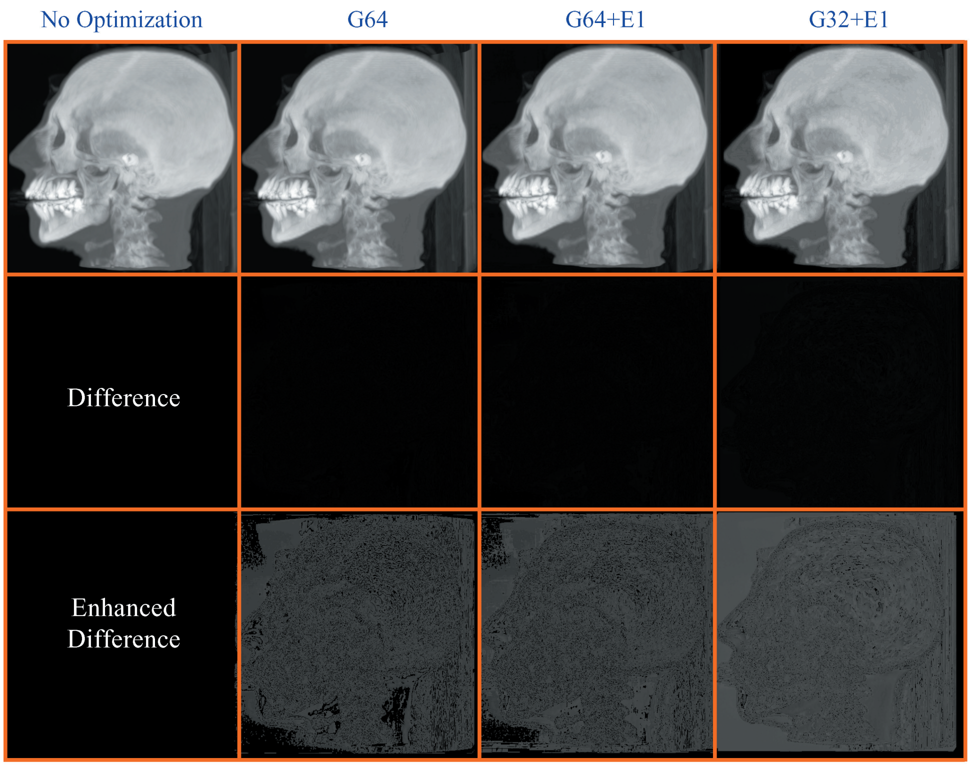


Fig. 10. Effect of reducing gray levels on image quality. The second row shows differences between optimized renderings and the regular rendering, while the third row enhances the brightness of the second raw.

both the *G64* and *G64* + *E1* optimizations are hardly distinguishable in practice and, therefore, are used as the default settings in our application. Only the *G32* + *E1* case shows noticeable errors in the final image, and should be restricted to fast previewing visualization. In cases where highly detailed images are needed, a 128, 256, or higher gray-level quantization can be used, which will slow the rendering performance but ensure artifacts are not visible for most applications with still an $O(n^2)$ average complexity.

## 7. CONCLUSION

Since maximum intensity projection is very widely used in the medical community, and sizes of datasets are always growing, improving MIP rendering is still very important. We have presented a major advance in this domain by demonstrating that the MIP complexity can be reduced to an average $O(n^2)$ complexity with the use of octrees and an $n$-bit gray-level quantization. We have also proposed and thoroughly described a high-quality object-order implementation that is significantly faster than an image-order approach. The results clearly confirm the theoretical improvement gained by our algorithm, showing a linear speedup compared to a state-of-the-art approach. Our algorithm is also very flexible, allowing, for example, rendering of floating point volumes and applying any transfer function.

Other methods to achieve a low-complexity for MIP renderings may exist and we think that more research in MIP visualization should be thoroughly investigated in the future. Particularly, performing better on-the-fly visibility, improving the worst-case complexity, and, having a perspective projection would be valuable.

Finally, our complexity analysis can be useful for proving the complexity of other volume rendering models, such as isosurface renderings. Our case strongly suggests that the combination of space-leaping and occlusion-culling inside an object-order algorithm may have a better complexity than space-leaping and early ray-termination inside an image-order algorithm.

## APPENDIX

### A. PROOF OF THE COMPLEXITY FOR $S_k$(m)

In order to prove the complexity of $S_k(m)$, we have to enclose this function between two functions that differ from $4^{m/3}$ by a constant factor.

From the property

$$\int_{a+i}^{a+i+1} \frac{dx}{x} \leq \frac{1}{a+i} \leq \int_{a+i-1}^{a+i} \frac{dx}{x} \tag{18}$$

implying

$$\frac{1}{k} \int_{\frac{4^l}{k}+i}^{\frac{4^l}{k}+i+1} \frac{dx}{x} \leq \frac{\frac{1}{k}}{\frac{4^l}{k}+i} \leq \frac{1}{k} \int_{\frac{4^l}{k}+i-1}^{\frac{4^l}{k}+i} \frac{dx}{x} \tag{19}$$

we get (by summing on $i$)

$$\frac{1}{k} \int_{\frac{4^l}{k}}^{\frac{4^l}{k}+k\cdot 2^{m-l}} \frac{dx}{x} \leq \sum_{i=0}^{k\cdot 2^{m-l}-1} \frac{1}{4^l+k\cdot i} \leq \frac{1}{4^l} + \frac{1}{k} \int_{\frac{4^l}{k}}^{\frac{4^l}{k}+k\cdot 2^{m-l}-1} \frac{dx}{x} \tag{20}$$

implying

$$\frac{1}{k} \int_{\frac{4^l}{k}}^{\frac{4^l}{k}+k\cdot 2^{m-l}} \frac{dx}{x} \leq \sum_{i=0}^{k\cdot 2^{m-l}-1} \frac{1}{4^l+k\cdot i} \leq \frac{1}{4^l} + \frac{1}{k} \int_{\frac{4^l}{k}}^{\frac{4^l}{k}+k\cdot 2^{m-l}} \frac{dx}{x}. \tag{21}$$

By integrating and summing on $l$, we get a bounded estimation of the result $S_k(m)$:

$$\begin{cases} S_k(m) \geq \dfrac{1}{k} \displaystyle\sum_{l=0}^{m} 4^l \ln\left(1+k^2 2^{m-3l}\right), \\ S_k(m) \leq m+1+\displaystyle\sum_{l=0}^{m} 4^l \ln\left(1+k^2 2^{m-3l}\right). \end{cases} \tag{22}$$

Now we have both to overestimate and to underestimate the term $T_k(m)$ in order to prove complexity (notice here that $(m+1)$ will be negligible compared to the order of magnitude of $S_k(m)$):

$$T_k(m) = \sum_{l=0}^{m} 4^l \ln\left(1+k^2 2^{m-3l}\right). \tag{23}$$

### A.1 Overestimation

From the properties

$$\forall x \;\geq\; 0,\;\; \ln(1+x) \leq \sqrt{x}, and \;\; \ln(1+x) \leq x, \tag{24}$$

$$T_k(m) \;\leq\; \sum_{0 \leq l \leq \frac{m}{3}} 4^l \ln(1+k^2 2^{m-3l}) + \sum_{\frac{m}{3} < l \leq m} 4^l \ln(1+k^2 2^{m-3l}), \tag{25}$$

we deduce

$$T_k(m) \;\leq\; \sum_{0 \leq l \leq \frac{m}{3}} k \cdot 2^{2l} 2^{\frac{m-3l}{2}} + \sum_{\frac{m}{3} < l \leq m} k^2 2^{2l} 2^{m-3l} \tag{26}$$

$$\Rightarrow\; T_k(m) \leq \mathrm{k} \cdot 2^{\frac{m}{2}} \sum_{0 \leq l \leq \frac{m}{3}} 2^{\frac{l}{2}} + k^2 2^m \sum_{\frac{m}{3} < l \leq m} 2^{-l} \tag{27}$$

$$\Rightarrow\; T_k(m) \leq \mathrm{k} \cdot 2^{\frac{m}{2}} \left( \frac{\left(2^{0.5}\right)^{\frac{m}{3}+1} - 1}{2^{0.5} - 1} \right) + k^2 2^m 2^{-\frac{m}{3}} \left(1 - 2^{-\frac{2m}{3}}\right) \tag{28}$$

$$\Rightarrow\; T_k(m) \leq \; a \cdot 2^{\frac{2m}{3}} \text{ - } a' 2^{\frac{m}{2}}, \tag{29}$$

with $a$ and $a'$ both positive.

### A.2 Underestimation

From the property

$$\forall x \geq 0, \forall k \geq 1,\;\; \ln\left(1+k^2 x\right) \;\geq \ln(1+x) \;\geq\; x - \frac{x^2}{2} \tag{30}$$

we get

$$T_k(m) \;\geq\; \sum_{l=0}^{m} 4^l \ln\left(1+2^{m-3l}\right) \geq \sum_{\frac{m}{3} < l \leq m} 4^l \left(2^{m-3l} \;-\; \frac{4^{m-3l}}{2}\right), \tag{31}$$

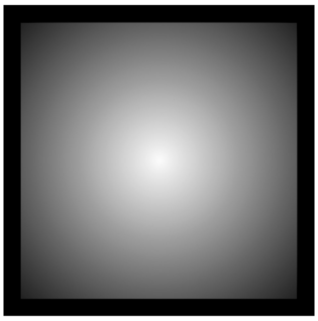
$$\Rightarrow\; T_k(m) \;\geq\; 2^m \sum_{\frac{m}{3} < l \leq m} 2^{-l} - \frac{4^m}{2} \sum_{\frac{m}{3} < l \leq m} 16^{-l}. \tag{32}$$

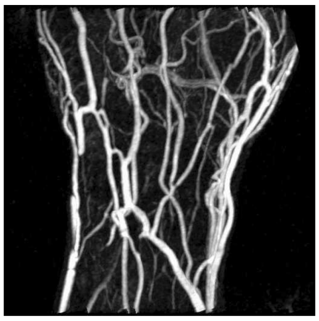
By developing the two geometric sums (not shown here), we are able to obtain

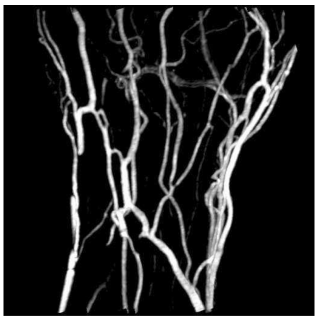
$$T_k(m) \;\geq\; a'' \cdot 2^{\frac{2m}{3}} \tag{33}$$

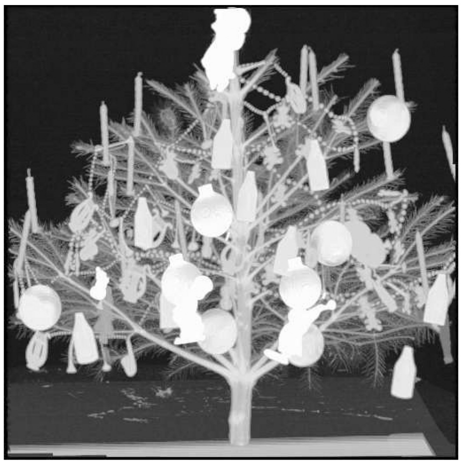
with $a''$ positive.

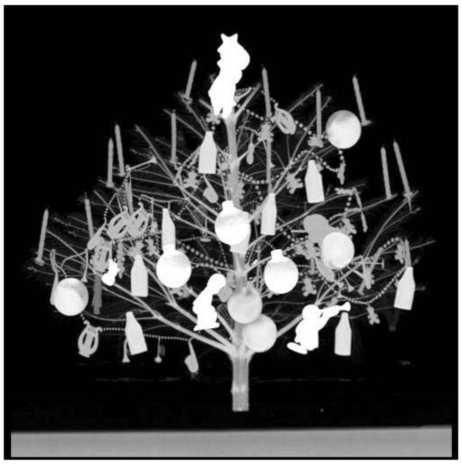
## B. RENDERINGS

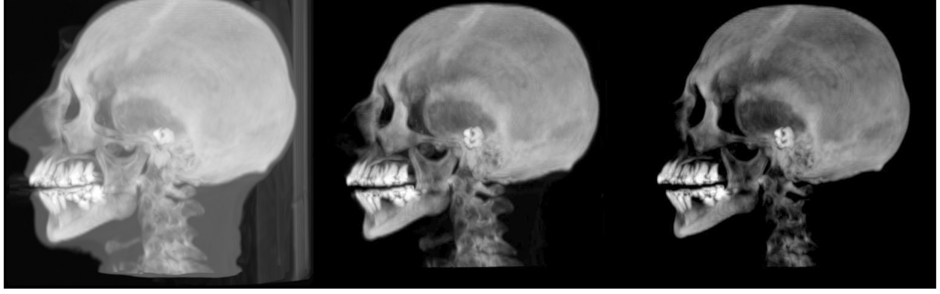

Distance to center
100%


Hand
100%


Hand
1%


Xmas Tree
83%


Xmas Tree
2%


UNC Head
100%

UNC Head
16%

UNC Head
4%

ACKNOWLEDGMENTS

We thank Lukas Mroz for kindly providing a version of the Java renderer we compared to our system. We thank Kirk Riley, Grégoire Moncouqiol, and Céline Moine for assistance with revising this manuscript and the anonymous reviewers for their helpful, thorough comments that have improved the final manuscript.

The CT-dataset is from the University of North Carolina, Chapel Hill. The Xmas Tree was generated from a real-world Christmas tree by the Department of Radiology, University of Vienna, and the Institute of Computer Graphics and Algorithms, Vienna University of Technology. This article is based upon work supported by the National Science Foundation (grant numbers 0222675, 0081581, 0098443, 0121288, 0196351, 9978032, and 0328984).

## REFERENCES

Brodlie, K. and Wood, J. 2001. Recent advances in volume visualization. *Comput. Graph. Forum 20*, 2 (June), 125–148.

Cabral B., Cam, N., and Foran, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 Symposium on Volume Visualization*. 91–98.

Cai, W. and Sakas, G. 1998. Maximum intensity projection using splatting in sheared object-space. In *Proceedings of EUROGRAPHICS'98*. 113–124.

Csébfalvi, B., König, A., and Gröller, E. 1999. Fast maximum intensity projection using binary shear-warp factorisation. In *Proceedings of WSCG'99*. 47–54.

Glassner, A. S. 1995. *Principles of Digital Image Synthesis*. Morgan Kaufmann, San Francisco, CA.

Heidrich, W., McCool, M., and Stevens, J. 1995. Interactive maximum projection volume rendering. In *Proceedings of IEEE Visualization'95*. 11–18.

Kim, K. H. and Park, H. W. 2001. A fast progressive method of maximum intensity projection. *Comput. Med. Imaging Graph. 25*, 3, 433–441.

Lacroute, P. and Levoy, M. 1994. Fast volume rendering using a shear-warp transformation of the viewing transform. In *Proceedings of ACM SIGGRAPH'94*. 451–459.

Lee, R. K. and Ihm, I. 2000. On enhancing the speed of splatting using both object-and-image space coherence. *Graph. Models Image Process. 62*, 4, 263–282.

Levoy, M. 1990. Efficient ray tracing of volume data. *ACM Trans. Graph. 9*, 3, 245–261.

Malzbender, T. 1993. Fourier volume rendering. *ACM Trans. Graph. 12*, 3, 233–250.

Mora, B., Jessel, J. P., and Caubet, R. 2000. Accelerating volume rendering with quantized voxels. In *Proceedings of IEEE/ACM SIGGRAPH Visualization and Graphics Symposium*. 80–87.

Mora, B., Jessel, J. P., and Caubet, R. 2002. A new object-order ray-casting algorithm. In *Proceedings of IEEE Visualization'2002* (Boston, MA, October). 203–210.

Mroz, L., Hauser, H., and Gröller, E. 2000a. Interactive high-quality maximum intensity projection. In *Proceedings of EUROGRAPHICS'2000*. 341–350.

Mroz, L., König, A., and Gröller, E. 2000b. Maximum intensity projection at warp speed. *Comput. Graph. 24*, 3 (June), 343–352.

Parker, S., Parker, M., Livnat, Y., Sloan, P-P., Hansen, C., and Shirley, P. 1999. Interactive ray tracing for volume visualization. *IEEE Trans. Vis. Comput. Graph. 5*, 3, 238–250.

Pekar V., Hempel, D., Kiefer, G., Busch M., and Wees, J. 2003. Efficient visualization of large medical image datasets on standard PC hardware. In *Proceedings of VisSym'2003*. 135–140.

Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H., and Seiler, L. 1999. The volume pro real-time ray-casting system. In *Proceedings of ACM SIGGRAPH'99*. 251–260.

Resk-Salama, C., Engel, K., Bauer, M., Greiner, G., and Ertl, T. 2000. Interactive volume rendering on standard PC hardware platform using multi-texture and multistage rasterization. In *Proceedings of EUROGRAPHICS/SIGGRAPH Workshop on Graphics Hardware*. 109–118.

Revelles, J., Ureña, C., and Lastra, M. 2000. An efficient parametric algorithm for octree traversal. In *Proceedings of WSCG'2000*. 212–219.

Roerdink, J. B. T. M. 2001. Multiresolution maximum intensity volume rendering by morphological pyramids. In *Proceedings of VisSym'01*. 45–54.

Sakas, G., Grimm, M., and Savopoulos, A. 1995. Optimized maximum intensity projection. In *Proceedings of the 5th EUROGRAPHICS Workshop on Rendering Techniques*. 55–63.

Sato, Y., Shiraga, N., Nakajima, S., Tamura, S., and Kikinis, R. 1998. Local maximum intensity projection (LMIP): A new rendering method for vascular visualization. *J. comput. Assist. Tomogr. 22*, 6, 912–919.

Westover, L. 1990. Footprint evaluation for volume rendering. In *Proceedings of ACM SIGGRAPH'90*. 367–376.

Wilhelms, J. and Van Gelder, A. 1992. Octrees for faster Isosurface generation. *ACM Trans. Graph. 11*, 3 (July), 201–227.

Zhang, H., Manocha, D., Hudson, T., and Hoff, K. E. 1997. Visibility culling using hierarchical occlusion maps. In *Proceedings of ACM SIGGRAPH'97*. 77–88.

Zuiderveld, K. J., J. Koning, A. H., and Viergever, M. A. 1994. Techniques for speeding up high-quality perspective maximum intensity projection. *Patt. Recogn. Lett. 15*, 507–517.

Received July 2004; revised January 2005, May 2005; accepted July 2005